

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](#)

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
<code>.lines()</code>	returns an <code>Iterable</code> that provides access to the contents of this opened file one line at a time	<pre>for (String line : fr.lines()) { // process each line in turn }</pre>
<code>.words()</code>	returns an <code>Iterable</code> that provides access to the contents of this opened file one word at a time	<pre>for (String word : fr.words()) { // process each word in turn }</pre>
<code>.asString()</code>	returns the entire contents of this opened file as one <code>String</code>	<pre>String contents = fr.asString();</pre>
<code>.getCSVParser()</code>	returns a <code>CSVParser</code> object for this opened file, assuming it contains comma separated values with a header row	<pre>CSVParser parser = fr.getCSVParser();</pre>
<code>.getCSVParser(false)</code>	returns a <code>CSVParser</code> object for this opened file, assuming it contains comma separated values <i>without</i> a header row	<pre>CSVParser parser = fr.getCSVParser(false);</pre>

[URLResource](#)

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 *must* start 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
<code>.lines()</code>	returns an <code>Iterable</code> that provides access to the contents of this opened web page one line at a time	<pre>for (String line : ur.lines()) { // process each line in turn }</pre>
<code>.words()</code>	returns an <code>Iterable</code> that provides access to the contents of this opened web page one word at a time	<pre>for (String word : ur.words()) { // process each word in turn }</pre>
<code>.asString()</code>	returns the entire contents of this opened web page as one <code>String</code>	<pre>String contents = ur.asString();</pre>
<code>.getCSVParser()</code>	returns a <code>CSVParser</code> object for this opened web page, assuming it contains comma separated values with a header row	<pre>CSVParser parser = ur.getCSVParser();</pre>
<code>.getCSVParser(false)</code>	returns a <code>CSVParser</code> object for this opened web page, assuming it contains comma separated values <i>without</i> a header row	<pre>CSVParser parser = ur.getCSVParser(false);</pre>

[DirectoryResource](#)

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
<code>.selectedFiles()</code>	returns an <code>Iterable</code> that provides access to each of the files selected by the user one at a time	<pre>for (File f : dr.selectedFiles()) { // process each file in turn }</pre>





[Pixel](#)

This class provides methods for accessing and changing a [color within an image](#). A `Pixel` can only be created by copying the values of an existing `Pixel`:

- `new Pixel(otherPixel)`, where `otherPixel` most likely was returned from the `ImageResource` method `getPixel(x, y)`

For these examples, assume

- `pix1` is a pixel at coordinate (100, 200) representing the color Duke blue, with RGBA values of (0, 26, 87, 255) 
- `pix2` is a pixel at coordinate (300, 400) representing the color white, with RGBA values of (255, 255, 255, 255) 

Method name	Description	Example
<code>.getX()</code>	returns the pixel's x-coordinate within the image	<code>pix1.getX()</code> is 100
<code>.getY()</code>	returns the pixel's y-coordinate within the image	<code>pix1.getY()</code> is 200
<code>.getRed()</code>	returns the value of the pixel's red component (always in the range 0-255)	<code>pix1.getRed()</code> is 0
<code>.getGreen()</code>	returns the value of the pixel's green component (always in the range 0-255)	<code>pix1.getGreen()</code> is 26
<code>.getBlue()</code>	returns the value of the pixel's blue component (always in the range 0-255)	<code>pix1.getBlue()</code> is 87
<code>.getAlpha()</code>	returns the value of the pixel's alpha, or transparency, component (always in the range 0-255)	<code>pix1.getAlpha()</code> is 255
<code>.setRed(newR)</code>	changes the value of the pixel's red component to <code>newR</code> (if <code>newR</code> is not in the range of 0-255 it is changed to be in that range)	<code>pix1.setRed(255)</code> changes the color to (255, 26, 87, 255) 
<code>.setGreen(newG)</code>	changes the value of the pixel's green component to <code>newG</code> (if <code>newG</code> is not in the range of 0-255 it is changed to be in that range)	<code>pix1.setGreen(255)</code> changes the color to (0, 255, 87, 255) 
<code>.setBlue(newB)</code>	changes the value of the pixel's blue component to <code>newB</code> (if <code>newB</code> is not in the range of 0-255 it is changed to be in that range)	<code>pix1.setBlue(255)</code> changes the color to (0, 26, 255, 255) 
<code>.setAlpha(newA)</code>	changes the value of the pixel's alpha, or transparency, component to <code>newA</code> (if <code>newA</code> is not in the range of 0-255 it is changed to be in that range)	<code>pix1.setAlpha(100)</code> changes the color to (0, 26, 87, 100) 

ImageResource

This class provides methods for accessing attributes of an image, including each pixel in the image. You can create an `ImageResource` in a variety of ways:

- `new ImageResource()`, opens a dialog box prompting you to select an image file on your computer
- `new ImageResource("path/to/image.jpg")`, uses the given `String` to find an image file on your computer or within your BlueJ project
- `new ImageResource(existingFile)`, uses the given `File` (typically returned by using a `DirectoryResource`)
- `new ImageResource(width, height)`, creates an empty image (all black) of the given size in pixels
- `new ImageResource(otherImage)`, creates an image that is an exact copy of `otherImage`

For these examples, assume the variable `logo` has the value of the image to the right. It is 100 pixels wide and 85 pixels tall.



Method name	Description	Example
<code>.getWidth()</code>	returns the image's width, or number of pixels in the X direction	<code>logo.getWidth()</code> is 100
<code>.getHeight()</code>	returns the image's height, or number of pixels in the Y direction	<code>logo.getHeight()</code> is 85
<code>.getPixel(x, y)</code>	returns the pixel in this image at the coordinate (x, y)	<code>logo.getPixel(0, 0)</code> is the pixel (255, 255, 255, 255)
<code>.setPixel(x, y, pixel)</code>	copies the RGBA values from the given pixel into pixel at the (x,y) coordinates given	<code>logo.setPixel(50, 42, pix2)</code> changes the color to white
<code>.pixels()</code>	returns an <code>Iterable</code> that provides access to each of the pixels in the image, starting in the upper-left corner and moving down to the lower-right corner	<pre>for (Pixel p : logo.pixels()) { // process each pixel in turn }</pre>
<code>.draw()</code>	draws the image in a separate window	<code>logo.draw();</code>
<code>.save()</code>	saves the changes made to this image using its current file name	<code>logo.save();</code>
<code>.getFileName()</code>	returns the current file name of this image, or the empty string, "", if it was created as an empty image	<pre>String name = logo.getFileName();</pre>
<code>.setFileName(newName)</code>	changes the current file name of this image to the given name, useful when saving your changed image without modifying the original image on your computer	<code>logo.setFileName("changed.jpg");</code>

StorageResource

This class provides methods for storing and accessing a list of strings of any length. Generally you will start by creating an empty `StorageResource`, then adding string values as you find them in a file or web page:

- `new StorageResource()`, creates an empty list
- `new StorageResource(otherList)`, creates a list that is an exact copy of `otherList`

For these examples, assume the variable `sr` has been created.

Method name	Description	Example
<code>.add(item)</code>	adds the given <code>item</code> to the end of the list of strings	<pre>sr.add("first!"); sr.add("next ...");</pre>
<code>.size()</code>	returns the number of strings stored in this list	<code>sr.size()</code> is 2 (after the example above)

		<code>sr.size()</code> is 0 (immediately after <code>clear()</code> is called)
<code>.data()</code>	returns an <code>Iterable</code> that provides access to each string in the list one at a time	<pre>for (String item : sr.data()) { // process each string in turn }</pre>
<code>.contains(item)</code>	returns true only if the given <code>item</code> is in the list	<pre>sr.contains("first!") is true sr.contains("last") is false</pre>
<code>.clear()</code>	removes all strings from this list, making it empty	<code>sr.clear();</code>

[RangeResource](#)

This class provides methods for accessing a sequence of numbers within a given range. You can create a `RangeResource` in a variety of ways:

- `new RangeResource(end)`, create a sequence of numbers starting at 0 and going up to, but not including, `end`
- `new RangeResource(start, end)`, create a sequence of numbers starting at `start` and going up to, but not including, `end`
- `new RangeResource(start, end, increment)`, create a sequence of numbers starting at `start` and going up to, but not including, `end`, counting by `increment`

For these examples, assume the variable `rr` has been created.

Method name	Description	Example
<code>.sequence()</code>	returns an <code>Iterable</code> that provides access to each number in the sequence	<pre>for (int i : rr.sequence()) { // process each number in turn }</pre>

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](#)

This class provides methods for accessing individual data values in a line of data within [a CSV formatted file](#). You cannot create a `CSVRecord` directly, 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](#).

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
<code>.get(columnName)</code>	returns a <code>String</code> , the data in this record corresponding to the column with the given <code>columnName</code> it is an error if the <code>columnName</code> does not exist in the header row (or does not have the same case)	<code>rec.get("Name")</code> is "Fred" <code>rec.get("Food")</code> is "Pizza"
<code>.get(columnIndex)</code>	returns a <code>String</code> , the data in this record corresponding to the column at the given <code>columnIndex</code> note, the index of the first data value is 0	<code>rec.get(0)</code> is "Fred" <code>rec.get(3)</code> is "13"
<code>.size()</code>	returns the number of values in this record	<code>rec.size()</code> is 4

CSVParser

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
<code>.getCurrentLineNumber()</code>	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

This class provides methods for accessing a sequence of characters of any length.

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

Name	Returns	Example
<code>.equals(other)</code>	returns true only if this string has the same characters and in the same order as the other string	<code>s.equals("Colorful")</code> is true <code>s.equals("colorful")</code> is false
<code>.equalsIgnoreCase(other)</code>	returns true only if this string has the same characters and in the same order as the other string, regardless of case	<code>s.equalsIgnoreCase("Colorluf")</code> is false <code>s.equalsIgnoreCase("color")</code>

		ful") is true
.length()	returns number of characters in this string	s.length() is 8 "".length() is 0
.indexOf(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(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(startIndex)	returns a string with the characters of this string, starting from startIndex and 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(startIndex, endIndex)	returns a string with the characters of this string, starting from startIndex and 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()	returns a string with the same characters as this string, but with all letters lowercased	s.toLowerCase() is "colorful"
.toUpperCase()	returns a string with the same characters as this string, but with all letters uppercased	s.toUpperCase() is "COLORFUL"
.startsWith(prefix)	returns true only if this string starts with given prefix	s.startsWith("Color") is true s.startsWith("cool") is false
.endsWith(suffix)	returns true only if this string ends with given suffix	s.endsWith("ful") is true s.endsWith("fool") is false

Math

The class provides methods for performing common numeric functions.

For these examples, assume the variable `a` has the value 25, and `b` has the value -9

Name	Returns	Example
------	---------	---------

<code>.max (num1, num2)</code>	returns the larger of two given numbers	<code>Math.max (a, b)</code> is 25
<code>.min (num1, num2)</code>	returns the smaller of two given numbers	<code>Math.min (a, b)</code> is -9
<code>.abs (num)</code>	returns the absolute value of the given number	<code>Math.abs (a)</code> is 25 <code>Math.abs (b)</code> is 9
<code>.sqrt (num)</code>	returns the positive square root of the given number	<code>Math.sqrt (a)</code> is 5 <code>Math.sqrt (Math.abs (b))</code> is 3

File

This class is Java's standard way to access a file on your computer, but it is not easy to use so, for this course, we have provided an alternate, `FileResource`. Most likely you will not call any methods directly on a `File` object returned from the `DirectoryResource` method `selectedFiles()`, but just use it to create another `Resource` object (like an image). In any case, here are a few useful methods.

Name	Returns
<code>.getCanonicalPath ()</code>	returns the unique name of this file, i.e., where it is on the computer
<code>.getName ()</code>	returns the name of this file, not including where it is on the computer
<code>.length ()</code>	returns the length of this file

Standard Java Operators

Arithmetic Operators

Operator	Description	Example
+	addition	4 + 5 is 9
-	subtraction	9 - 5 is 4
*	multiplication	3 * 5 is 15
/	division	6 / 3 is 2 6 / 4 is 1 6.0 / 4 is 1.5
%	mod, or remainder	5 % 3 is 2
<i>A shorthand version of each of these exists to update a variable that appears on both sides of the assignment. For example, the statement: <code>x = x + 1</code> can be shortened to: <code>x += 1</code></i>		<code>x = 1</code> sets x to 1 <code>x += 1</code> sets x to 2 <code>x *= 2</code> sets x to 4

Comparing Primitive Values

Operator	Description	Example
<code>==</code>	is equal to	<code>3 == 3</code> is true

<code>!=</code>	is not equal to	<code>3 != 3</code> is false
<code>>=</code>	is greater than or equal to	<code>4 >= 3</code> is true
<code><=</code>	is less than or equal to	<code>4 <= 3</code> is false
<code>></code>	is strictly greater than	<code>4 > 3</code> is true
<code><</code>	is strictly less than	<code>3 < 3</code> is false

Combining Comparisons - Logic Operators

For these examples, assume the variable `x` has the value 5.

Operator	Description	Example
<code> </code>	returns true if at least one part of it is true note, if the first comparison is true, subsequent comparisons are not done	<code>(x < 3 x > 7)</code> is false <code>(x < 3 x < 7)</code> is true
<code>&&</code>	returns true only if both parts of it are true note, if the first comparison is false, subsequent comparisons are not done	<code>(x > 3 && x < 7)</code> is true <code>(x > 3 && x > 7)</code> is false
<code>!</code>	flips the value of a comparison or boolean	<code>(! x == 5)</code> is false

Converting Between Types

<code>Integer.parseInt(s)</code>	turn the String <code>s</code> into an integer value note, this can fail, e.g., <code>Integer.parseInt("abc")</code> throws an exception	<code>Integer.parseInt("123")</code> is the number 123
<code>Double.parseDouble(s)</code>	turn the String <code>s</code> into a real valued number note, this can fail, e.g., <code>Double.parseDouble("abc")</code> throws an exception	<code>Double.parseDouble("2.46")</code> is the number 2.46
<code>(int)x</code>	turn <code>x</code> into an integer value by truncating the fractional part of the number	<code>(int) 123.6</code> is 123
<code>(double)x</code>	turn <code>x</code> into a real valued number, for example if you wanted to calculate the average of several integer values	<code>(double) 123</code> is 123.0

Background Information

What is an Iterable?

Iterators provide a way to work with data where each item of the data is processed in turn, considering each item one at a time.

For example, suppose you wanted to add a bunch of numbers, like 3, 4, 7, 3, 1, 8, 10. It is possible that you would solve this by writing one big addition problem:

```
  3
  4
  7
  3
  1
  8
+ 10
-----
```

It is more likely, however, that you would add the numbers one at a time, like this:

$3 + 4 = 7$; $7 + 7 = 14$; $14 + 3 = 17$; $17 + 1 = 18$; $18 + 8 = 26$; $26 + 10 = 36$

It is generally easier to think about it one item at a time, so using this "running total" way of processing helps us iterate over (or consider individually) the numbers. In the same way, it is typically easier to write an algorithm which processes only one item of data at a time.

In Java, data is represented in a wide variety of ways, not just numbers. We may need to work with each word in a file, darken each pixel in an image, or open all files in a directory. In each case, we need a way to access the elements of the data one at a time, in order. Iterables provide a convenient way to do this because, no matter what type of things are in the data, they give one standard syntax to access each thing.

Imagine a vending machine. The machine maintains a line of products, and when a customer buys something, it dispenses the next one in line. You probably feel confident that you could use such a machine, right? Notice that we have not disclosed what product is being vended. It does not really matter. You are able to easily use any vending machine, whether it is dispensing soda, chips, novels (now seen in airports), or live bait (at a fishing spot - really!). That's because the way the machine works does not generally depend on what the product is. That is the beauty of Iterables. Once you learn to use them, you will feel confident that you can process any kind of data that you encounter.