COSC 237 Dierbach
Fall, 2023

PROGRAM 3

Translator Program
(Subclasses, Abstract Classes, Polymorphism, UML)
100 pts.
Due Date: December 5th by 11:59pm

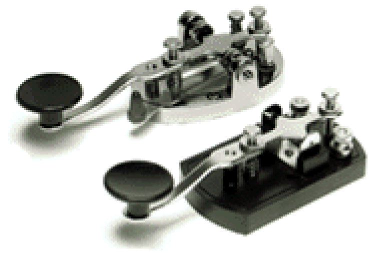
I. Assignment Overview

In this assignment, you are given a running program that is able to translate English messages into More code, and Morse code into English. You are to modify the program to translate between three different languages – English, Morse code and ASCII binary code.

II. Background

Morse Code

Morse code is a binary encoding method (i.e., a two-symbol encoding) first used in the early 1800's and was the basis of the transmission of messages over the important high-speed communications device of the day, the telegraph. The "symbols" were denoted by dots ("•") and dashes ("▬") but were actually implemented as short and long audible beeps over a telegraph wire. (Just as computer bits are represented as 1s and 0s, but are actually implemented as voltage levels, or magnetic particles, or optical pits.) Telegraph operators were trained to understand the Morse code encoded messages. They could both transmit English messages in Morse code and decode Morse code encoded messages back into English. Thus, a Morse code operator was a translator.



The first telegraph line was constructed between Baltimore and Washington and the first message, sent on May 24, 1844, was 'What hath God wrought!' In 1861 the two coasts of the United States were linked by telegraph, and high-speed electro-mechanical transmission of data across the country had begun, over one-hundred years before the development of the Internet, similarly relying on the simplicity of binary encoding (dots and dashes instead of 1s and 0s).

There is one notable difference between the binary coding of Morse and the binary coding used in current computer communication. Current digital (binary) communication is sent in fixed length groups of bits, each group representing one character. Such a grouping of bits is usually eight bits long, referred to as a "byte." Morse's code is a variable length code. Thus, the number of such symbols for any given character is not the same. For example, the Morse code for the letter "A"' is dot-dash (•▬), while the Morse code for the letter "Q" is dash-dash-dot-dash (▬ ▬ • ▬). This makes the translation of such an encoding a little more difficult than a fixed-length encoding.

ASCII Code

ASCII code (American Standard Code for Information Interchange, more properly called US-ASCII code) is a binary encoding scheme developed in the 1960s for the representation of English (and some special) characters. It is a seven-bit code (stored in 8-bit bytes) and thus only able to represent 128 different characters (2^7). It is now part of the Unicode standard able to encode over one-million characters (a varying length encoding using from 8-bits to 32-bits) with the purpose of encoding of essentially all languages past and present, and commonly used symbols (in mathematics, music, etc.).

III. Assignment Details

The Use of Text Files

All messages are stored in text files. Thus, a simple text editor such as Notepad or any program editor can be used to create a message file. Note that a word processor like Microsoft Word cannot be used for this purpose, since it will add extra unnecessary formatting characters that will confuse the translator.

The translator program reads message files and can create a translated version to an output file. For example, if the translator is told to translate a file containing a message (in English), it will open that file, read it, and produce a file containing the Morse code translation of that message (in a file named <original file name>_MORSE). The translator can then be told to translate the message file in <original file name>_MORSE back into English (in a file named <original file name>_MORSE_ENGLISH.

The program requires that all English messages be restricted to upper-case letters only and space characters. Punctuation and digit characters are not allowed!

The Translation of Characters

The program translates messages letter-by-letter. The "ordinal value" (i.e., which position in the alphabet a letter is) is used to determine which (English or encoded) letter to translate to. For example, if the letter 'H' is read from an English input message to be output to a Morse-coded file, the ordinal value of 'H' needs to be determined. This is provided by the getOrdinal() method of the InputEnglishMessage class. The same would need to be done when reading from a Morse-encoded or binary-encoded message. Thus, the getOrdinal() method is defined as an abstract method in the (abstract) InputMessage class, to be implemented by each of its subclasses. Similarly, when writing to an output message, the ordinal value of the current letter read from the input message needs to be converted into the appropriate letter representation of the output message. This is done by method getLetterWithOrdinal. This is declared as a protected (i.e., private method only accessible to the subclasses) in the OutputMessage class.

File Formats

English message files can contain any number of lines of text (restricted to upper-case letters and the space character). For Morse-encoded files, there will be one and exactly one encoded English letter per line. A blank line will indicate the end of a word, and two blank lines the end of a sentence, as given in example messages on the last page (along with ASCII and Morse code tables).

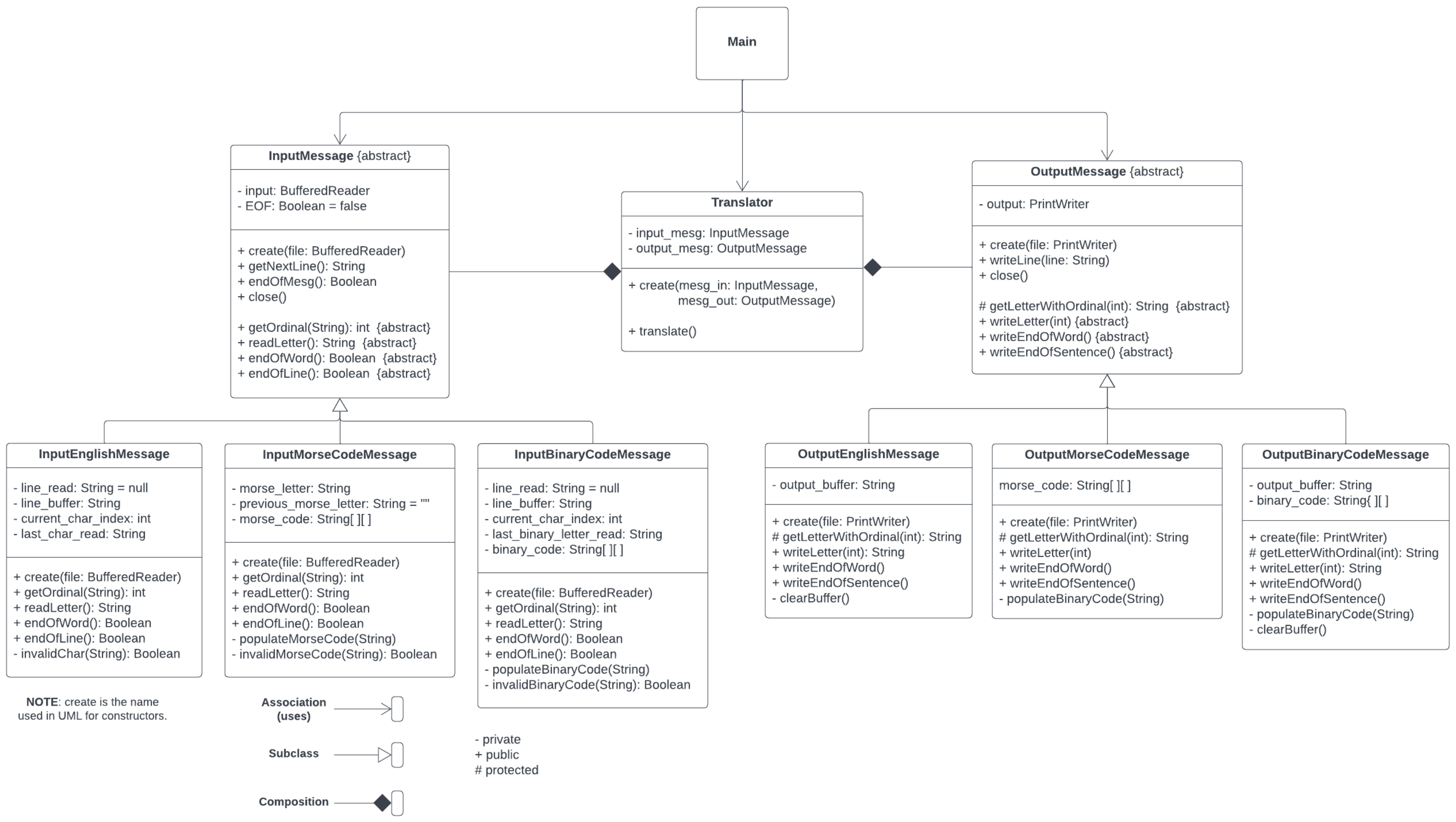
Program Design

Following is a class diagram for the program, given in a UML class diagram.

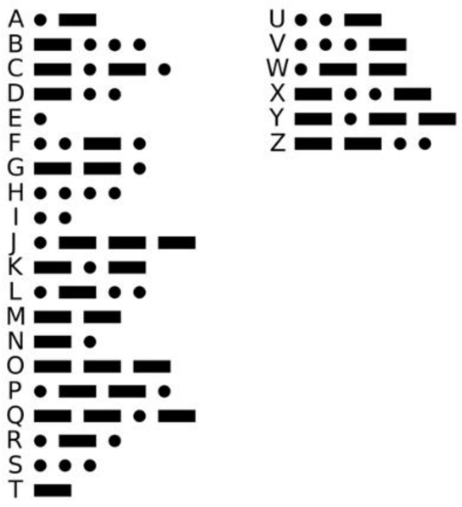
What to Do

You are to modify the design and implement the program to be able to convert between any of English, Morse-coded and binary-coded messages to any other message type. The main program for this is given. Therefore, you are to develop and test the InputBinaryMessage and OutputBinaryMessage classes.

IT IS STRONGLY RECOMMENDED THAT YOU USE A PROGRAM DEBUGGER FOR THIS ASSIGNMENT!
You may use any IDE that has an adequate debugger. Visual Studio Code (VSCode). It is available for download at: https://code.visualstudio.com/


Main
InputMessage {abstract}
- input: BufferedReader
- EOF: Boolean = false
+ create(file: BufferedReader)
+ getNextLine(): String
+ endOfMesg(): Boolean
+ close()
+ getOrdinal(String): int {abstract}
+ readLetter(): String {abstract}
+ endOfWord(): Boolean {abstract}
+ endOfLine(): Boolean {abstract}
Translator
- input_mesg: InputMessage
- output_mesg: OutputMessage
+ create(mesg_in: InputMessage, mesg_out: OutputMessage)
+ translate()
OutputMessage {abstract}
- output: PrintWriter
+ create(file: PrintWriter)
+ writeLine(line: String)
+ close()
# getLetterWithOrdinal(int): String {abstract}
+ writeLetter(int) {abstract}
+ writeEndOfWord() {abstract}
+ writeEndOfSentence() {abstract}
InputEnglishMessage
- line_read: String = null
- line_buffer: String
- current_char_index: int
- last_char_read: String
+ create(file: BufferedReader)
+ getOrdinal(String): int
+ readLetter(): String
+ endOfWord(): Boolean
+ endOfLine(): Boolean
- invalidChar(String): Boolean
InputMorseCodeMessage
- morse_letter: String
- previous_morse_letter: String = ""
- morse_code: String[ ][ ]
+ create(file: BufferedReader)
+ getOrdinal(String): int
+ readLetter(): String
+ endOfWord(): Boolean
+ endOfLine(): Boolean
- populateMorseCode(String)
- invalidMorseCode(String): Boolean
InputBinaryCodeMessage
- line_read: String = null
- line_buffer: String
- current_char_index: int
- last_binary_letter_read: String
- binary_code: String[ ][ ]
+ create(file: BufferedReader)
+ getOrdinal(String): int
+ readLetter(): String
+ endOfWord(): Boolean
+ endOfLine(): Boolean
- populateBinaryCode(String)
- invalidBinaryCode(String): Boolean
OutputEnglishMessage
- output_buffer: String
+ create(file: PrintWriter)
# getLetterWithOrdinal(int): String
+ writeLetter(int): String
+ writeEndOfWord()
+ writeEndOfSentence()
- clearBuffer()
OutputMorseCodeMessage
morse_code: String[ ][ ]
+ create(file: PrintWriter)
# getLetterWithOrdinal(int): String
+ writeLetter(int)
+ writeEndOfWord()
+ writeEndOfSentence()
- populateBinaryCode(String)
OutputBinaryCodeMessage
- output_buffer: String
- binary_code: String{][ ]
+ create(file: PrintWriter)
# getLetterWithOrdinal(int): String
+ writeLetter(int): String
+ writeEndOfWord()
+ writeEndOfSentence()
- populateBinaryCode(String)
- clearBuffer()
NOTE: create is the name used in UML for constructors.
Association (uses)
Subclass
Composition
- private
+ public
# protected

Morse Code



A	● ▬
B	▬ ● ● ●
C	▬ ● ▬ ●
D	▬ ● ●
E	●
F	● ● ▬ ●
G	▬ ▬ ●
H	● ● ● ●
I	● ●
J	● ▬ ▬ ▬
K	▬ ● ▬
L	● ▬ ● ●
M	▬ ▬
N	▬ ●
O	▬ ▬ ▬
P	● ▬ ▬ ●
Q	▬ ▬ ● ▬
R	● ▬ ●
S	● ● ●
T	▬
U	● ● ▬
V	● ● ● ▬
W	● ▬ ▬
X	▬ ● ● ▬
Y	▬ ● ▬ ▬
Z	▬ ▬ ● ●

Charachter	ASCII	Binary
Space	32	00010000
0	48	00110000
1	49	00110001
2	50	00110010
3	51	00110011
4	52	00110100
5	53	00110110
6	54	00110110
7	55	00110111
8	56	00111000
9	57	00111001
A	65	01000001
B	66	01000010
C	67	01000011
D	68	01000100
E	69	01000101
F	70	01000110
G	71	01000111
H	72	01001000
I	73	01001001
J	74	01001010
K	75	01001011
L	76	01001100
M	77	01001101
N	78	01001110
O	79	01001111
P	80	01010000
Q	81	01010001
R	82	01010010
S	83	01010011
T	84	01010100
U	85	01010101
V	86	01010110
W	87	01010111
X	88	01011000
Y	89	01011001
Z	90	01011010

Example Message File

English Message File

```
HI  THERE
HOW ARE YOU
```

Morse-Coded File

```
....
..
<blank line>
-
....
.
.-.
.
<blank line>
<blank line>
....
---
.--
<blank line>
.-
.-.
.
<blank line>
-.--
---
...-
```