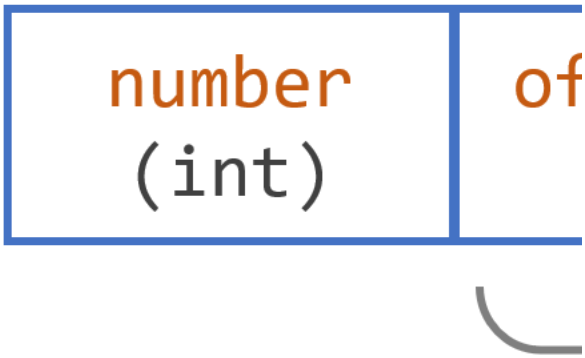
Putting it All Together

Introducing the Shakespeare Sonnet Example

Shakespeare wrote a number of plays and 154 sonnets, that you can find here on the Gutenberg website. Here is the first

```
From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thy self thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
  Pity the world, or else this glutton be,
  To eat the world's due, by the grave and thee.
```

This use case consists in creating a file to store them all, in a compressed way. Here is the format of the file that you nee



This format is composed of several elements.

1. The total number of sonnets. It is very unlikely that Shakespeare writes any more sonnet (he died in 1616), but you stil
2. For each sonnet, you want to write two elements: an offset and a length. The length is the number of bytes you need t
3. And then comes the text of each sonnet, compressed with GZIP.

This file format stores text files in a compressed form, and integer numbers. It requires several elements of the Java I/O A

Reading the Sonnets Text File

There are two ways to read this text file. You can just download it and store it locally on your machine. Or you can write sc

Here is the code to read it online. It is built on the HttpClient API. It produces an `InputStream` that you will convert to a `Re`

```
URI sonnetsURI = URI.create("https://www.gutenberg.org/cache/epub/1041/pg1041.txt");
HttpRequest request =
        HttpRequest.newBuilder(sonnetsURI)
                .GET()
                .build();
HttpClient client =
        HttpClient.newBuilder().build();
```

```
HttpResponse<InputStream> response = client.send(request, HttpResponse.BodyHandlers.ofInputStream());
InputStream inputStream = response.body();
```

Here is the code to read is from a file, using the `Files` factory class.

```
Path path = Path.of("files/sonnets.txt");
BufferedReader reader = Files.newBufferedReader(path);
```

None of these two pieces of code are complete: the exception handling part is missing, as well as the closing of the resou

Analyzing the Sonnets Text File

First, you need to read and analyze the text file provided on the Gutenberg website, and to read the text of the sonnets.

The text of the sonnets starts on line 33 of the text file. Then the file is structured as follow:

1. some blank lines,
2. the number of the sonnet, written as a roman number,
3. then some more blank lines,
4. and then the text of the sonnet itself.

You know that there are no more sonnets to read when you encounter the following line.

```
*** END OF THE PROJECT GUTENBERG EBOOK THE SONNETS ***
```

To tackle this problem, you can decorate the `BufferedReader` class, keeping its features and adding your own. There are

1. skipping the first lines of the text file,
2. skipping the sonnet header,
3. and reading the text of the sonnet, checking if you have reached the end of the file.

To read the sonnets, you can write the following code. Two pieces are missing: the `SonnetReader` class and the `Sonnet` cl

```
int start = 33;

List<Sonnet> sonnets = new ArrayList<>();

try (var reader = new SonnetReader(inputStream);
) {
    reader.skipLines(start);
    List<Sonnet> sonnet = reader.readNextSonnet();
    while (sonnet != null) {
        sonnets.add(sonnet);
        sonnet = reader.readNextSonnet();
    }

} catch (IOException e) {
    e.printStackTrace();
}

System.out.println("# sonnets = " + sonnets.size());
```

The `SonnetReader` class is a decoration of the `BufferedReader` class. Here is an example of the code you can write.

```
class SonnetReader extends BufferedReader {

    public SonnetReader(Reader reader) {
        super(reader);
    }

    public SonnetReader(InputStream inputStream) {
        this(new InputStreamReader(inputStream));
    }

    public void skipLines(int lines) throws IOException {
        for (int i = 0; i < lines; i++) {
            readLine();
        }
    }

    private String skipSonnetHeader() throws IOException {
        String line = readLine();
        while (line.isBlank()) {
            line = readLine();
        }
        if (line.equals("*** END OF THE PROJECT GUTENBERG EBOOK THE SONNETS ***")) {
            return null;
        }
        line = readLine();
        while (line.isBlank()) {
            line = readLine();
        }
        return line;
    }

    private Sonnet readNextSonnet() throws IOException {
        String line = skipSonnetHeader();
        if (line == null) {
            return null;
        } else {
            var sonnet = new Sonnet();
            while (!line.isBlank()) {
                sonnet.add(line);
                line = readLine();
            }
            return sonnet;
        }
    }
}
```

Running this code you display the following on your console.

```
# sonnets = 154
```

The `skipLines()` method is used to skip the file header that contains some technical and legal information on the file itse

The `skipSonnetHeader()` method reads and throws away the header of each sonnet in the file. It is composed of some b

The `readNextSonnet()` method read the text of the sonnet. There is no blank line in this text. If a blank line is met, then th

This class creates an instance of the `Sonnet` class, which is the following.

```
class Sonnet {
    private List<String> lines = new ArrayList<>();

```

```
    public void add(String line) {
        lines.add(line);
    }
}
```

This class is a simple wrapper on a `List<String>` with a simple `add(String)` method. Using this kind of simple class ma

Because it is a decoration of the `BufferedReader` class, your `SonnetReader` class can be used in a *try-with-resources* stat

Writing a Single Compressed Sonnet

Let us begin by writing a single sonnet to a compressed file.

This compressed file is a binary file, compressed with GZIP. Fortunately, the Java I/O API gives you a `GZIPOutputStream` c

You can add the following method to the `Sonnet` class.

```
byte[] getCompressedBytes() throws IOException {
    ByteArrayOutputStream bos = new ByteArrayOutputStream();
    try (GZIPOutputStream gzos = new GZIPOutputStream(bos);
         PrintWriter printWriter = new PrintWriter(gzos);) {

        for (String line : lines) {
            printWriter.println(line);
        }
    }

    return bos.toByteArray();
}
```

This method writes the lines of a sonnet in a `ByteArrayOutputStream`, decorated with a `GZIPOutputStream`, itself decorat

Even if no I/O resource is used in this method, using a *try-with-resources* statement is still very useful: it will flush for you t

Writing all the Sonnets

Writing all the sonnets consists in concatenating all the compressed sonnets into one array of bytes, and storing the offs

Once you have all this information, writing the bytes can be done with a plain `BufferedOutputStream`, and writing the offs

You can write the following code to create the final file.

```
int numberOfSonnets = sonnets.size();
Path sonnetsFile = Path.of("files/sonnets.bin");
try (var sonnetFile = Files.newOutputStream(sonnetsFile);
     var dos = new DataOutputStream(sonnetFile);) {

    List<Integer> offsets = new ArrayList<>();
    List<Integer> lengths = new ArrayList<>();
    byte[] encodeSonnetsBytesArray = null;

    try (ByteArrayOutputStream encodedSonnets = new ByteArrayOutputStream();) {
        for (Sonnet sonnet : sonnets) {
            byte[] sonnetCompressedBytes = sonnet.getCompressedBytes();

```

```
            offsets.add(encodedSonnets.size());
            lengths.add(sonnetCompressedBytes.length);
            encodedSonnets.write(sonnetCompressedBytes);
        }

        dos.writeInt(numberOfSonnets);
        for (int index = 0; index < numberOfSonnets; index++) {
            dos.writeInt(offsets.get(index));
            dos.writeInt(lengths.get(index));
        }
        encodeSonnetsBytesArray = encodedSonnets.toByteArray();
    }
    sonnetFile.write(encodeSonnetsBytesArray);

} catch (IOException e) {
    e.printStackTrace();
}
```

The first part of this code loops through all the sonnets and compress them to a first array of bytes. Then the offset and t

At the end of the day, all you need to do is follow the format of the file, that is:

1. write the number of the sonnets,
2. for each sonnet: write the offset and the length,
3. then write the array containing all the compressed sonnets.

Note that the offsets are computed from the start of the array containing all the compressed sonnets, not the start of the

Reading a Single Sonnet

Reading back a single sonnet consists in locating the right compressed array of bytes in the file, and decoding it. The reac

Let us begin by writing the code to read the number of sonnets, and for each sonnet, the offset and the length.

```
Path path = Path.of("files/sonnets.bin");

try (var file = Files.newInputStream(path);
     var bis = new BufferedInputStream(file);
     var dos = new DataInputStream(file);) {

        int numberOfSonnets = dos.readInt();
        System.out.println("numberOfSonnets = " + numberOfSonnets);
        List<Integer> offsets = new ArrayList<>();
        List<Integer> lengths = new ArrayList<>();
        for(int i = 0; i < numberOfSonnets; i++) {
            offsets.add(dos.readInt());
            lengths.add(dos.readInt());
        }

        // At this point, you have the offests and the lengths of
        // all the sonnets
}
```

Suppose you need to read the sonnet number 75. What you need to do is to skip the sonnets before this one, and read the

Skipping a fixed number of elements from an I/O stream is a little tricky. You need to keep in mind that a stream can be ve

```
long skip(BufferedInputStream bis, int offset) throws IOException {
    long skip = 0L;
    while (skip < offset) {
        skip += bis.skip(offset - skip);
    }
    return skip;
}
```

The same goes for the reading of a fixed amount of bytes. It is possible that the amount of bytes read by the input stream

```
byte[] readBytes(BufferedInputStream bis, int length) throws IOException {
    byte[] bytes = new byte[length];
    byte[] buffer = new byte[length];
    int read = bis.read(buffer);
    int copied = 0;
    while (copied < length) {
        System.arraycopy(buffer, 0, bytes, copied, read);
        copied += read;
        read = bis.read(buffer);
    }
    return bytes;
}
```

With these two methods, you can then add the following code after the reading of the offsets and the lengths.

```
int sonnet = 75; // the sonnet you are reading
int offset = offsets.get(sonnet - 1);
int length = lengths.get(sonnet - 1);

skip(bis, offset);
byte[] bytes = readBytes(bis, length);

try (ByteArrayInputStream bais = new ByteArrayInputStream(bytes);
     GZIPInputStream gzbais = new GZIPInputStream(bais);
     InputStreamReader isr = new InputStreamReader(gzbais);
     BufferedReader reader = new BufferedReader(isr);) {

    List<String> sonnetLines = reader.lines().toList();
    sonnetLines.forEach(System.out::println);
}
```

This code reads the bytes of the compressed sonnet. It then builds a `ByteArrayInputStream` on this array, and decorates

Here is the text of this sonnet, that should be printed on your console.

```
  So are you to my thoughts as food to life,
  Or as sweet-season'd showers are to the ground;
  And for the peace of you I hold such strife
  As 'twixt a miser and his wealth is found.
  Now proud as an enjoyer, and anon
  Doubting the filching age will steal his treasure;
  Now counting best to be with you alone,
  Then better'd that the world may see my pleasure:
  Sometime all full with feasting on your sight,
  And by and by clean starved for a look;
  Possessing or pursuing no delight,
  Save what is had, or must from you be took.
    Thus do I pine and surfeit day by day,
    Or gluttoning on all, or all away.
```

Last update: *January 25, 2023*

Back to Tutorial List