Notatki - Streamy - PrintStream i PrintWriter

Spis treści

PrintStream i PrintWriter

`PrintStream` jest klasą, której założeniem jest ułatwienie formatowania danych w `OutputStreamie`, który jest pod spodem. Do tego różni się ona tym, że nie wyrzuca `IOException` jak poprzednio poznane klasy, a w przypadku błędu przestawia wewnętrzną flagę, którą można sprawdzać przy wykorzystaniu metody `checkError()`. Możliwe też jest utworzenie tej klasy w taki sposób aby `flush()` wykonywał się automatycznie po wywołaniu metody drukującej. Klasa ta służy do ułatwienia formatowania drukowanych wartości (np. do pliku albo na konsolę).

`PrintWriter` jest klasą, której założeniem jest ułatwienie formatowania danych w `Writerze`, który jest pod spodem. Zawiera wszystkie metody, które są dostępne w klasie `PrintStream`. Podobnie jak `PrintStream`, dzięki tej klasie możliwe jest ustawienie automatycznego flushowania przy każdym wywołaniu metody `print()`. Podobnie jak `PrintStream`, metody tej klasy nie wyrzucają wyjątków `IOException`, a ewentualne błędy można sprawdzić przez metodę `checkError()`. `PrintWriter` ma również konstruktor, który pozwala zrobić nakładkę `PrintWriter` na `OutputStream`.

Czyli podsumowując, obie te klasy są nakładkami, które dodają nam opcje prostszego formatowania tekstu oraz to, że przy każdym ich wywołaniu nie musimy obsługiwać `IOException`.

Trzymając się jednocześnie konwencji nazewnictwa można wywnioskować, że `PrintStream` zapisuje dane w formie bajtów, podczas gdy `PrintWriter` zapisuje dane przy wykorzystaniu znaków.

I dopiero teraz mogę powiedzieć, że od samego początku nauki używamy `PrintStreama`. Gdzie?

```
System.out.println("some text");
```

Gdy wejdziemy w zmienną `out` w klasie `System`, okaże się, że jest to `PrintStream` drukujący na ekranie. Oprócz zmiennej `out` w klasie `System`, mamy również Stream `err`, który służy do drukowania błędów. Co jednocześnie wyjaśnia, w jaki sposób wyjątki są drukowane w innym kolorze.

```
System.err.println("some error");
```

To teraz wyobraź sobie jak by wyglądało drukowanie elementów na ekranie, gdyby metody `print()` i `println()` deklarowały wyjątek `IOException`.

in, out oraz err

Czym są zmienne `in`, `out` oraz `err`?

Zmienna `in` reprezentuje standardowe wejście. `Standard Input` jest Streamem, który pozwala nam "komunikować się z aplikacją" i przekazywać jej dane. Najczęściej ten Stream wyłapuje dane wejściowe z klawiatury i przechwytuje w ten sposób wsad od użytkownika. Czyli korzystając z tego Streama możemy wpisywać dane do programu w trakcie jego działania i reagować na wprowadzone w ten sposób dane. Zobaczymy przykład takiego kodu jak przejdziemy do omówienia klasy `Scanner`.

Zmienna `out` reprezentuje standardowe wyjście. `Standard Output` jest Streamem, który pozwala nam drukować dane w terminalu. Jeżeli uruchamiamy aplikację Java w Intellij to tak na prawdę Intellij za nas uruchamia aplikację i pokazuje co jest rezultatem. Możemy natomiast robić to sami w terminalu. Zmienna `out` daje nam możliwość drukowania danych na wyjście standardowe aplikacji - czyli do terminala. W przykładach `PrintStream` będziemy natomiast drukować dane do pliku, a nie na terminal.

Zmienna `err` reprezentuje standardowe wyjścia dla błędów. `Standard Error Output` jest streamem, który służy do drukowania błędów w terminalu. Konwencja mówi, że Stream ten powinien być użyty do wydrukowania informacji, które powinny zostać natychmiast zauważone przez użytkownika aplikacji - czyli np. błędy.

Wspomniane Streamy są otwarte od razu po uruchomieniu aplikacji, nie musimy robić tego ręcznie.

Należy również dodać, że jeżeli będziemy starali się drukować zaraz po sobie `System.out.println()` i `System.err.println()` to nie mamy gwarancji w jakiej kolejności zostaną one wydrukowane. Są to dwa oddzielne Streamy, które drukują informacje na ekranie. Można sobie to wyobrazić w ten sposób, że żaden nie komunikuje się z drugim kiedy wydrukować coś na ekranie żeby zgadzała się kolejność, tylko drukują kiedy im wygodnie 😀.

Możliwe jest ustawienie wspomnianych Streamów `in`, `out` tak żeby operowały na plikach, a nie w terminalu. Domyślnie natomiast, bez żadnych wprowadzonych przez nas zmian operują one na terminalu.

Metody warte uwagi

Zarówno `PrintStream` i `PrintWriter` ułatwiają drukowanie danych poprzez poniższe metody.

print()

Najbardziej podstawowa metoda drukująca. Zwróć uwagę, że gdy wykorzystywaliśmy `FileReader` lub `FileWriter` to mogliśmy do metod zapisujących przekazać `String` albo tablicę `char[]`. Metoda `print()` ma wiele przeładowanych swoich wersji, dzięki czemu możemy do niej przekazać dowolny obiekt lub prymityw. Pod spodem zostanie wywołana wtedy metoda `String.valueOf()` i metoda `write()` tak jak

widzieliśmy to wcześniej.

println()

Metoda `println()` robi to samo co poprzednia, tylko, że po swoim wywołaniu dodaje przejście do nowej linijki. Drugim sposobem na dodanie nowej linijki w `Stringu` jest użycie `\n`. Należy jednak pamiętać o tym, że różne systemy operacyjne dokonują przejścia do nowej linijki na różne sposoby, dlatego lepiej jest używać metody `println()`, bo za nas zostaną rozwiązanie sytuacje skrajne, które mogą doprowadzić do niespodziewanych błędów.

printf() i format()

Pamiętasz metodę `String.format()`? Te dwie metody (`printf()` i `format()`) robią to samo, tylko, że od razu drukując na ekranie.

Przejdźmy do kodu:

```
public class PrintStreamWriterExamples {

    public static void main(String[] args) throws IOException {
        File file = new File("example.txt");
        try (PrintWriter writer = new PrintWriter(new BufferedWriter(new FileWriter(file)))) {
            writer.print(1L);
            writer.write(String.valueOf(1L));

            Car car = new Car("Roman");
            writer.print(car);
            writer.write(car == null ? "null" : car.toString());

            writer.println(); ①
            writer.println("some stuff"); ②
            writer.printf("some value: [%s]", 5); ③
        }
    }
}
```

① wydrukuje tylko nową linijkę

② wydrukuje "some stuff" i nową linijkę

③ przy tej metodzie trzeba uważać, bo samoistnie nie przechodzi ona do nowej linijki

Podsumowanie

Chyba właśnie omówione zostały wszystkie bardziej przydatne klasy typu `Stream`, `Reader` oraz `Writer`, które zostały wpisane w tabelce w notatkach ☺. Oprócz poruszonych klas istnieją jeszcze inne, ale nie poruszam ich tutaj, żeby nie było tego za dużo. I tak w praktyce się okaże, że jak będą one potrzebne to będziesz tego szukać w internecie, najprawdopodobniej ☺.