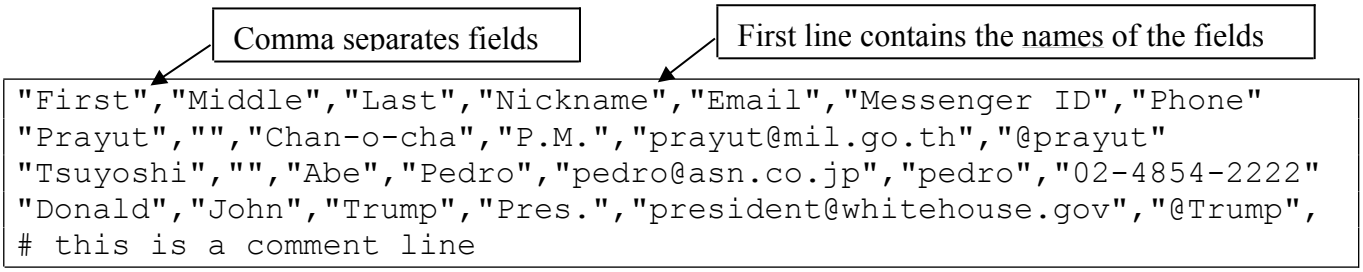
Assignment	1. Write a CSVReader that can read a file or input stream. It reads comma separated values (CSV) data line by line and returns an array of String values for each line using an *Iterator*. 2. Use the package ku.util for the CSVReader class. 3. Write the code yourself. Don't use a CSV library. Don't use your friend's code.
What to Submit	Commit your project to Github Classroom using the repository created for this assignment.

Description

Comma Separated Values (CSV) is a standard format for data exchange. Yahoo and Google Mail address books, Excel worksheets, and almost any database can import or export data in CSV format.

Each line of text in a CSV file contains the data for one record or "row". Each line contains multiple values (fields) separated by a *delimiter character*, usually a comma. The fields may or may not be enclosed in quote marks.

Yahoo and Google Mail use CSV to import and export contacts. When you export your Yahoo Address Book it creates a file like this:




```
"First","Middle","Last","Nickname","Email","Messenger ID","Phone"
"Prayut","","Chan-o-cha","P.M.","prayut@mil.go.th","@prayut"
"Tsuyoshi","","Abe","Pedro","pedro@asn.co.jp","pedro","02-4854-2222"
"Donald","John","Trump","Pres.","president@whitehouse.gov","@Trump",
# this is a comment line
```

Yahoo and Google Mail use quotes around all fields, and the first line contains the names of the fields. Empty fields are empty strings.

Microsoft Excel and LibreOffice can read a worksheet as CSV or save it as CSV. When you open a CSV file, a dialog will ask you what the separator char is, whether there are quotes, and other details. If you save a worksheet as CSV, the format looks similar to this:

```
# First,Last,Email
Kanyapak,Prayoonpat,kanyapak.pra@ku.th
Fatalai,,fatalaijon@gmail.com          (field 2 is empty)
,Sittipongpanich,nutta.sit@ku.th       (field 1 is empty)
Teeranut,Sawanyawat,teeranut.s@ku.th
```

Each line contains one row from the spreadsheet. By default, it does not put quotes around fields, but you can specify quotes as an option. Fields can be empty, as in the example above.

Requirements for CSVReader

1. CSVReader reads data from an InputStream or a file, which are given as parameter to the constructor. It splits each line into fields using a delimiter character. The default delimiter is a comma.

2. CSVReader implements *Iterator*. The next( ) method returns one line of data as an **array of Strings**. Each line of data may contain a different number of fields, so the array size may be different for each line. Some fields may be empty! In that case, next() will return an array with some empty Strings (not null).

```
first,second,third,fourth    (4 fields - next returns String[4])
this,line,has,,,6 fields     (6 fields, 2 are empty)
first,second,,               (4 fields, last 2 are empty)
```

3. The reader **skips** blank lines (empty line or line containing only space and/or tabs) and lines beginning with # (comment lines). next() does not return these lines and hasNext() skips past them.

4. When parsing Strings from a line, you need to do (at least) 2 things to the input data:

a) if a field is surrounded by quotation marks ("like this") then remove them, but don't remove quotation marks *inside* a field (like "this" example).

b) remove space characters at the start or end of a field, but not in the middle of a field. Don't remove space that is *inside quotation marks* (" like this "). Be careful of TABs because TAB may be the field delimiter character.

Example: `" inner ", "space"` returns array[0] = `" inner "`, array[1] = `"space"`

5. If application calls next() when there is no more input data, the next() method should throw a NoSuchElementException. This is part of the specification for *Iterator* in the Java API.

6. CSVReader can read the **input source *only once*** and *must not try to store the entire input in memory!* Only buffer and process one line at a time, and only do it when necessary. The input may be contain millions of lines.

7. CSVReader should **not print anything** on System.out. Not even error messages!

8. The **default delimiter** between fields is comma (',') but the user can change this at any time by calling setDelimiter(char).

9. For simplicity, we require that the delimiter (e.g. comma) can not appear inside a field even if the field is surrounded by quotation marks.

Example: This input is a file containing this data:

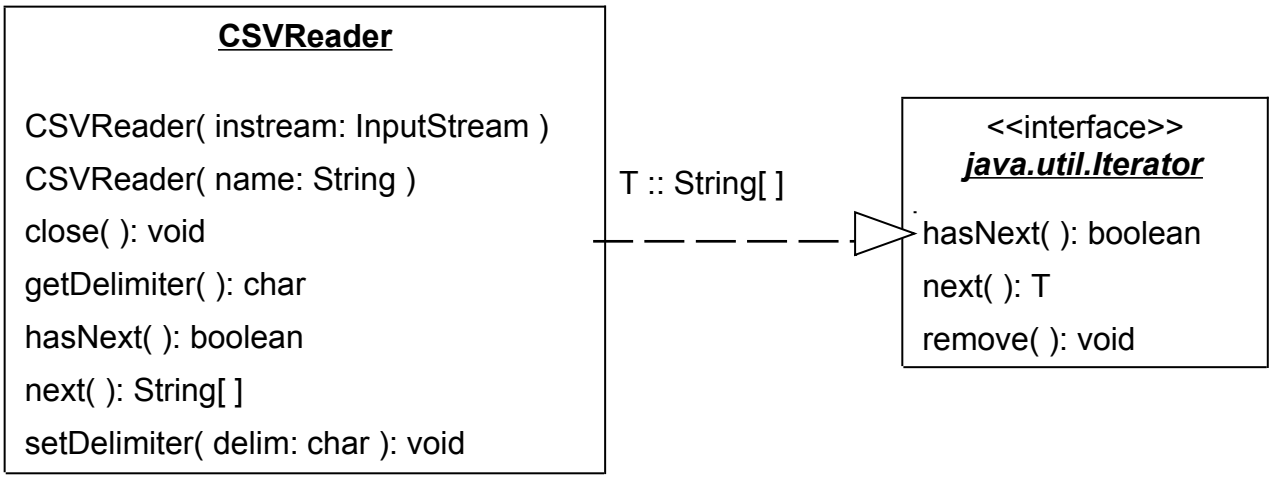
```
FIRST,LAST,AGE
"Harry", "Potter"     , 16

Magic, Owl,
Albus, Dumbledore, unknown, Professor
```

In BlueJ Codepad we would see

```
> CSVReader csv = new CSVReader("sample.csv");
> csv.hasNext( )
true
> csv.next( )
["FIRST", "LAST", "AGE"]  // array of Strings
```

```
> csv.next( )
["Harry", "Potter", "16"]
  // the line has 4 fields. Note that it removed extra space.
  // the quote marks in the CSV data are NOT part of the Strings!
> csv.hasNext()
true
> csv.next( )
["Magic", "Owl", ""]
// Because of the trailing "," there are 3 fields, last field is empty
> Albus, Dumbledore, unknown, Professor
["Albus", "Dumbledore", "unknown", "Professor"]
// the iterator returns the next() value even if user does not call
// hasNext() first.
> csv.hasNext( )
false                 // last line is empty, so no more data
> csv.next( )
  java.util.NoSuchElementException at CSVReader.next(...)
  // throws exception because there is no more data.
```



Method Descriptions

CSVReader(InputStream)	Constructor for a CSVReader that reads from an InputStream. The parameter value must be a valid InputStream (not null).
CSVReader(String filename)	Constructor with a filename to read data from. Exceptions: May throw FileNotFoundException if file cannot be opened or doesn't exist. See the constructor for FileInputStream(String) for explanation of when and why these exceptions are thrown.
`void close( )`	Close the input source (InputStream or file).
`setDelimiter(char c)`	Set the delimiter character for separating the input line into fields. The default delimiter is a comma character.
`char getDelimiter( )`	Return the current delimiter character.
`boolean hasNext( )`	Return true if there is more data in the input, otherwise return false.
`String[] next( )`	Return an array of Strings containing the next line of CSV data from the input, separated into fields. 1. In the input, fields are separated by a delimiter character (default is comma) and may be surrounded by quotation marks, as in the example above. 2. If any fields are empty, the array returned by **`next`** should have a zero length String for that element of the array (*not* a null). 3. If any fields begin/end with quotation marks such as `"Red Dog"` then remove the quotation marks from beginning and end of the field data. Also remove space from the beginning and end of the field. 4. The length of the array returned by **`next`** should exactly match the number of fields (including empty fields) in the data itself. Don't make the array larger than the data. Throws NoSuchElementException if no more data available.
`void remove( )`	Does nothing. Leave this method empty.

Exceptions Thrown	The next( ) methods may throw NoSuchElementException if there is no data to return. The constructors may throw FileNotFoundException (a checked exception) if file cannot be opened. The methods never throw checked exceptions, but may throw a RuntimeException if there are any errors. See below for how to "wrap" an I/O exception in a RuntimeException.

Programming Hints

1. To open a file as an InputStream, you can use:

```
InputStream input = new FileInputStream( filename );
```

this may throw a FileNotFoundException. The constructor for CSVReader can just propagate this exception up to the caller by declaring it throws the exception:

```
/**
 * Initialize a new CSV Reader that reads from a file.
 * @param ...
 * @throws FileNotFoundException if file doesn't exist or cannot be opened.
 */
public CSVReader(String filename) throws FileNotFoundException {
    input = new FileInputStream( filename );
}
```

2. Scanner is slow. A more efficient way it read lines is a BufferedReader,. It can read lines of input as String (like Scanner nextLine()). To create a BufferedReader, use:

```
private BufferedReader reader;
public CSVReader(InputStream input) {
      reader = new BufferedReader(input);
}
```

3. To split the String into fields (tokens), for each line read from the input use one of these methods:

(a) `String.split( regex )`

Splits a String into fields using a pattern and returns an array containing the fields. Patterns use regular expression notation, but you can also just use a constant string, for example:

```
String[] words = data.split(",");
```

For the curious, a regular expression that matches 0 or more space chars then a comma, then more space chars is: `" *, *"`. Try it: `"Hello ,, hacker.".split(" *, *").`

Regular expressions also us **`\\s`** to mean "space or tab". Be careful using this because the field delimiter character may be TAB.

(b) **`StringTokenizer`** also splits a String at a delimiter.

4. Don't Read Any Input in the Constructor. Reading data should be performed by hasNext. That's the only way to check if there is really another line of data or not (has to skip blank lines and comments line). hasNext should read and save one line of data.

5. The methods of CSVReader should not throw checked exceptions. We don't want to require users of CSVReader to write `try` - `catch` around all their code, so the CSVReader class will catch exceptions

and "rethrow" them as a RuntimeException, which is *unchecked* (the user does not have to use try - catch). Many application frameworks do this, too.

Here is how to catch an IOException and "wrap" it in a RuntimeException.

```
// Attribute for a BufferedReader that reads the input stream
private BufferedReader reader;

/** Read one line from the input and return as a String. */
public String readOneLine() {
     try {
          String line = reader.readLine();
          return line;
     } catch ( IOException ex ) {
          // this code "wraps" the IOException in a RuntimeException
          throw new RuntimeException( ex.getMessage(), ex );
     }
}
```

6. As usual, write good Javadoc. You should document all exceptions thrown by methods and constructors. The Javadoc syntax for this is:

```
/**
 * Initial a new CSVReader for reading from a file or URL.
 * @param filename blah, blah, blah
 * @throws FileNotFoundException if the file does not exist,
 *       is not a regular file, or cannot be opened
 * @throws IOException (write it yourself. See FileInputStream for example)
 */
```

Test your code early and often.

- Write the constructors and then see if you can simply read one line and print it!
- Then write hasNext and next to read the file line-by-line. hasNext does most of the work.
- When that works, try splitting the line into fields as required.

Sample Data and Tests

Will be posted if anyone accepts this assignment.