Working with the file system

Martin Pola

Persistent storage

The file system is *persistent*, which means that what you store there will be kept until removal.

The memory, where objects and primitives live during program execution, is volatile to program interrupts and power outages.

Persistent storage

The file system is *persistent*, which means that what you store there will be kept until removal.

The memory, where objects and primitives live during program execution, is volatile to program interrupts and power outages.

Files are stored in directories (folders).

At the very basic level, all files are just sequences of bytes. To store text, one would typically apply some character encoding (e.g. UTF-8, ASCII or latin1).

File management in programming

Saving is known as *writing* in software development.

You cannot, however, just read or write – consider the following.

- Files can have permissions (read, read+write) that apply to various users
- Java programs run with the permissions from a user account

File management in programming

Saving is known as *writing* in software development.

You cannot, however, just read or write – consider the following.

- Files can have permissions (read, read+write) that apply to various users
- Java programs run with the permissions from a user account
- Files can be very large – too large for memory

Typical steps:

1. Open a *file handler* or *file descriptor* resource
2. Set the pointer by seeking to a place in the file
3. Perform reading or writing
4. Close the resource

Reading from a file

```java
RandomAccessFile r = new RandomAccessFile("f.txt",
                         "r");

r.seek(0);

byte[] bytes = new byte[(int) r.length()];
r.read(bytes);

r.close();

String content = new String(bytes);
```

Reading will advance the seek pointer by the number of read bytes.

Writing to file

```java

String content = "Hello world!"

RandomAccessFile r = new RandomAccessFile("f.txt",
                                "rw");

r.seek(0);
r.write(content.getBytes());

r.close();
```

Writing will advance the seek pointer by the number of written bytes.

It's possible to add content to the end of the file by seeking to the end (`r.seek(r.length())`) before writing.

Pathfinding

File paths can be *relative* or *absolute*. Relative paths start at the *working directory*. If the working directory changes, the relative path might point to the wrong location.

Pathfinding

File paths can be *relative* or *absolute*. Relative paths start at the *working directory*. If the working directory changes, the relative path might point to the wrong location.

- `../../Downloads/MyFile.java` – relative path
- `/Users/mpol/Downloads/MyFile.java` – absolute path
- `C:\Users\mpol\Downloads\MyFile.java` – absolute path

Note that you'd need to escape your backslashes on Windows!

```
    String myPath = "C:\\Users\\mpol\\...";
```

Working directory

Used as a baseline for relative paths.

Typically where you stood in your terminal when the program was executed. Change your working directory with `cd` in your terminal.

- `echo %cd%` in Windows command line
- `pwd` in Unix-like systems (e.g. OS X)

Use

```
System.getProperty("user.dir")
```

to see the current working directory.