# Project 1

Due October 13, 2020 at 9:00 PM

This project specification is subject to change at any time for clarification. For this project you will be writing several Java classes to implement a programming language scanner and a CSV parser. The classes developed in this project will be used in subsequent projects. A Token class has been provided for you and an interface the Scanner will rely on for input has also been provided. The Scanner class must take in a PeekableCharacterStream and a List of keywords and must produce a stream of Tokens that are queried from either peekNextToken or getNextToken methods. The rules for the tokens defined below. You will also develop the CSVParser class that will parse CSV files into Maps, one for each row.

Your first task is to develop a class that implements the PeekableCharacterStream interface for a FileInputStream.

```
public interface PeekableCharacterStream{
    // Returns true if more characters are available, false otherwise
    public boolean moreAvailable();

    // Returns the next character that would be returned without consuming
    // the character. If no more characters are available -1 is returned.
    public int peekNextChar();

    // Returns the character ahead in the stream without consuming the
    // the character. peekAheadChar(0) returns the same character as
    // peekNextChar(). If no more characters are available at that position
    // -1 is returned.
    public int peekAheadChar(int ahead);

    // Returns the next character and consumes it. If no more characters are
    // available -1 is returned.
    public int getNextChar();

    // Closes the stream.
    public void close();
}
```

Your second task is to develop the Scanner class that will rely on the PeekableCharacterStream interface and the Token class. The Scanner must have the following minimal interface.

```
public class Scanner{
    // Constructor that takes in a stream and a list of keywords.
    public Scanner(PeekableCharacterStream stream, List<String> keywordlist);

    // Returns the next token without consuming it. If no more tokens are
    // available a None token is returned.
    public Token peekNextToken();

    // Returns the next token and consumes it. If no more tokens are
    // available a None token is returned.
```

```
    public Token getNextToken();
}
```

**Token Rules:**

```
Identifier := ( _ | Alpha ) { ( _ | Digit | Alpha )  }
Operator := ( | , | ) | { | } | = | == | < | > | <= | >= | != | + | - | * |
            / | ;
IntConstant := [ - ] Digit { Digit }
FloatConstant := [ - ] Digit { Digit } [ . Digit { Digit } ]
StringConstant := " { ( CharacterLiteral | EscapedCharacter ) } "
Digit := 0 – 9
Alpha := A – Z | a – z
WhiteSpace := Space | Tab | CarriageReturn | NewLine
CharacterLiteral := Space - ! | # - ~
EscapedCharacter := \b | \n | \r | \t | \\ | \' | \"
```

**Additional Tokenizing Rules:**

- Keywords are identifiers that are in the List provided to the Scanner.
- Whitespace must be skipped during the tokenizing.
- A negative sign immediately preceding an integer or float constant must be tokenized as an operator if the previous token was a constant or identifier. For example, "`A -5`" must be tokenized as Identifier "`A`", Operator "`-`", and IntConstant "`5`", not as Identifier "`A`", and IntConstant "`-5`".
- If an underscore or Alpha character immediately follows a constant, the token is considered Invalid. All Alpha, Digit, and underscore characters will be part of the Invalid token.
- If an invalid character is in a string constant, the characters are consumed until the next non-escaped " is reached or the end of stream is reached. The token type is considered Invalid.
- Any invalid character beginning a token will be considered an Invalid token by itself. For example, "`@#4`" must be tokenized as Invalid "`@`", Invalid "`#`", and IntConstant "`4`".

**Implementation Requirements:**

- You may use java.io.FileInputStream.
- You may use java.util.Set, java.util.HashSet, java.util.Arrays and similar containers.
- You may **not** use java.util.regex or similar packages.
- You may **not** use java.util.StringTokenizer or similar library classes.

Your final task is to develop the CSVParser class that will rely on the PeekableCharacterStream interface. The CSVParser must have the following minimal interface.

```
public class CSVParser{
    // Constructor that takes in a stream.
    public CSVParser(PeekableCharacterStream stream);

    // Returns the next row without consuming it. If no more rows are
    // available null is returned.
    public Map<String,String> peekNextRow();

    // Returns the next row and consumes it. If no more rows are
    // available null is returned.
    public Map<String,String> getNextRow();
}
```

**CSV Format Rules**

- CSV files must have a header row, and no column in the header may be repeated or empty.
- Each row is terminated by a newline character.
- Each column is terminated by a comma character.
- Any whitespace (space, tab, carriage return, or newline) character that is part of a column must be a double quoted " column. The escape sequence for a double quote in a double quoted column is two double quotes in a row.
- Any empty columns or missing columns will return a value of null for the corresponding value in the returned Map.
- Valid CSV files are not allowed to have more columns in a data row than the header row but may have fewer.

Your Scanner and CSVParser classes must have a main function that takes in a filename as an argument and outputs the contents similar to the examples in `/home/cjnitta/ecs140a/proj1` on the CSIF. You can run the provided solutions by running the shell scripts `Scanner.sh` or `CSVParser.sh` and providing a filename to open. Your code will be tested on the CSIF and is expected to compile and run on the CSIF.

You **must** submit the source file(s), a Makefile, and README.txt file, in a tgz archive. Do a `make clean` prior to zipping up your files so the size will be smaller. You will want to be in the parent directory of the project directory when creating the tgz archive. You can tar gzip a directory with the command:

```
tar -zcvf archive-name.tgz directory-name
```

You should avoid using existing source code as a primer that is currently available on the Internet. You **must** specify in your readme file any sources of code that you have viewed to help you complete this project. You must also provide the URL any code sources in comments of your source code. All class projects will be submitted to MOSS to determine if students have excessively collaborated. Excessive collaboration, or failure to list external code sources will result in the matter being referred to Student Judicial Affairs.