Notatki - Streamy - FileReader oraz FileWriter

Spis treści

FileReader oraz FileWriter

`Reader` oraz `Writer` różnią się tym od `InputStream` oraz `OutputStream`, że pozwalają nam operować od razu na ciągach znaków. Kolejna rzecz to możliwość określenia kodowania jakie ma być użyte przy odczycie znaków z pliku. `Reader` oraz `Writer` są dedykowane do pracy na plikach tekstowych. Ich użycie jest bardzo podobne do `InputStream` oraz `OutputStream`.

Pokazujemy od razu przykład z `Buffered` bo w praktyce zależy nam przecież na wydajności 😀.

```
public class ReaderWriterExamples {

    public static void main(String[] args) throws IOException {
        File inputFile = new File("myInputFile.txt");
        File outputFile = new File("myOutputFile.txt");

        List<String> fileRead = readFile(inputFile);
        for (String line : fileRead) {
            System.out.println(line);
        }
        writeFile(fileRead, outputFile);
    }

    public static List<String> readFile(File inputFile) throws IOException {
        List<String> result = new ArrayList<>();
        try (BufferedReader reader = new BufferedReader(new FileReader(inputFile))) {
            String line = reader.readLine();
            while (line != null) {
                result.add(line);
                line = reader.readLine();
            }
        }
        return result;
    }

    public static void writeFile(List<String> fileRead, File outputFile) throws IOException {
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile))) {
            for (String line : fileRead) {
                writer.write(line);
                writer.newLine();
            }
        }
    }
}
```

Zwróćmy uwagę na parę kwestii:

- metoda `readLine()` zwraca `String`, a nie `int`. W związku z tym, `readLine()` w przypadku końca pliku nie zwróci `-1`, tylko `null`.
- przedstawiony kod jest wygodniejszy jeżeli chcemy operować na odczytanych wartościach. Nie trzeba ich rzutować na `char` i wczytujemy całe linijki na raz. Nie mówiąc już o kodowaniu polskich znaków i możliwości łatwej pracy na nich.
- w tym przypadku możemy zapisywać całą linijkę tekstu jednocześnie.

Widać już, że zarówno poprzednie przykłady jak i ten mogą służyć do tego samego, natomiast w przypadku operowania na danych tekstowych - `Reader` i `Writer` są wygodniejsze w użyciu.

Character encoding

Wracamy znowu do tematu kodowania znaków. Przypomnijmy sobie, że kodowanie znaków określa w jaki sposób znaki są kodowane i przechowywane w reprezentacji bajtowej i w jaki sposób z bajtów są one dekodowane z powrotem jako znaki.

Dlaczego ten aspekt jest tutaj istotny? Przypomnij sobie przykład z polskimi znakami i odczytem pliku

bezpośrednio przez `FileInputStream`. Dlatego właśnie `Reader` oraz `Writer` są wygodniejsze przy operacjach na plikach tekstowych.