File management

Filesystem management

In this document we are going to talk about the classes provided by Java to manage our file system. We will learn how to do some basic operations, such as:

- Check if a file exists or not, or check if it's a normal file or a directory
- Copy/Move/Delete files
- Move through the file system, from one folder to its parent folder, or list the complete list of files and folders of a given location.

1. File class

`File` class is the most basic (and ancient) class to deal with our filesystem. It also belongs to `java.io` package, and has some methods to:

- Create files
- Delete files
- List files/folders contained in a given folder
- Get the file size in bytes
- ...

For instance, this piece of code checks if a file exists in our system:

```
if (! (new File("example.txt")).exists() )
{
    System.err.println("File example.txt not found");
}
else
{
    // File exists
}
```

This other piece of code lists all the files and folders of a given folder in the system. For every subfolder, it marks it as *DIR*, and for every file, it prints its size in KB.

```
File location = new File("D:" + File.separatorChar + "Downloads");
File[] contents = location.listFiles();
for(File f: contents)
{
    System.out.print(f.getName());
    if (f.isDirectory())
        System.out.println (" (DIR)");
    else
        System.out.println (" " + (f.length() / 1024) + "KB");
}
```

Note that we can use `File.separatorChar` property from `File` class to represent the separator char of current operating system. This will be `\` in Windows, or `/` in Linux (although you can also use `/` in Windows systems).

2. Path and Paths

`Path` is an interface representing a path in the system (this is, a sequence of folders and subfolders pointing to a given location). It is a newer element in Java API, that belongs to `java.nio` package, and it provides some useful methods, such as:

- `startsWith` / `endsWith` : to check if current path starts or ends with a given subpath.
- `getParent` : to get parent folder of current path
- `getRoot` : to get initial, root folder of current path
- `iterator` : to explore every folder and subfolder of this path
- `toAbsolutePath` : to get the absolute path of current path (from the root point)
- ...

`Paths` class contains some static methods to deal with paths. It also belongs to `java.nio` package, and lets us, for instance, get a `Path` object from a given route:

```
Path location = Paths.get("/home/user/myFolder");
System.out.println("Parent folder is " + location.getParent().toString());
```

3. Files class

`Files` class is also a newer class in the Java API (it belongs to `java.nio` package) with a list of static methods to deal with our filesystem:

- `readAllLines(Path)` reads all the lines of the file and returns them in a `List`
- `copy(Path, Path)` or `move(Path, Path)` copy a file specified in a `Path` object into a `Path` destination (they only copy or move files, not folders)
- `delete(Path)` deletes a file (not a folder)

- `createDirectory(Path)` creates a new directory specified by the given `Path`
- `exists(Path)` checks if a given path exists (similar to File's `exists` method, but without having to create an object for this).
- ...

This piece of code reads the whole contents of a text file in one single line, and then we can just explore the `List` returned:

```java
try
{
    List<String> data = Files.readAllLines(Paths.get("data/file.txt"));
    for(String line: data)
    {
        ...
    }
}
catch (IOException ex)
{
    ...
}
```