Compression Engine

Build Instructions:

Build tool used in project: gradle

In order to install gradle on a Mac use:

```
brew install gradle
```

On a linux enviornment:

```
$ sudo add-apt-repository ppa:cwchien/gradle

$ sudo apt-get update

$ sudo apt-get install gradle
```

After installing gradle, run:

```
gradle build
```

This should build the project and run all the tests.

Finally use:

```
./gradlew run
```

This will run the project.

An alternative to this, is importing the project as a gradle project on Eclipse or IntelliJ, and running Main.java

About this project:

In my solution to this projet given to me by Hadean Supercomputing. I first thought about doing Huffman or LZ-78 encoding, but soon realised that a uniform distribution of data is likely to cause worst case in both these methods, since they rely on the un-eveness of symbols or sequences in the input. I also thought about what James said in the email about the coding task itself not taking too long, and I came to the conclusion to use a much simpler method.

In my solution, I make a first pass of the input file in order to figure out how many unique characters there are in the file. I am using constant length encoding in order to represent each symbol, so a log of the number of symbols gave me the number of bits needed to encode each symbol. From here on, I assigned each unique symbol an encoding of it's own starting at 1 (So 0001 if we needed 4 bits to represent each symbol) and going upto whatever number was needed. I then added the number of unique symbols and the total number of symbols in the input file as meta-data to my compressed file, followed by the dictionary mapping

symbols to code and lastly the encoded input file data. This brings me to the first limitation of my program, it takes a file with name say filename.txt and calls it filename-compressed.txt, and then takes the file called filename-compressed.txt in order to produces filename-reconstructed.txt. I have no tests to see if the file entered to decode is actually a file that I encoded, I could have done using a specific extension (.sip -> shreyash zip) but I chose to not do anything like that for the sake of simplicity. It might be possible to mess-up the filenames because I have a very specific procedure for turning filaname-compressed.txt names to filename-reconstructed.txt names and it might be possible to mess with the naming scheme, but since that wasn't the point of this project and I went over 4.5 hours while writing my solution, I chose to leave it as it is right now.

In terms of compression, the algorithm does really well when the number of symbols is small, and gets progressively weaker with files containing a larger number of symbols. The project is designed to work well in scenarios when the number of occurences of different symbols is of the same order, and will perform much worse as compared to say a Huffman in cases where there is a skew in the frequencies of letters.

In terms of design, the project is meant to run as an interactive appliction, through the main. I have BitInputStream and BitOutputStream classes to help me read/write a BitSequence (a variable length sequence of bits that I created) from the files. These classes were very useful in abstracting the specifics of how reading and writing in terms of bits was done. I also have ICompressor and IDecompressor interfaces which will allow the creation of many other compresser-decompressor pairs whcih have the same method to encode and decode information and thus, could be used intercahngably.