Files and reading data

Learning Objectives

- You'll review reading keyboard input.
- You know what a file and a filesystem are, and are able to add an empty text file into the filesystem.
- You know how to create a write a program that reads data from a file.

A considerable amount of software is in one way or another based on handling data. Software created for playing music handles music files and those created for the purpose of image manipulation handle image files. Applications that run on the internet and mobile devices, such as Facebook, WhatsApp, and Telegram, handle user information that is stored in file-based databases. What these all have in common is that they read and manipulate data in one way or another. Also, the data being handled is ultimately stored in some format in one or more files.

Reading From the Keyboard

We've been using the `Scanner`-class since the beginning of this course to read user input. The block in which data is read has been a while-true loop where the reading ends at a specific input.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    String line = scanner.nextLine();

    if (line.equals("end")) {
        break;
    }
    // add the read line to a list for later
    // handling or handle the line immediately

}
```

In the example above, we pass system input (`System.in`) as a parameter to the constructor of the Scanner-class. In text-based user interfaces, the input of the user is directed into the input stream one line at a time, which means that the information is sent to be handled every time the user enters a new line.

Programming exercise:

Number of Strings

Points

1/1

Write a program that reads strings from the user until the user inputs the string "end". At that point, the program should print how many strings have been read. The string "end" should not be included in the number strings read. You can find some examples below of how the program works.

Sample output

```
I
have
a
feeling
that
I
have
written
this
wrong
before
end
11
```

Sample output

```
end
0
```

Exercise submission instructions

How to see the solution

The user input is read in string form. If we wanted to handle the input as integers, for instance, we'd have to convert it to another form. An example program has been provided below - it reads input from the user until the user inputs "end". As long as the user input is not "end" the inputs are handled as integers — in this case, the number is simply printed.

```java
Scanner scanner = new Scanner(System.in);

while (true) {
    String row = scanner.nextLine();

    if (row.equals("end")) {
        break;
    }

    int number = Integer.valueOf(row);
    System.out.println(row);
}
```

Programming exercise:

Cubes

Points

1/1

Write a program that reads strings from the user until the user inputs the string "end". As long as the input is not "end", the program should handle the input as an integer and print the cube of the number provided (i.e., number _ number _ number). Below are some sample outputs

Sample output

```
3
27
-1
-1
11
1331
end
```

Sample output

```
end
```

Exercise submission instructions

How to see the solution

Files and the Filesystem

Files are collections of data that live in computers. These files can contain, among other things, text, images, music, or any combination of these. The file format determines the content of the file as well as the program required to read the file. For example, PDF files are read with a program suited for reading PDF files, and music files are read with a program suited for reading music files. Each of these programs is made by humans, and the creators of these programs — i.e., programmers — also specify the file format as part of the work.

Computers have several different programs for browsing files. These programs are specific to the operating system. All programs used for browsing files make use of the filesystem of the computer in one way or another.

Our development environment provides us with the ability to browse the files of a project. In NetBeans you can take a look at all the files attached to a project by selecting the `Files` tab, which is found in the same place as the `Projects` tab. If the tab cannot be be found, it can be opened from the `Window` menu. Clicking the project to open it will reveal all its files.

Programming exercise:

Creating a New File

Points

1/1

NB! In this exercise, we won't be programming. Instead, you'll familiarize yourself with the `Files`-tab in NetBeans and how to create a new file.

Create a file called `file.txt` in the root folder (the folder containing the folder `src` and the file `pom.xml`) of the exercise template using the `Files`-tab in NetBeans. Edit the file and write the message `Hello, world!` on the first line of the file.

Exercise submission instructions

How to see the solution

The Concrete File Storage Format

Files exist on the hard drive of a computer, which is, in reality, a large set of ones and zeros, i.e., bits. Information is made up of these bits, e.g., one variable of type int takes up 32 bits (i.e., 32 ones or zeros). Modern terabyte-sized hard drives hold about 8 trillion bits (written out the number is 8,000,000,000,000). On this scale, a single integer is very small.

Files can exist practically anywhere on a hard drive, even separated into multiple pieces. The computer's **filesystem** has the responsibility of keeping track of the locations of files on the hard drive as well as providing the ability to create new files and modify them. The filesystem's main responsibility is abstracting the true structure of the hard drive; a user or a program using a file doesn't need to care about how, or where, the file is actually stored.

Reading From a File

Reading a file is done using the Scanner-class. When we want to read a file using the Scanner-class, we give the path for the file we want to read as a parameter to the constructor of the class. The path to the file can be acquired using Java's `Paths.get` command, which is given the file's name in string format as a parameter: `Paths.get("filename.extension")`.

When the `Scanner`-object that reads the file has been created, the file can be read using a while-loop. The reading proceeds until all the lines of the file have been read, i.e., until the scanner finds no more lines to read. Reading a file may result in an error, and it's for this reason that the process requires separate blocks - one for the `try`, and another to `catch` potential errors. We'll be returning to the topic of error handling later.

```java
// first
import java.util.Scanner;
import java.nio.file.Paths;

// in the program:

// we create a scanner for reading the file
try (Scanner scanner = new Scanner(Paths.get("file.txt"))) {

    // we read the file until all lines have been read
    while (scanner.hasNextLine()) {
        // we read one line
        String row = scanner.nextLine();
        // we print the line that we read
        System.out.println(row);
    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
```

A file is read from the project root by default (when `new Scanner(Paths.get("file.txt"))` is called), i.e., the folder that contains the folder `src` and the file `pom.xml` (and possibly other files as well). The contents of this folder can the inspected using the `Files`-tab in NetBeans.

Programming exercise:

Printing a File

Points 1/1

Write a program that prints the contents of a file called "data.txt", such that each line of the file is printed on its own line.

If the file content looks like so:

Sample data

```
In a
world
```

Then the program should print the following:

Sample output

```
In a
world
```

Exercise submission instructions

How to see the solution

Programming exercise:

Printing a Specified File

Points 1/1

Write a program that asks the user for a string, and then prints the contents of a file with a name matching the string provided. You may assume that the user provides a file name that the program can find.

The exercise template contains the files "data.txt" and "song.txt", which you may use when testing the functionality of your program. The output of the program can be seen below for when a user has entered the string "song.txt". The content that is printed comes from the file "song.txt". Naturally, the program should also work with other filenames, assuming the file can be found.

Sample output

```
Which file should have its contents printed?
song.txt
No option for duality
The old is where we come
Clockspeed is fast, but we'll survive
The new will overcome
We are challengers, not followers
We take the ball to build
Easy safe services
Are here to stay

Value for society
Value for life
For you and me
Tieto is here allright!
```

Exercise submission instructions

How to see the solution

In the example below, we read all the lines of the file "file.txt", which are then added to an ArrayList.

```java
ArrayList<String> lines = new ArrayList<>();

// we create a scanner for reading the file
try (Scanner scanner = new Scanner(Paths.get("file.txt"))) {

    // we read all the lines of the file
    while (scanner.hasNextLine()) {
        lines.add(scanner.nextLine());
    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}

// we print the total number of lines
System.out.println("Total lines: " + lines.size());
```

Programming exercise:

Guest List From a File

Points
1/1

The exercise template comes ready with functionality for the guest list application. It checks whether names entered by the user are on the guest list.

However, the program is missing the functionality needed for reading the guest list. Modify the program so that the names on the guest list are read from the file.

Sample output

```
Name of the file:
guestlist.txt

Enter names, an empty line quits.
Chuck Norris
The name is not on the list.
Jack Baluer
The name is not on the list.
Jack Bauer
The name is on the list.
Jack Bower
The name is on the list.
```

```
Thank you!
```

NB! The exercise template comes with two files, `names.txt` and `other-names.txt`, which have the following contents. Do not change the contents of the files!

names.txt:

```
ada
arto
leena
test
```

other-names.txt:

```
leo
jarmo
alicia
```

Exercise submission instructions

How to see the solution

Programming exercise:

Is it in the file?

Points 1/1

The exercise template comes with two files, `names.txt` and `other-names.txt`. Write a program that first asks the user for the name of the file to be read, after which the user is prompted for the string that they're looking for. The program then reads the file and searches for the desired string.

If the string is found, the program should print "Found!". If not, the program should print "Not found.". If reading the file fails (the reading ends in an error) the program should print the message "Reading the file " + file + " failed.".

Sample output

```
Name of the file:
names.txt
Search for:
Antti
Not found.
```

Sample output

```
Name of the file:
names.txt
Search for:
ada
Found!
```

Sample output

```
Name of the file:
nonexistent.txt
Search for:
test
Reading the file nonexistent.txt failed.
```

Exercise submission instructions

How to see the solution

Programming exercise:

Numbers From a File

Points

1/1

Write a program that prompts the user for a filename, as well as the upper and lower bounds for the accepted range of numbers. Then the program reads the numbers contained in the file (each number is on its own line) and only accounts for the numbers which are inside the given range. Finally, the program should print the number of numbers that were inside the given range.

You can convert a string-type integer read from a file into a proper integer using the command `Integer.valueOf` (just as when handling input from a user).

Sample output

```
File? numbers-1.txt
Lower bound? 15
Upper bound? 20
Numbers: 2
```

Sample output

```
File? numbers-1.txt
Lower bound? 0
Upper bound? 300
Numbers: 4
```

NB! The exercise template comes with two files, `numbers-1.txt` and `numbers-2.txt` that have the following contents. Do not change the contents of these files.

numbers-1.txt:

Sample data

```
300
9
```

```
20
15
```

numbers-2.txt:

Sample data

```
123
-5
12
67
-300
1902
```

Exercise submission instructions

How to see the solution

An Empty Line In a File

Sometimes an empty line finds it way into a file. Skipping an empty line can be done using the command `continue` and the `isEmpty`-method of the string.

In the example below, we read from a file

Reading data is straightforward.

```java
// we create a scanner for reading the file
try (Scanner scanner = new Scanner(Paths.get("henkilot.csv"))) {

    // we read all the lines of the file
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();

        // if the line is blank we do nothing
        if (line.isEmpty()) {
            continue;
        }

        // do something with the data

    }
} catch (Exception e) {
    System.out.println("Error: " + e.getMessage());
}
```

Reading Data of a Specific Format From a File

The world is full of data that are related to other data — these form collections. For example, personal information can include a name, date of birth and a phone number. Address information, on the other hand, can include a country, city, street address, postal number and so on.

Data is often stored in files using a specific format. One such format that's already familiar to us is comma-separated values (CSV) format, i.e., data separated by commas.

```
Scanner scanner = new Scanner(System.in);

while (true) {
    System.out.print("Enter name and age separated by a comma: ");
    String line = scanner.nextLine();

    if (line.equals("")) {
        break;
    }

    String[] parts = line.split(",");
    String name = parts[0];
    int age = Integer.valueOf(parts[1]);

    System.out.println("Name: " + name);
    System.out.println("Age: " + age);
}
```

The program works as follows:

Sample output

```
Enter name and age separated by a comma:
virpi,19
Name: virpi
Age: 19
Enter name and age separated by a comma:
jenna,21
Name: jenna
Age: 21
Enter name and age separated by a comma:
ada,20
Name: ada
Age: 20
```

Reading the same data from a file called `records.txt` would look like so:

```java
try (Scanner scanner = new Scanner(Paths.get("records.txt"))) {

    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();

        String[] parts = line.split(",");
        String name = parts[0];
        int age = Integer.valueOf(parts[1]);

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

Programming exercise:

Records From a File

Points

1/1

In this exercise, we'll be working with files stored in CSV-format that contain names and ages separated by commas. The file format may look like this:

Sample data

```
lily,3
anton,5
levi,4
amy,1
```

Your task is to write a program that first prompts the user for the name of the file they want to read. The program then prints the content of the file in the following way (we're assuming below that the output is from the above-mentioned file):

Sample output

```
Name of the file:
data.txt
lily, age: 3 years
anton, age: 5 years
levi, age: 4 years
amy, age: 1 year
```

NB! The word "year" should be formatted based on the age.

Exercise submission instructions

How to see the solution

Reading Objects From a File

Creating objects from data that is read from a file is straightforward. Let's assume that we have a class called `Person`, as well as the data from before.

Reading objects can be done like so:

```java
ArrayList<Person> people = new ArrayList<>();
```

```
try (Scanner scanner = new Scanner(Paths.get("records.txt"))) {

    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();

        String[] parts = line.split(",");
        String name = parts[0];
        int age = Integer.valueOf(parts[1]);

        people.add(new Person(name, age));
    }
}

System.out.println("Total amount of people read: " + people.size());
```

Reading objects from a file is a clear responsibility in and of itself, and should for that reason be isolated into a method. This is what we'll be doing in the next exercise.

Programming exercise:

Storing Records

Points 1/1

In this exercise, we'll be working with files stored in CSV format, which contain names and ages separated by commas. The file format may look like this:

Sample data

```
lily,3
anton,5
levi,4
amy,1
```

The exercise template already has a `Person` class, and the class `StoringRecords` has a body for the method `public static ArrayList<Person> readRecordsFromFile(String file)`. Implement the `readRecordsFromFile` method such that it reads the persons from the file passed as a parameter, and finally returns them in the list returned by the method.

The exercise template has a `main` method that you can use to test how your program works. In this exercise, only modify the method `readRecordsFromFile`.

Exercise submission instructions

How to see the solution

Programming exercise:

Sport Statistics

Points

2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

In this exercise, we'll be working with files stored in CSV format. Each line of the file contains the home team, visiting team, home team points, and visiting team points, all separated by commas.

You can see an example below of the file's contents. The file shown below is also included in the exercise template with the name "data.csv".

Sample data

```
ENCE,Vitality,9,16
ENCE,Vitality,16,12
ENCE,Vitality,9,16
ENCE,Heroic,10,16
SJ,ENCE,0,16
SJ,ENCE,3,16
FURIA,NRG,7,16
FURIA,Prospects,16,1
```

Write a program that prompts the user for a filename, after which it reads the match statistics from the file. The program then prompts the user for the name of a team, and prints the data specified in the following parts for that team.

Part 1: Games Played

Implement the ability to output the number of games played by any given team. We're using the above-mentioned **data.csv** file.

Sample output

```
File:
data.csv
Team:
FURIA
Games: 2
```

Sample output

```
File:
data.csv
Team:
ENCE
Games: 6
```

Part 2: Wins and Losses

Extend the program so that it has the ability to print the number of wins and losses of a given team. The winner of a game is the team that has gained more points from it.

You may assume that the games cannot be tied. Below, we're using the above-mentioned **data.csv** file.

Sample output

```
File:
data.csv
Team:
FURIA
Games: 2
```

```
Wins: 1
Losses: 1
```

Sample output

```
File:
data.csv
Name:
ENCE
Games: 6
Wins: 3
Losses: 3
```

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 4. Summary

Remember to check your points from the ball on the bottom-right corner of the material!

In this part: