Assignment	1. Write a CSVReader that can read a file or URL that contains data in CSV format. CSVReader provides an *Iterator* for reading CSV data one line at a time. 2. The reader should provide the constructor and methods described below. 3. Use the package ku.util for this class.
What to Submit	Commit your project to Github Classroom, using the repository that Github creates for you when you accept this assignment (invitation sent by email). The repository name is csvreader-*your_github_id*. The repository is private.

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" from the application.

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

Comma separates fields

First line contains the names of the fields

```
"First","Middle","Last","Nickname","Email","Messenger ID","Phone"
"Taksin","","Shinawat","Pres.","taksin@ais.co.th","taksin","1175"
"Tsuyoshi","","Abe","Pedro","pedro@asn.co.jp","pedro","+02-4854-2222"
"Donald","John","Trump","Pres.","president@whitehouse.gov","@Trump",
```

Fields are separated by a comma and each field value is placed in quote marks, although quotes are *not required* for CSV (unless the field value contains the separator character (comma)). Fields with no data are output as empty quotes. The first line contains the *field names,* and the remaining lines are your contacts (of course). The field names (first line) makes the data easy to import. For example, Gmail will figure out what data is in the file using field names on the first line.

Microsoft Excel or 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:

```
StudentID,Firstname,Lastname,Github Username
5910545639,Kanchanok,Kannee,mailtoy
5910545647,Kwankaew,Uttama,ploykwan
5910545655,Jiranan,Patrathamakul,jiranan-pat
5910545663,Charin,Tantrakul,kaizofaria
5910545671,Chawakorn,Suphepre,winChawakorn
5910545680,Triwith,Mutitakul,famefryer
```

Each line contains one row from the spreadsheet. By default, it doesn't put quotes around fields, but you can specify quotes as an option.

Requirements for CSVReader

1. CSVReader reads data from an InputStream, a file, or a URL. It splits each line into fields using a delimiter character (the default is comma) and returns the values as an array of Strings. It has 2 constructors as described below.

2. CSVReader implements *Iterator*. The next( ) method returns one line of data as an array of String.

3. Each line of input data may contain a different number of fields, and some fields may be empty! CSVReader next( ) returns an array *exactly* matching the number of fields on the current source line, including empty fields. For example:

```
first,second,third,fourth    (4 fields)
this,line,has,,,6 fields     (6 fields, but 2 fields are empty)
```

4. If a field is blank then return an empty String for that field, **not a null**. For example, this line:
`Santa,Claus,,"I love Christmas"`
would be parsed as `["Santa", "Claus", "", "I love Christmas"]`

5. Remove any whitespace (space or tab characters) at the start or end of field data. If a field begins with a double quote char (") then also remove the quote character from the beginning and end of the field. Quotes around field values are optional (as in the example above). Don't remove whitespace that is *inside* quote characters. For example, `"inner  ",  "space"` the first value should be `"inner  "`.

5. Skip blank lines and comment lines. Skip any line containing only whitespace. Also skip lines where the first non-space character is #. Lines beginning with # are comment lines (by convention).

6. If application calls next() when there is no more input data, the next() method should throw a NoSuchElementException.

7. CSVReader can read the **input source *only once*** and *must not attempt 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.

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

9. **Do not use Scanner**. You can use InputStreamReader, BufferedReader, StringTokenizer, or other classes from the Java API.

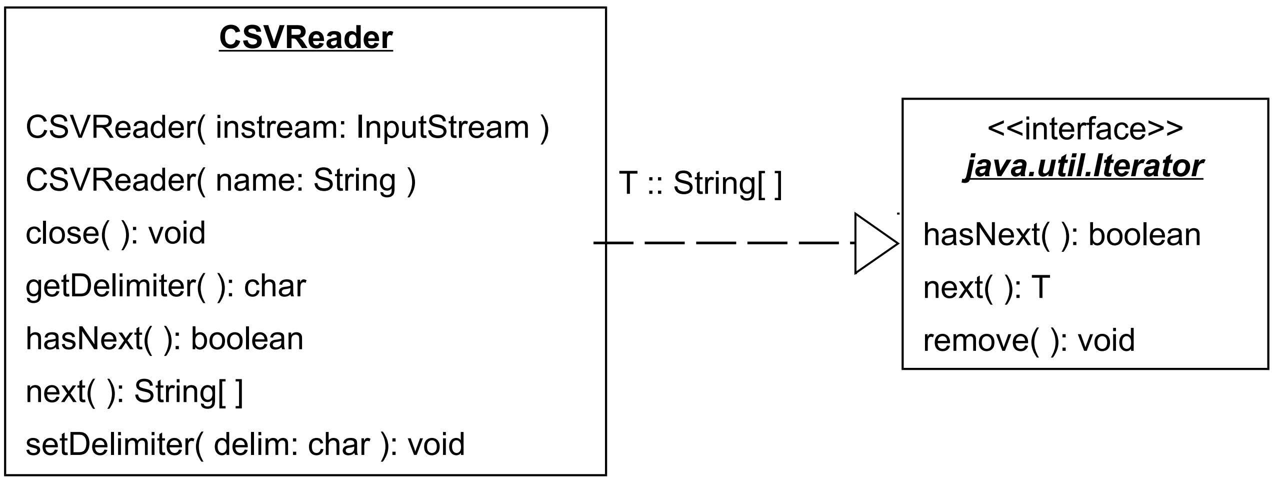
10. The **default delimiter** between fields is comma (',') but the user can change this at any time by calling setDelimiter(char).

Example: This input is a file named `"sample.csv"`

```
FIRST,LAST,EMAIL,TELEPHONE
"Harry",  "Potter"     , harry@wizards.com,086-9999999
Magic, Owl,, 12345678 ,
```

In BlueJ Codepad we would see

```
> CSVReader csv = new CSVReader("sample.csv");
> csv.hasNext( )
true
> csv.next( )
["FIRST", "LAST", "EMAIL", "TELEPHONE"]  // array of Strings
> csv.next( )
["Harry", "Potter", "harry@wizards.com", "086-9999999"]
  // 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", "", "12345678", ""]
  // Because of the trailing "," this line has 5 fields but the
  // last field is empty.
> csv.hasNext( )
false
> csv.next( )
  java.util.NoSuchElementException at CSVReader.next(...)
  // throws exception because there is no more data.
  // This is part of the specification of Iterator.next()
```



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 name)	Constructor with a filename or URL to read data from. If the string matches the syntax for a URL (see below), then the URL is open for input. Otherwise it is treated as the name of a local file and opened for input. A URL should begin with a protocol name (at least 2 letters to distinguish it from a Windows disk drive letter), followed by "//" and a path: `protocol://path` For example: "`http://se.cpe.ku.ac.th/students.csv`" or "`ftp://somehost/file.txt`". You can even specify a local file as a URL: "`file:///C:/temp/filename.txt`". Exceptions: May throw FileNotFoundException, SecurityException, or MalformedURLException. See the constructor for FileInputStream(String) and URL(String) for explanation of when and why these exceptions are thrown.
`void close( )`	Close the input source (InputStream, file, or URL).
`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 data.
`void remove( )`	Does nothing. Leave it empty.
Exceptions Thrown	The next( ) methods may throw NoSuchElementException if there is no data to return. The constructors may throw several kinds of checked exceptions. 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. For the constructor with String parameter, you need to distinguish between a local file and the name of a URL. Open URLs using the URL class. For local files, use classLoader.getResourceAsStream. If the file name is given as an absolute path you can also open it using FileInputStream.

To recognize URLs, use a *pattern* that the URL string must match. The String class has a e matches method for testing if a String matches a pattern written as a *regular expression. Regular expressions* have a common syntax that is used in all modern programming languages.

"*protocol*://*location/path*". string.matches( ) uses For example:

// Match 2 word characters (**\w\w**) or more (**+**) followed by colon and then **//**, followed by anything except whitespace (**\\S+**). Must use capital "\S" to mean "not whitespace".

```
String URLPATTERN = "^\\w\\w+://\\S+";
String name = "http://somewhere.com/filename";
name.matches(URLPATTERN)   // true
name = "file:///C:/temp/filename.csv";
name.matches(URLPATTERN)   // true
name = "C:/temp/filename.csv";
name.matches(URLPATTERN)   // false
```

2. The Java URL class makes it *easy* to read from a URL. You create a new URL object and then call openStream() to create an InputStream connected to the URL.

3. **Test your code early and often**.

- Write the constructors and see if you can simply read one line and print it!
- Try all cases: InputStream, local file, and URL.
- Then write hasNext and next to read the file line-by-line and return the entire line using next.
- When that works, try splitting the line into fields as required.

4. **Don't use Scanner** to split an input line into fields. Scanner is slow. Better ways are:

(a) Read lines using a BufferedReader to read lines of input and use a StringTokenizer to split the String

(b) Read lines using a BufferedReader to read lines of input and use String.split( )

(c) Read the input line into an array of characters and split it yourself using a loop. Process each character in the array looking for the delimiter character. Create Strings from characters using a StringBuilder. If you do it this way you can handle fields that contain the delimiter character inside of quotation marks `"like,this"` (the way Excel and CSV toolkits do).

5. **Don't Read Any Input in the Constructor.** Reading data should be performed by hasNext (that's the only way to check if there really is a next line).

6. 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.

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

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

7. 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)
 * @throws MalformedURLException if string matches a URL but is not valid
 */
```

8. Write the code yourself. Don't use a CSV library. Don't use your friend's code.

Sample Data and Tests

Will be posted online.