**Distributed Computing: Fall 2016**

**Programming Assignment #1**

**JAVA RMI xotd Service**

**-Implementation Document-**

2013010959

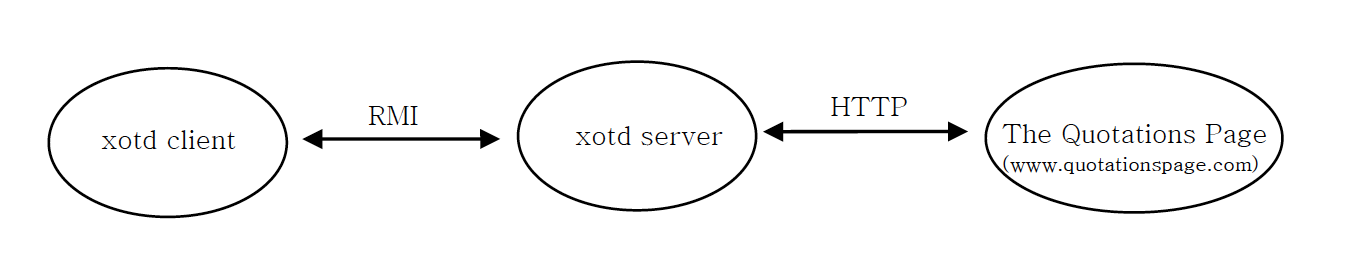
컴퓨터공학부 소프트웨어 전공

곽호진

**-Contents-**

1. **과제 설명**
   1. **과제에 대하여**
   2. **개발환경**
2. **디렉터리 구조**
3. **실행 과정**
   1. **컴파일 단계**
   2. **서버 사이드 실행**
   3. **클라이언트 실행**
4. **실행 결과**
   1. **Qotd**
   2. **Wotd**
5. **느낀점**
6. **과제 설명**
   1. **과제에 대하여**

이번 과제의 목표는 오늘의 인용구나 단어를 화면에 표시해 주는 xotd 서비스를 “JAVA RMI”를 이용하여 구현하는 것이다. 인용구나 단어를 표시하고 HTML을 파싱하는 것 보다 JAVA RMI를 사용해보는 경험이 본 과제의 주요 목표 이다. 아래는 과제 설명서에 첨부된 xotd 서비스의 구조이다.



제공될 RMI interface는 다음의 두 메서드를 제공해야 한다.

* String getQotd() : 오늘의 인용구를 제공하는 메서드이다.
* String getWotd() : 오늘의 단어를 제공하는 메서드 이다.
  1. **개발 환경**

OS: Windows(Parallels), maxOS Sierra

IDLE: Eclipse, vi

Java version: 1.8.0\_73

1. **디렉터리 구조**

**rmi-GwakHoJin/** : 전체 프로젝트 폴더

**ㄴsrc/** : 소스 코드가 들어있는 폴더

**ㄴclient/** : client package

**ㄴXotdClientMain.java** : client의 수행 코드

**ㄴ server/** : server package

**ㄴ ServerMain.java :** server의 수행 코드

**ㄴXotdServiceImpl.java** : XotdService interface의 구현 코드

**ㄴ xotd/ : xotd package**

**ㄴ XotdService.java** : xotdService interface

**ㄴ bin/** : 컴파일 된 바이너리파일 (.class)가 위치할 디렉터리

**ㄴ client/** : client 의 바이너리가 저장될 디렉터리

**ㄴ server/** : server의 바이너리, XotdService의 구현의 바이너리가 저장될 디렉터리

**ㄴ xotd/** : xotdService interface의 바이너리가 저장될 디렉터리

**ㄴ doc/** : 구현 문서가 저장될 디렉터리

**ㄴ rmi\_doc-GwakHoJin.docx** : 본 문서

**ㄴlib/** : 외부 라이브러리가 저장될 디렉터리

**ㄴ jsoup-1.10.1.jar** : html parsing을 위한 외부 라이브러리

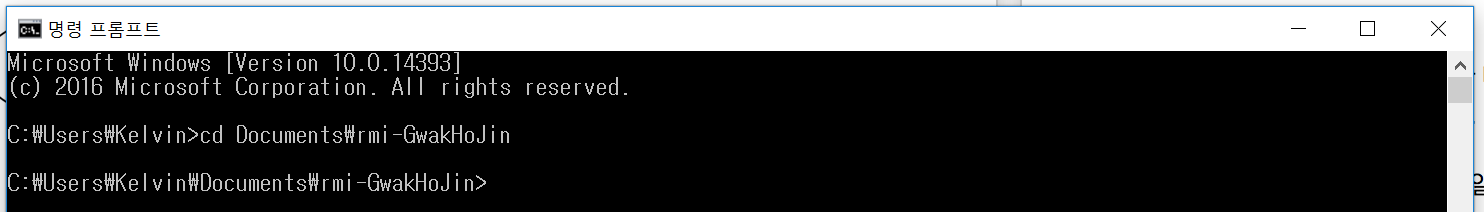
**ㄴ security.policy** : RMISecurityManager를 위한 정책 파일

1. **실행 과정**
   1. **컴파일 단계**

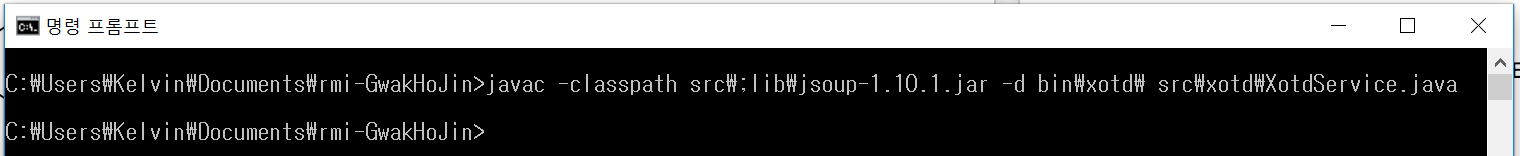
본 과제의 컴파일은 IDLE를 사용하지 않고 terminal, cmd를 이용하여 컴파일 하는 것을 전제로 한다. 컴파일과정과 실행과정 모두 프로젝트 폴더의 최상위에서 진행되며 디렉터리 이동을 하지 않는다.

[Windows에서의 컴파일]

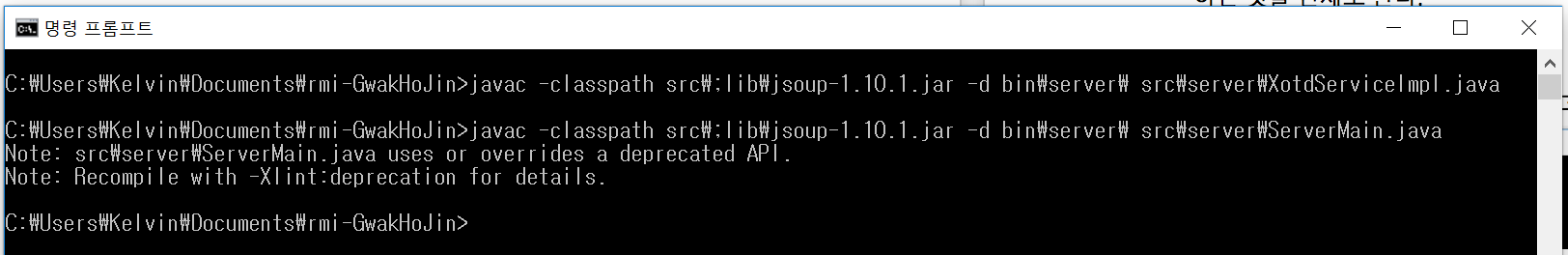
1. cmd를 실행하고 프로젝트 폴더로 이동한다.



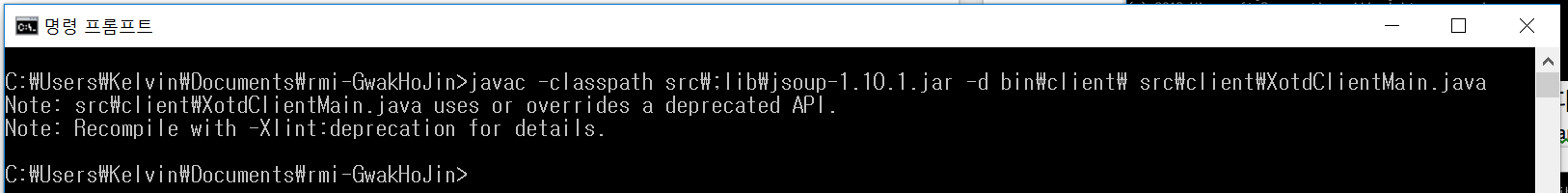
1. xotd 인터페이스를 컴파일 한다.  
   javac -classpath src\;lib\jsoup-1.10.1.jar -d bin\xotd\ src\xotd\XotdService.java



1. 서버 사이드를 컴파일 한다.  
   javac -classpath src\;lib\jsoup-1.10.1.jar -d bin\server\ src\server\XotdServiceMain.java  
   javac -classpath src\;lib\jsoup-1.10.1.jar –d bin\server\ src\server\ServerMain.java

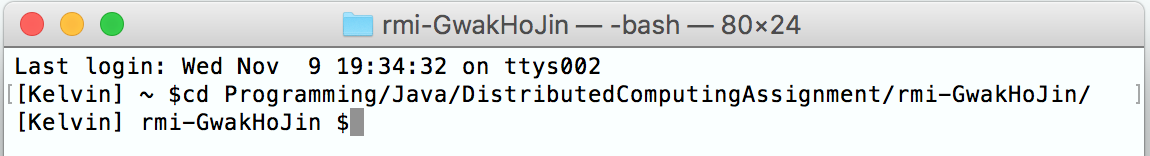


1. 클라이언트 사이드를 컴파일 한다.  
   javac -classpath src\;lib\jsoup-1.10.1.jar -d bin\client\ src\client\XotdClientMain.java



[Ubuntu(Linux), macOS 에서의 컴파일]

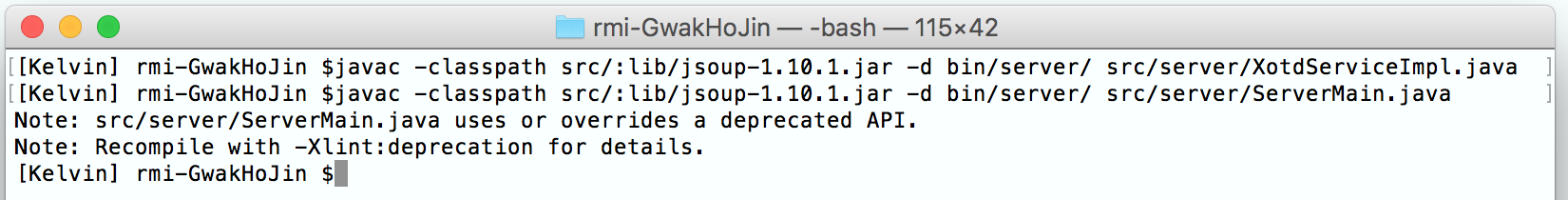
1. terminal을 실행하고 프로젝트 폴더로 이동한다.



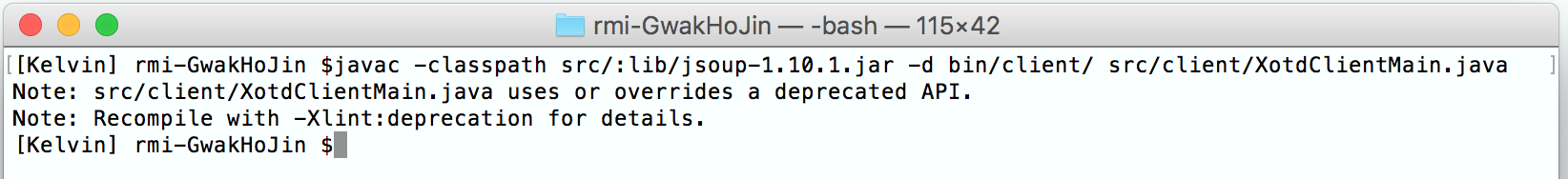
1. xotd 인터페이스를 컴파일 한다.  
   javac -classpath src/:lib/jsoup-1.10.1.jar -d bin/xotd/ src/xotd/XotdService.java



1. 서버 사이드를 컴파일 한다.  
   javac -classpath src/:lib/jsoup-1.10.1.jar -d bin/server/ src/server/XotdServiceMain.java  
   javac -classpath src/:lib/jsoup-1.10.1.jar –d bin/server/ src/server/ServerMain.java



1. 클라이언트 사이드를 컴파일 한다.  
   javac -classpath src/:lib/jsoup-1.10.1.jar -d bin/client/ src/client/XotdClientMain.java

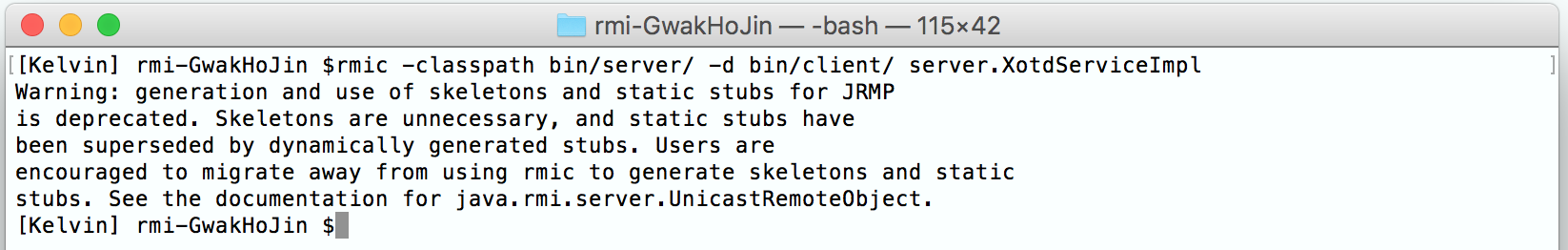


사실 Windows와 Linux계열에서의 컴파일의 차이점은 이용하는 프로그램이 cmd냐 terminal이냐의 차이, classpath 를 통해 전달될 인자의 구분자가 세미콜론(;)이나 콜론(:)이냐의 차이, 디렉터리 구분자가 역슬래시(\)냐 슬래시(/)냐 정도밖에 없다.

* 1. **서버 사이드 실행**

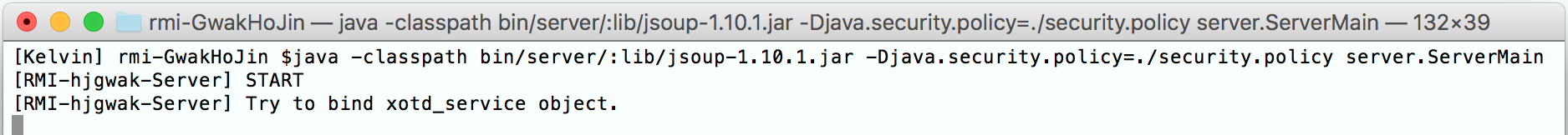
1. rmic를 이용하여 sub파일 만들기

rmic -classpath bin/server/ -d bin/client/ server.XotdServiceImpl



1. 서버 메인 실행하기

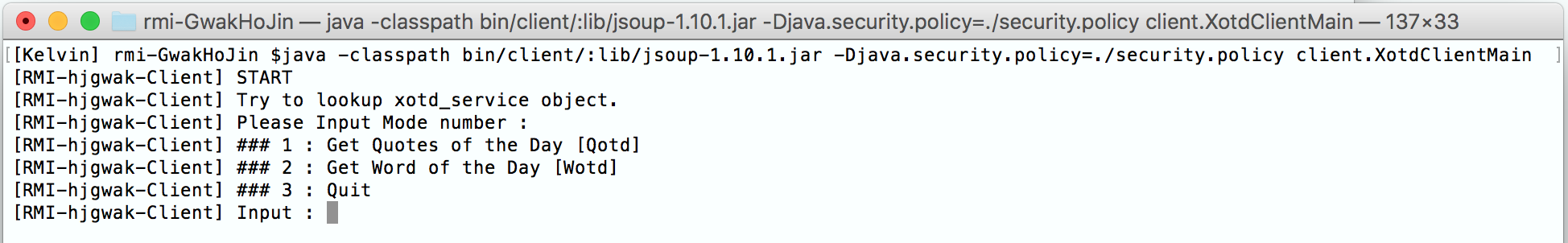
java -classpath bin/server/:lib/jsoup-1.10.1.jar -Djava.security.policy=./security.policy server.ServerMain



* 1. **클라이언트 사이드 실행**

1. 클라이언트 메인 실행하기

java -classpath bin/client:lib/jsoup-1.10.1.jar -Djava.security.policy=./security.policy client.XotdClientMain

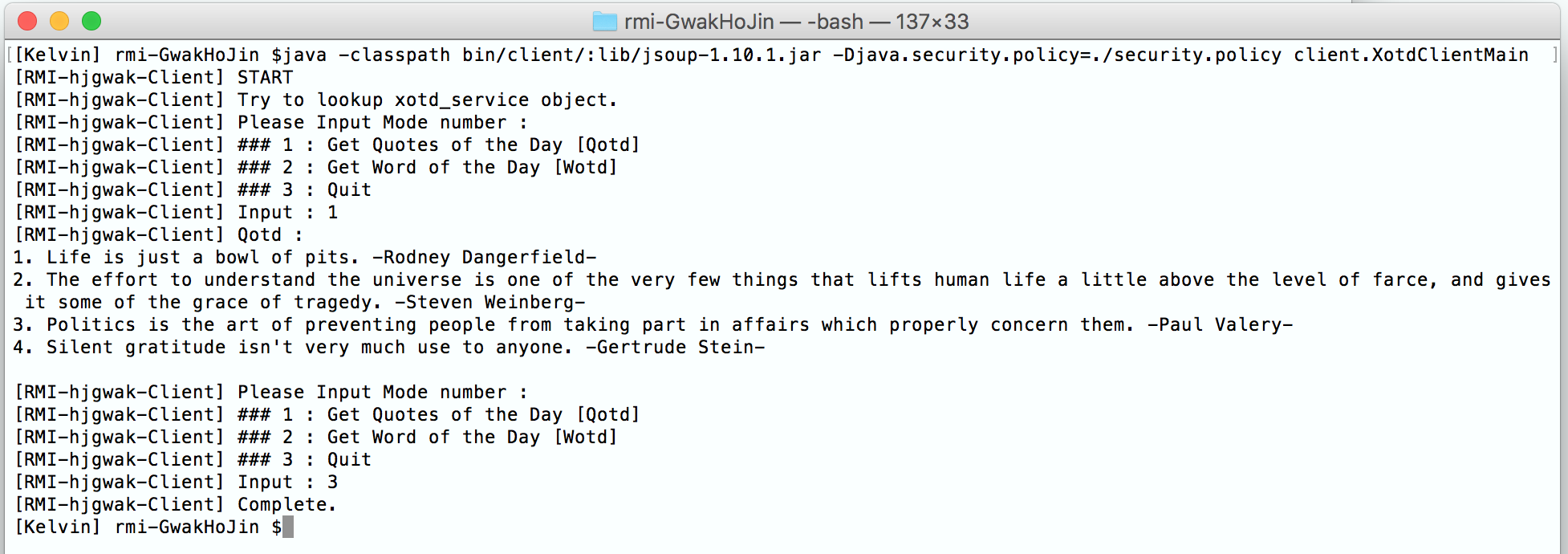


1. **실행 결과**

client 프로그램의 경우 lookup을 통해 RemoteObject를 가져온 뒤, 제대로 가져왔을 경우 사용자의 입력을 받는 모드로 넘어간다. 입력은 숫자로 이루어지며, 1을 입력했을 경우 Quotes of the Day를 출력하며, 2를 입력했을 경우 Word of the Day를 출력한다. 3을 입력했을 경우 프로그램을 종료하며, 다른 숫자를 입력했을 경우 입력을 다시 기다린다. 사용자가 3을 입력하기 전까지 프로그램은 강제종료를 제외하고 종료되지 않고 입력된 명령을 수행 후 다시 사용자의 입력을 기다린다..

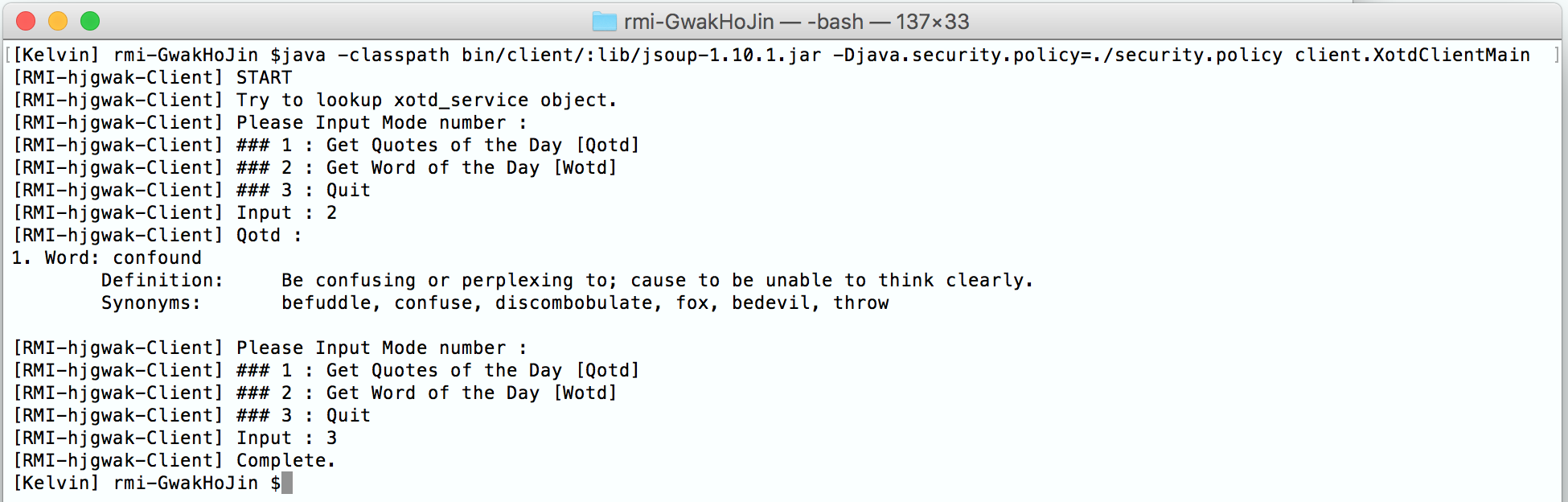
* 1. **Qotd**

현재 Quotationspage 홈페이지가 정상운영되고 있지 않다. <http://www.quotationspage.com/qotd.html>을 이용할 경우 아무런 인용구가 출력되지 않으며, <http://www.quotationspage.com/qotd/previous.html> 를 이용할 경우 인용구가 출력된다. 현재 코드에선 previous.html을 통하도록 프로그래밍 되어있으며, qotd.html을 통하는 코드는 주석처리 되어있다. 필요에 따라 수정 후 컴파일하여 수행할 수 있다.

****

* 1. **Wotd**

현재 WordOfTheDay 페이지의 경우, 정적인 페이지를 로드 후 javascript를 이용하여 TheFreeDictionay 페이지의 정보를 동적으로 로드하는 형식으로 구현되어있다. 때문에 java에서 jsoup를 통한 html통신시 javascript가 수행되지 않아서 정적인 페이지만 받아올 수 있다. 우선은 정적페이지를 파싱하는 것으로 과제를 구현하였다.



1. **느낀점**

이번 프로젝트를 하기 전 Java를 제대로 써본 적이 한 번도 없었다. 따라서 이번 과제가 내 첫 Java 프로젝트 였다. Package, 컴파일 옵션등 Java의 많은 것들이 생소함 투성이었다.

우선 처음으로 놀란 것은 Java에서 정말 많은 것들이 기본 패키지로 제공된다는 것이었다. RMI 패키지만 보아도 알 수 있었다. 그리고 모든 것이 Object로 구성되고 서로간의 상속을 통해 구현되는 점에서 처음에는 C++에 익숙하던 내가 받아들이기 힘들었지만, 점점 Java만의 특성과 색을 알아갈 수 있었다.

Java에서 편리하게 느꼈던 것이 바로 package이다. Package를 통해서 서로 관련있는 소스들을 묶어서 관리할 수 있고, 간편하게 import할 수 있는 것도 매우 편리했다. C++를 이용할 땐 직접 디렉토리를 나누고 구분했어야 하는 것과 비슷한 개념으로 다가왔었는데 Java에선 조금 더 편리한 기능들을 지원하고 있었다. 하나의 큰 프로젝트에 여러개의 메인함수가 존재해도, package를 통해 구분할 수 있었고 심지어 실행할 때도 package를 통해 구분하여 실행할 수 있으니 메인함수마다 프로젝트를 새로 만드는 번거로움을 회피할 수 있었다.

RMI에 대해선 더욱 더 생소했었다. 과제를 시작하기 전 간단한 RMI 예제를 따라해 보며 RMI에 대한 감을 잡았었는데, 처음 컴파일에 성공하기 까지 무려 14시간이 걸렸다. 물론, Java에 대한 지식이 없던 것도 컴파일 성공 시간을 늘리는 것에 한몫 했다. RMI에 대해 자세하게 이해할 수 있도록 도와준 것은 컴파일 과정에서 마주했던 무수한 에러들이었다. 여러 에러메세지 중, 클라이언트에서 RMI를 호출하고 그 메세지가 TCP를 통해 서버까지 도달하는 과정의 스택이 출력된 에러메세지가 있었다. 그 메세지를 읽어보면서 RMI 메세지가 클라이언트부터 서버까지 전달되는 과정을 알 수 있었다.