Minor Assignment 2

By Ronald W. Ritchey
An assignment for SWE 781
Submitted on Sept 27th, 2007

Introduction

This assignment is meant to demonstrate how a program can use regular expressions to perform comprehensive input validation. The specific requirement is to produce a program that implements a telephone directory. This directory must allow new names and phone numbers to be posted to the directory, allow entries to be deleted, and to produce a listing of the directory contents. *The program must only accept valid values for all input.* This includes properly validating names and telephone numbers as well as all other input including menu selections. The assignment optionally included a requirement to store and retrieve the directory to disk so that the contents of the directory persists across executions of the program.

Design

This response to the requirements is a text-based application implemented in Java. It implements four key behaviors: add directory entries, delete directory entries, list directory entries, and long-term storage/retrieval. The following section describes how the application is organized and highlights some of the unique features of the design.

High Level Design

The application is implemented using four main classes. Two classes are used to validate, parse, and store names and telephone numbers. Another class is used to bind names and telephones into a single directory entry. The last class implements the behavior of the application including implementing the user interface and the storage/retrieval of the directory from the disk. The high level diagram of the design is shown in Figure 1.

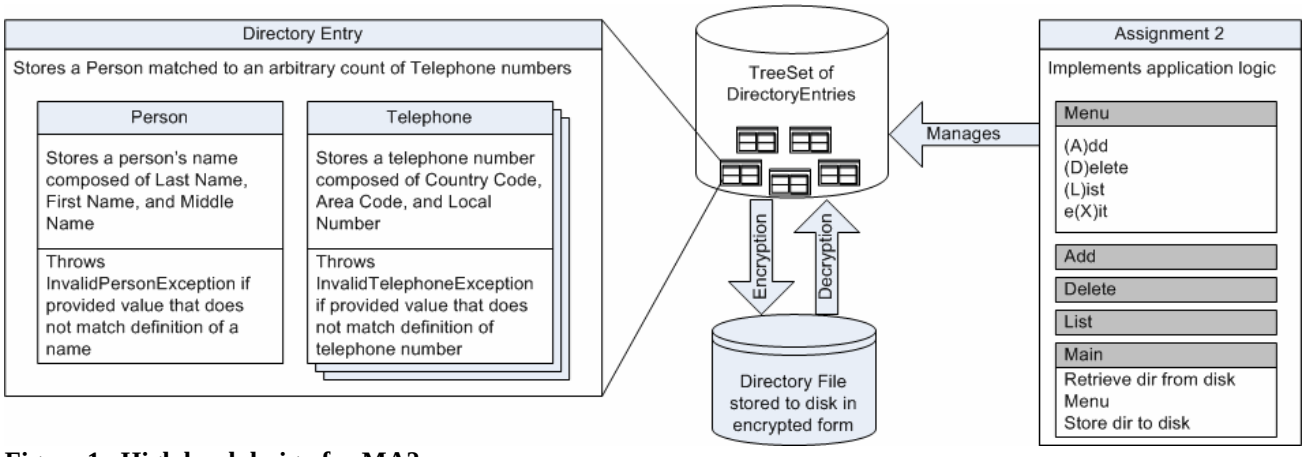

Figure 1. High level design for MA2

The application implements four main commands.

Table 1. MA2 Commands

Command	Accepts	Action/Behavior
Add	Name and Telephone Number	Adds a new entry to the directory or if the person already exists in the directory adds a new telephone number to the persons entry
Delete	Name or Telephone Number	Deletes an entry from the directory or remove a telephone number from an entry with multiple telephone numbers
List	N/A	Produces a listing of People and telephone numbers ordered by lastname, firstname middle name
Exit	N/A	Writes encrypted directory to disk and exits

Validation System

The application relies upon the Telephone and Person classes to perform the major validation functions. Upon instantiation, both classes apply regular expressions to the constructor arguments to see if they represent valid input for the class. If the input looks valid, the classes then parse the input. In the case of Person, the input is parsed into first name, middle name, and last name. At present, suffixes and prefixes are not support. In the case of Telephone, the input is parsed into country code, area code and local number. Two exception classes have been created to record when invalid input is received. These are InvalidPersonException and InvalidTelephoneException. Should either the initial input format check fail, or the parsing fail Person and Telephone will throw their respective exceptions to indicate that invalid input has been received and the class can not be properly instantiated.

It should also be noted that once a valid Telephone or Person has been created, the parsing they perform internally basically normalizes the data. This allows names and telephone numbers to be compared based upon value even when the exact input format is different. For instance if “Ron Ritchey” and “Ritchey, Ron” were used to instantiate two new Person objects, the objects would compare as equals. Other benefit of this is sort order which for Person is Last Name, First Name, Middle Name and for Telephone is Country Code, Area Code, Local Number.

Regular expressions are relied upon heavily to perform validation of the telephone numbers and names that the application accepts. The regular expressions and examples of what the accept and reject are shown in tables 2 and 3.

Table 2. Regex used to validate Names

Regex for Name	

`namePattern = "[a-zA-Z]+('[a-zA-Z]+)?(-[a-zA-Z]+('[a-zA-Z]+)?)?"`
`FirstMidLastPattern = "^("+namePattern+" ){0,2}"+namePattern+"$"`
`LastMidFirstPattern = "^" + namePattern + ", ?(" + namePattern +" )?" +`
`  namePattern + "$"`
`FullNamePattern = "("+FirstMidLastPattern+")|("+LastMidFirstPattern+")`	

Accepted	**Rejected**
Value / **Results**	
Ron Ritchey → Ritchey, Ron Ritchey, Ron → Ritchey, Ron Ritchey, Ron Wayne → Ritchey, Ron Wayne O'Malley, John F. → O'Malley, John F. John O'Malley-Smith → O'Malley-Smith, John Cher → Cher	Ron O''Henry Ron O'Henry-Smith-Barnes L33t Hacker <Script>alert("XSS")</Script> Brad Everett Samuel Smith select * from users;

Table 3. Regex used to validate Telephone numbers

Regex for Telephone		

`telLocal = "[1-9][0-9 ,.\\-/]{4,8}$"`
`telAccess = "((011[ \\-,./])|(00[ \\-,./]?)|(\\+))"`
`telTrunk = "([01][ \\-,./]?)?"`
`telCountry = "(([1-9][0-9]{0,2})|(\\([1-9][0-9]{0,2}\\)))[ ,.\\-/]?"`
`telAreaCode = "\\([1-9][0-9]{0,5}\\)[ ,.\\-/]?"`
`FullTelPattern = "(("+telAccess + telCountry + telAreaCode + ")|(" +`
`  telTrunk + telAreaCode + "))?" + telLocal`		

Accepted		**Rejected**
Value	**Resulst**	
12345 → 12345 (703)111-2121 → (703) 111-2121 123-1234 → 123-1234 +1(703)111-2121 → +1 (703) 111-2121 +32 (21) 212-2324 → +32 (21) 212-2324 1(703)123-1234 → (703) 123-1234 011 701 111 1234 → +701 111 1234 12345.12345 → (12345) 12345 011 1 703 111 1234 → +1 (703) 111 1234		123 1/703/123/1234 Nr 102-123-1234 <script>alert("XSS")</script> 7031111234 +1234 (201) 123-1234 (001) 123-1234 +01 (703) 123-1234 (703) 123-1234 ext 204

There is still room for improvement in both of these regex systems. For names, implementing suffix and prefixes would enable a broader set of names to be entered (e.g. Ron Ritchey, Ph.D.)! Number is a harder challenge to extend. US phone numbers are relatively well structured and would have made this assignment significantly easier to deploy. International numbers and the standards commonly in use to write them vary broadly depending upon the region of the world that they refer to. The variance occurs not only country by country, but also at the city level. For instance, in Belgium local numbers can contain between 6 and 7 digits depending upon the city. These variances make the creation of a single regex expression difficult. The regex that has been provided should except the majority of telephone number formats in use while providing at least some valid method to accept all telephone numbers world wide. There are two obvious opportunities for improvement though. First, the ability to record extensions

would be useful (e.g. (703) 123-1234 ext 204). The second is the ability to mark what type of number it is (e.g. home, work, mobile, fax).
The remaining input validation for user input is for the menuing system. This routine allows up to 100 characters to be entered but only considers the first character for evaluation. This is compared against a list of valid characters. If a valid character is entered it is returned to the calling function. If not, an InvalidInputException is thrown.

Storage

An additional feature of the system is secure storage of the directory entries on disk. Upon start-up, the application checks to see if a directory has previously been written to disk. This file, if present is an encrypted container that holds all of the directory entries that existed when the application was last closed. The filename is currently hard coded to "Assignment2.ser". If this file is found an attempt is made to open it, decrypt it, and then read it in. Note: the encryption/decryption system currently uses a key derived from a hard coded string. To truly be secure, this password value should be read in at runtime from a trusted source and then securely overwritten in memory after use.

Multiple validation steps take place as the file is read into memory. For one, the decryption process implicitly verifies that the data has not been corrupted. Any changes to the contents of the encrypted file will cause the decryption to fail. Next, the standard Java serialization system checks to see if the objects being read are the objects that are expected. Any discrepancies will cause an Exception to be thrown. In addition, as each DirectoryEntry and its associated Person and Telephone objects are read in, localized validation routines are executed by the serialization engine. This was accomplished by implementing the Serializable routine readObject in Person and Telephone. Both classes register a validation routine inside of readObject which is used to verify that the class instance variables still meet the format requirements. Failure of the verification will result in an InvalidPersonException or InvalidTelephoneException as appropriate for the object being re-instantiated.

Instructions for Building and Running Application

The application was written using the Java J2SE SDK and requires at least version 1.5. It consists of seven class files (shown in Table 4) that can be successfully compiled by placing all of the files in a single directory and executing the following command.

```
javac *.java
```

Note: This will throw a warning recommend the use of Xlint. The specific warnings refer to recommendations to provide Serialization IDs for each of the serializable classes and a cast warning when reading in the TreeSet object from the object input stream. Both of these warning types have been analyzed to verify that they represent no potential negative impacts to the resulting application.

It should also be mentioned that the application requires the use of the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files. If these are not installed the application will immediately throw a Key Size exception on start up as

the application is using Triple DES encryption to protect the directory file. This exceeds the allowable cryptographic strength for the default cryptographic policy files.

The application can be run using the following command.

```
java Assignment2
```

Note: if the directory file (Assignment2.ser) is corrupt, it will need to be deleted to enable the application to start up.

Note: the logging system currently writes SEVERE level log entries to the console. This can be changed by modifying the logging policy settings (see documentation for java.util.logging.logger).

Appendix A. Source Code

Included in attached zip file “Ritchey, Ronald MA2.zip”

Appendix B

Test Harness

A test application was developed to verify the correct operation of the Person and Telephone application. It is included in the zip file and is named test.java. It can be easily modified to test the acceptance and rejection of a large number of test entries.