Files and Streams

Working with files in Java
(Quick refresher)

Classes for working with files are part of the `java.io` package.

- General I/O, beyond files: keyboard input, screen output, abstractions for manipulating bytes

Main abstraction: **stream** (a sequence of bytes)

You have used streams before:

- `System.out` is of type `java.io.PrintStream`

There are many classes for working with streams and, particularly, files

A first example

Reading a file with a bit of structure

Contents of file `entries.txt`

```
Nobara, 16, Hammer and Nails
Itadori, 15, Hands and Feet
Fushiguro, 15, Cursed Spirits
Toudou, 18, Boogie Woogie
Gojo, 28, Too overpowered to mention
```

One entry per line (terminator depends on the OS), three comma-separated parts per entry

Before we continue, a quick disclaimer:

Not every file is a text file!

Some files are **binary**. Cheaper for the computer to process, but unreadable to humans. Here we focus on **text** files

Nobara, 16, Hammer and Nails\n

```
Nobara, 16, Hammer and Nails\n
```

For a text file, bytes are read so as to form characters. Each roman char. (w/o accent) takes **one byte**

```
Ñobara, 16, Hammer and Nails\n
```

Roman characters with accents in languages such as Portuguese and Spanish take **two bytes**

```
野薔薇, 16, Hammer and Nails\n
```

Chinese/Japanese characters
may take **two or three bytes**

```
野薔薇, 16, Hammer and Nails\n
```

Furthermore, the end of the line is a character, although we don’t see it. Other examples are tabs and carriage returns

```
野薔薇, 16, Hammer and Nails\n
```

We will also need to deal with how the parts of each entry are delimited.

Two tasks:

- Reading the file line by line
- Parsing the contents of each line

Two tasks:

- Reading the file line by line
- Parsing the contents of each line

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new Buffer
            System.out.println("Reading fil

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

`IOException` is the general class representing input/output errors

`FileReader` is a class for reading files character by character. All classes whose name ends with **Reader** (i.e., implementing the `Reader` interface) in the **`java.io`** package aim to support reading text files.

`BufferedReader` reads sequences of characters into a buffer. Allows the programmer to read the contents of a file line by line, by identifying specific characters that end a line automatically.

We do not need to worry about bytes, **just characters.**

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
                                                    ll) { ... }
```

```
                                                    d: " + e.getMessage());
```

This line does the following:
- Opens a text file named “`entries.txt`” for reading, using a `FileReader`. FileReaders read files character by character.
- Wraps the `FileReader` object with a `BufferedReader`, so as to be able to read line by line, by keeping read characters in a buffer.
- Does all of this within the context of a `try`-with-resources block. This guarantees that, by the end of the block, the file will be released, even if exceptions occur.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExam
    public static void main(Strin
        String filePath = "entri

        try (BufferedReader br      w BufferedReader(new FileReader(filePath))) {
            System.out.printl  Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

Reads the file line by line. Each call to `readLine()` returns a `String` object corresponding to a line of the file (a sequence of characters ending in a line break) and assigns it to the line local variable. The line break character is not included. Pay attention to the loop condition. This is an assignment, not an equality comparison. Furthermore, each call to `readLine()` reads a consecutive line. If we want to go back in the file, we will need to invoke the `reset()`

```java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

An alternative using the Scanner class.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderExampleWithScanner {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (Scanner scanner = new Scanner(new FileReader(filePath))) {

            System.out.println("Reading file entries:");

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                ...
            }
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
```

Two tasks:

- Reading the file line by line
- Parsing the contents of each line

Two tasks:

- Reading the file line by line
- Parsing the contents of each line

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class FileReaderSimpleExample {
    public static void main(String[] args) {
        String filePath = "entries.txt";

        try (BufferedReader br = new BufferedReader(new FileReader(filePath))) {
            System.out.println("Reading file entries:");

            String line;
            while ((line = br.readLine()) != null) { ... }

            ...
        } catch (IOException e) {
            System.out.println("An error occurred: " + e.getMessage());
        }
    }
}
```

```
...
while ((line = br.readLine()) != null) {
    // Parse the line without regular expressions
    int firstComma = line.indexOf(',');
    int secondComma = line.indexOf(',', firstComma + 1);

    if (firstComma != -1 && secondComma != -1) {
        String name = line.substring(0, firstComma).trim();
        String age = line.substring(firstComma + 1, secondComma).trim();
        String profession = line.substring(secondComma + 1).trim();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Specialty: " + profession);
        System.out.println("-------------");
    } else {
        System.out.println("Invalid entry: " + line);
    }
}
...
```

```
...
while ((line = br.readLine()) != null) {
    // Parse the line without regular expressions
    int firstComma = line.indexOf(',');
    int secondComma = line.indexOf(',', firstComma + 1);

    if (firstComma != -1 && secondComma != -1) {
        String name = line.substring(0, firstComma).trim();
        String age = line.substring(firstComma + 1, secondComma).trim();
        String profession = line.substring(secondComma + 1).trim();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Specialty: " + profession);
        System.out.println("-------------");
    } else {
        System.out.println("Invalid entry: " + line);
    }
}
...
```

```
...
while ((line = br.readLine()) != null) {
    // Split the line into tokens using the split method
    String[] tokens = line.split(","); // Split by comma

    if (tokens.length == 3) {
        String name = tokens[0].trim();
        String age = tokens[1].trim();
        String profession = tokens[2].trim();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Specialty: " + profession);
        System.out.println("-------------");
    } else {
        System.out.println("Invalid entry: " + line);
   }
}
...
```

```
...
while ((line = br.readLine()) != null) {
    // Split the line into tokens using the split method
    String[] tokens = line.split(","); // Split by comma

    if (tokens.length == 3) {
        String name = tokens[0].trim();
        String age = tokens[1].trim();
        String profession = tokens[2].trim();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Specialty: " + profession);
        System.out.println("-------------");
    } else {
        System.out.println("Invalid entry: " + line);
   }
}
...
```

```
...
while ((line = br.readLine()) != null) {
    // Split the line into tokens using the split method
        System.out.println("Invalid entry: " + line);
    }
}
...
```

If the processing of each part of the `String` is more complicated, it pays off to use the `StringTokenizer` class

You can find more about it in the Java API spec.

In practice:

This file structure (values separated by commas, or CSVs) is common and there are libraries to process them

`opencsv` (https://opencsv.sourceforge.net/)

Very popular in data science

What about writing to a file?

Use the `FileWriter` class

- To add more lines to an existing file, the second parameter of the constructor (`append`) should be `true`

It includes a `write(String)` method

Use `try`-with-resources and prepare for `IOExceptions`