### Course Specific Classes

The methods included in these tables are the ones used in this course, and the most commonly useful methods of each class. Links are provided to the complete documentation if you are interested.

#### [FileResource](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/FileResource.html" \t "_blank)

This class provides methods for accessing a file on your computer. You can create a FileResource in a variety of ways:

* new FileResource(), opens a dialog box prompting you to select a file on your computer
* new FileResource("path/to/file.ext"), uses the given String to find a file on your computer or within your BlueJ project
* new FileResource(existingFile), uses the given File(typically returned by using a DirectoryResource)

For these examples, assume the variable fr has been created for a specific file.

|  |  |  |
| --- | --- | --- |
| **Method name** | **Description** | **Example** |
| .[lines](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/FileResource.html#lines--)() | returns an Iterable that provides access to the contents of this opened file one line at a time | for (String line : fr.lines()) {  // process each line in turn  } |
| .[words](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/FileResource.html#words--)() | returns an Iterable that provides access to the contents of this opened file one word at a time | for (String word : fr.words()) {  // process each word in turn  } |
| .[asString](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/FileResource.html#asString--)() | returns the entire contents of this opened file as one String | String contents = fr.asString(); |

#### [URLResource](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/URLResource.html" \t "_blank)

This class provides methods for accessing a web page. You can create a URLResource by giving it a complete URL, or web address (note, it muststart with either http:// or https://):

* new URLResource("http://www.something.com/file.ext"), uses the given address to download the referenced file
* new URLResource("https://www.something.com/file.ext"), uses the given address to download the referenced file

For these examples, assume the variable ur has been created for a specific URL.

|  |  |  |
| --- | --- | --- |
| **Method name** | **Description** | **Example** |
| .[lines](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/URLResource.html#lines--)() | returns an Iterable that provides access to the contents of this opened web page one line at a time | for (String line : ur.lines()) {  // process each line in turn  } |
| .[words](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/URLResource.html#words--)() | returns an Iterable that provides access to the contents of this opened web page one word at a time | for (String word : ur.words()) {  // process each word in turn  } |
| .[asString](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/URLResource.html#asString--)() | returns the entire contents of this opened web page as one String | String contents = ur.asString(); |

#### [DirectoryResource](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/DirectoryResource.html" \t "_blank)

This class provides a method for choosing one or more files on your computer. You can only create a DirectoryResource with no parameters:

* new DirectoryResource()

For these examples, assume the variable dr has been created.

|  |  |  |
| --- | --- | --- |
| **Method name** | **Description** | **Example** |
| .[selectedFiles](http://www.dukelearntoprogram.com/course5/doc/javadoc/edu/duke/DirectoryResource.html#selectedFiles--)() | returns an Iterable that provides access to each of the files selected by the user one at a time | for (File f : dr.selectedFiles()) {  // process each file in turn  } |

### Apache Commons CSV Classes

The methods included in these tables are the ones used in this course, and the most commonly useful methods of each class. Links are provided to the complete documentation if you are interested.

#### [CSVRecord](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVRecord.html" \t "_blank)

This class provides methods for accessing individual data values in a line of data within [a CSV formatted file](http://www.dukelearntoprogram.com/course5/doc/#csv). You cannot create a CSVRecorddirectly, instead it will be provided for you when you iterate using a CSVParser. Data values are always returned as a String, so you will need to convert any values you plan to use in your calculations to [the appropriate numeric value](http://www.dukelearntoprogram.com/course5/doc/#converting).

For these examples, assume the variable rec has been created for the second row of data below (the first line represents the header row that names the columns of data).

Name,Food,Color,Number

Fred,Pizza,Purple,13

|  |  |  |
| --- | --- | --- |
| **Method name** | **Description** | **Example** |
| .[get](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVRecord.html#get-java.lang.String-)(columnName) | returns a String, the data in this record corresponding to the column with the given columnName it is an error if the columnName does not exist in the header row (or does not have the same case) | rec.get("Name")is "Fred" rec.get("Food")is "Pizza" |
| .[get](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVRecord.html#get-int-)(columnIndex) | returns a String, the data in this record corresponding to the column at the given columnIndex note, the index of the first data value is 0 | rec.get(0) is "Fred" rec.get(3) is "13" |
| .[size](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVRecord.html#size--)() | returns the number of values in this record | rec.size() is 4 |

#### [CSVParser](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVParser.html" \t "_blank)

This class provides you the ability to iterate over each line of data within a CSV formatted file as a record of the individual data values. Most likely you will not call any methods directly on a CSVParser object, but use it as an Iterable within your loop (you do not even have to call a method to do so, just use the object itself). In any case, here is one possibly useful method.

|  |  |
| --- | --- |
| **Method name** | **Description** |
| .[getCurrentLineNumber](http://www.dukelearntoprogram.com/course5/doc/javadoc/org/apache/commons/csv/CSVParser.html#getCurrentLineNumber--)() | returns the line number of the current record in the iteration |

### Standard Java Classes

The methods included in these tables are the ones used in this course, and the most commonly useful methods of each class. Links are provided to the complete documentation if you are interested.

#### [String](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html" \t "_blank)

This class provides methods for accessing a sequence of characters of any length. Strings are immutable, which means they cannot be changed after they are created. There are two general ways of creating a String:

* String s = new String("Colorful");
* String s = "Colorful";

For these examples, assume the variable s has the value "Colorful"

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .[equals](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#equals-java.lang.Object-)(other) | returns true only if this string has the same characters and in the same order as the other string | s.equals("Colorful") is true s.equals("colorful") is false |
| .[equalsIgnoreCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#equalsIgnoreCase-java.lang.String-)(other) | returns true only if this string has the same characters and in the same order as the other string, regardless of case | s.equalsIgnoreCase("Colorluf")is false s.equalsIgnoreCase("colorful")is true |
| .[length](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#length--)() | returns number of characters in this string | s.length() is 8  "".length() is 0 |
| .[indexOf](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#indexOf-java.lang.String-)(str) | returns the index within this string of the first occurrence of the given string  note, indices returned start at 0, the first character in the string, and go to s.length()-1, the last character note, returns -1 if the given string is not in this string | s.indexOf("l") is 2 s.indexOf("ful") is 5 |
| .[indexOf](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#indexOf-java.lang.String-int-)(str, startIndex) | returns the index within this string of the first occurrence of the given string, starting at startIndex note, indices returned start at 0, the first character in the string, and go to s.length()-1, the last character note, returns -1 if the given string is not in this string | s.indexOf("l", 3) is 7 s.indexOf("o", 1) is 1 |
| .[substring](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-)(startIndex) | returns a string with the characters of this string, starting from startIndexand going to the end of this string note, indices given start at 0, the first character in the string, and go to s.length()-1, the last character | s.substring(1) is "olorful" s.substring(5) is "ful" |
| .[substring](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#substring-int-int-)(startIndex, endIndex) | returns a string with the characters of this string, starting from startIndexand going up to, but not including, the character at endIndex note, indices given start at 0, the first character in the string, and go to s.length()-1, the last character | s.substring(1, 2) is "o" s.substring(1, 4) is "olo" |
| .[toLowerCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toLowerCase--)() | returns a string with the same characters as this string, but with all letters lowercased | s.toLowerCase() is "colorful" |
| .[toUpperCase](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#toUpperCase--)() | returns a string with the same characters as this string, but with all letters uppercased | s.toUpperCase() is "COLORFUL" |
| .[startsWith](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#startsWith-java.lang.String-)(prefix) | returns true only if this string starts with given prefix | s.startswith("Color") is true s.startswith("cool") is false |
| .[endsWith](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#endsWith-java.lang.String-)(suffix) | returns true only if this string ends with given suffix | s.endswith("ful") is true s.endswith("fool") is false |
| .[split](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html#split-java.lang.String-)(String regex) | returns an array of substrings of the original string any string matching the given regular expression counts as a separator note, the answer array does not include the separators | String[] output = s.split("o"); output is now ["C", "l", "rful"] |

#### [StringBuilder](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html" \t "_blank)

Strings are sequences of characters that cannot be changed once they are created. StringBuilder, on the other hand, allows adding and deleting from the sequence dynamically. Just like Strings, each character in the sequence has an index number starting at 0. There are two general ways of creating a new StringBuilder:

* new StringBuilder(), creates a StringBuilder with no characters in it.
* new StringBuilder("Colorful"), creates a StringBuilder initialized with the given characters.

For these examples, assume the variable sb is a StringBuilder with the value "Colorful"

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .[append](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html#append-char-)(char c) | adds the given character to the end of the sequence | sb.append('X'); sb now has the sequence "ColorfulX" |
| .[insert](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html#insert-int-char-)(int offset, char c) | inserts the given character at the specified index in the sequence remember that indexing of letters starts at 0 | sb.insert(1,'X'); sb now has the sequence "CXolorful" |
| .[charAt](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html#charAt-int-)(int index) | returns the character at the specified index in the sequence | sb.charAt(0) is 'C' |
| .[setCharAt](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html#setCharAt-int-char-)(int index, char c) | replaces the character at the specified index in the sequence with the given character | sb.setCharAt(1,'X'); sb now has the sequence "CXlorful" |
| .[toString](https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html#toString--)() | returns a String version of the sequence | sb.toString() is "Colorful" |

### Standard Java Collections

#### [Array](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \t "_blank)

The array is the most basic Java collection. As such, it has limitations in that it has a fixed size and few methods. All elements of an array must have the same type. Like Strings, elements in an array are numbered beginning at 0, and these numbers are referred to as indices.

There are two ways to create an array:

* String[] words = new String[4]; creates an array that can hold Strings with exactly 4 slots. The index of the first slot is 0, and the index of last slot is 3. When arrays are created in this way, the array is filled with default values for each slot. If this were an array of numbers, that would mean that every slot would be initialized with the value 0. Since this is an array of objects, it means that every slot is initialized with a null value.
* String[] words = {"this", "is", "an", "array"};creates an array of 4 Strings as well. The difference is that the slots are initialized with the given words.

For these examples, assume the variable words is an array initialized as in the second line.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .length | returns the number of slots in the array note there are no parentheses after the word length (since it is an instance variable, not a method) | words.length is 4 |
| [] | gives access to the element in the array at the given index remember that indices start at 0 and end at length-1 | words[0] is "this" words[1] = "was"; words is now ["this", "was", "an", "array"] |

#### [ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \t "_blank)

Like arrays, ArrayLists are collections. Unlike arrays, they have more functionality and their size can change. When you create an ArrayListyou declare what kind of object it holds using < > notation.

* ArrayList<String> list = new ArrayList<String>(); creates an empty collection

For these examples, assume the variable list is an ArrayList of Strings and has the words "this", "is", "a", "list" inserted.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .[add](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#add-E-)(object) | adds the given object to the end of the collection | list.add("yes"); list is now ["this", "is", "a", "list", "yes"] |
| .[get](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#get-int-)(int index) | returns the object at the given index remember that indexing starts at 0 | list.get(0) is "this" |
| .[set](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#set-int-E-)(int index, object) | changes the element at the given index to the given object remember that indexing starts at 0 | list.set(1, "was"); list is now ["this", "was", "a", "list"] |
| .[contains](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#contains-java.lang.Object-)(object) | returns whether or not the list has the given object in it | list.contains("this") is true |
| .[indexOf](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#indexOf-java.lang.Object-)(object) | returns the index of the given object or -1 if the object is not in the list | list.indexOf("this") is 0 |
| .[size](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#size--)() | returns the number of elements in the list | list.size() is 4 |
| .[clear](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#clear--)() | removes all the elements of the list | list.clear(); list is now empty with a .size() of 0 |
| .[remove](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-int-)(int index) | removes the element at the given index note that the indices of elements past index get shifted down | list.remove(1); list is now ["this", "a", "list"] |
| .[remove](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-java.lang.Object-)(object) | removes the given element from the list, if it is present | list.remove("is"); list is now ["this", "a", "list"] |
| Iterable | a list is Iterable, allowing access to each item one at a time | for (String s : list) {  // process each item in turn  } |

#### [HashSet](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html" \t "_blank)

A set is a collection that is not intended to have any duplications. Like ArrayLists, the <> notation tells the type of object being stored.

* HashSet<Integer> set = new HashSet<Integer>(); creates an empty collection

For the following examples, assume the set has the values 101, 202, and 303 in it.

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .[add](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html#add-E-)(object) | adds the element to the set, but only if it is not already present | set.add(404); set.add(101); set is now [101, 202, 303, 404] |
| .[contains](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html#contains-java.lang.Object-)(object) | returns whether or not the set contains the given element | set.contains(400) is false |
| .[size](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html#size--)() | returns the number of elements in the set | set.size() is 3 |
| .[clear](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html#clear--)() | removes all elements from the set | set.clear(); set is now empty with a .size() of 0 |
| .[remove](https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html#remove-java.lang.Object-)(object) | removes the given object from the set, provided it exists | set.remove(101); set is now [202, 303] |
| Iterable | a set is Iterable, allowing access to each item one at a time | for (Integer i : set) {  // process each item in turn  } |

#### [HashMap](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html" \t "_blank)

A map is a collection that works as a lookup table. The collection is kept as pairs of keys and their values. Like ArrayLists, the < > notation tells the type of object used as keys and the type used as values. For example, suppose we want to keep a collection of names with ID numbers. We can create a HashMap like this:

* HashMap<String, Integer> map = new HashMap<String, Integer>(); creates an empty collection

For the following examples, assume the map variable has two entries:

* "Robert" maps to 101
* "Drew" maps to 252

|  |  |  |
| --- | --- | --- |
| **Method Name** | **Description** | **Example** |
| .[containsKey](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#containsKey-java.lang.Object-)(object) | returns whether or not the map has an entry with a key that equals the given object | map.containsKey("Robert") is true |
| .[get](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#get-java.lang.Object-)(object) | returns the value associated with the given key, or null if the key doesn't exist | map.get("Robert") is 101 |
| .[put](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#put-K-V-)(key, value) | adds a new entry to the map with the given key and value | map.put("Susan", 400); map now contains 3 entries |
| .[size](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#size--)() | returns the number of entries in the map | map.size() is 2 |
| .[clear](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#clear--)() | removes all the entries from the map | map.clear(); map is now empty with a .size() of 0 |
| .[remove](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#remove-java.lang.Object-)(object) | removes the entry with the given key from the map, if it is present | map.remove("Robert"); map now contains 1 entry |
| .[keySet](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#keySet--)() | returns a set of all the keys in the map | for (String s : map.keySet()) {  // process each key in turn  } |
| .[values](https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html#values--)() | returns a collection of all the values in the map | for (Integer v : map.values()) {  // process each value in turn  } |