Writing a text file

Class `java.io.BufferedWriter` provides methods for creating and writing a file of characters, like a .txt file. One can create a `BufferedWriter` for a `Path` object `p` using:

```
BufferedWriter bf= Files.BufferedWriter(p);
```

The class has three methods of importance here:

`p.write(s, k, len);`	// Here, `s` is a String; write the substring `s[k..k+len-1]` to the file.
`p.newLine();`	// Write a line separator —whatever your OS uses as a separator.
`p.close();`	// Close the file. Should be called when no more is to be written on the file.

The class is called a *buffered* writer because it "buffers" the text. When a call on `p.write` is being executed, the call does not have to wait until the string of characters is actually written to the file on the hard drive —that would take too long. Instead, the characters are added to a buffer, and the call on `p.write` then terminates. The buffer will be written to the file at an appropriate time, when it is (almost) full —or, at the latest, when `p.close` is called.

Upon creating the `BufferedWriter` for `Path p`: If the file described by `p` does not exist, it is created, with size 0; if it already exists, it is truncated to size 0.

PrintWriter: a solution to two problems with BufferedWriter

There are two problems with class `BufferedWriter`. First, only `String` values can be written using procedure `write`. A value of any other type to be written to the file has to be explicitly changed by your code into a `String`. We have all used procedure `System.out.println(…)`, putting any expression we want as the argument. That name `println` is so overloaded that a value of any type can be given as an argument. Why doesn't `Buffered-Writer` allow the same flexibility?

Second, newline characters have to be written explicitly.

To get around these two problems, wrap the `BufferedWriter` in an object of class `java.io.PrintWriter`:

```
PrintWriter pr= new PrintWriter(bf);
```

`PrintWriter` has two overloaded procedures `pr.print` and `pr.println`, which work exactly like their counter-parts in `System.out`.[1] That's all there is to solving the two problems.

An example

Put this code in some method in an Eclipse project.

```
Path p= Paths.get("number.txt");
PrintWriter pr;
try {
    pr= new PrintWriter(Files.newBufferedWriter(p));
} catch (IOException io) {
    throw new RuntimeException("newBufferedWriter threw IO Exception");
}
for (int k= 0; k < 10; k= k+1) {
    pr.println(k);
}
pr.close();
```

This code will write a file `number.txt` in the Eclipse project directory. It will contain 10 lines, each containing one of the numbers in 0..9. If the file already exists, it is effectively replaced by the new one.

The assignment to pr has been placed in a try-block because the call `Files.newBufferedWriter(p)` may throw an `IOException`. If we don't use the try-statement, the method will need a throws clause. We would rather see the complication here rather than need the throws clause on one or more methods.

Don't forget the statement `pr.close();` at the end. This will flush the buffer, writing out any characters re-maining in it, and then close this resource.

[1] Methods `printf` and `format` are available too, and work as they do in `System.out`.

Appending to a File

It is possible to append to an existing file instead of writing over the existing file. This requires *one simple* change. Write the creation of a new `BufferedWriter` as

```
Files.newBufferedWriter(p, StandardOpenOption.APPEND)
```

The second argument, `StandardOpenOption.APPEND`, indicates that an existing file should *not* be deleted and characters should be added to it. Of course, it also works if the file doesn't exist yet.

Wow, that's all there is to it? Yes!

Here's an example. The following code appends the integers in 0..9 to an existing file.

```
Path p= Paths.get("number.txt");
PrintWriter pr;
try {
    pr= new PrintWriter(Files.newBufferedWriter(p, StandardOpenOption.APPEND));
} catch (IOException io) {
    throw new RuntimeException("newBufferedWriter threw IO Exception");
}
for (int k= 0; k < 10; k= k+1) {
    pr.println(k);
}
pr.close();
```