Obsługa plików

Krzysztof Gębicz

Co to jest JSON?

- JSON (JavaScript Object Notation) to tekstowy format zapisu danych, który pozwala w prosty sposób przechowywać i wymieniać informacje między różnymi aplikacjami i językami programowania.
- Został stworzony na podstawie składni JavaScript, ale jest niezależny od języka – używany praktycznie wszędzie: w Javie, Pythonie, C#, PHP, C++ itd.

Przykład

```
{
   "imie": "Jan",
   "wiek": 17,
   "klasa": "2B",
   "oceny": [5, 4, 3, 5],
   "czy_zdal": true
}
```

Budowa

- JSON składa się z par klucz-wartość.
- Każdy klucz musi być w cudzysłowie " " i jest oddzielony dwukropkiem : od wartości.
- Elementy oddziela się przecinkami.

Typ danych	Przykład	Opis
Tekst	"Jan"	Ujęty w cudzysłowie
Liczba	25	Bez cudzysłowów
Wartość logiczna	true / false	Prawda/fałsz
Tablica	["Java", "Matematyka"]	Lista wartości
Obiekt	{ "miasto": "Kraków" }	Zagnieżdżone dane
null	null	Brak wartości

JSON w Javie — jak go obsługiwać?

Język Java nie ma wbudowanej obsługi JSON, dlatego używamy zewnętrznych bibliotek.

Biblioteka	Producent	Zalety	Wady	Główne klasy
Gson	Google	- Prosta w użyciu- Lekka i szybka- Dobra dla początkujących	- Mniej elastyczna przy bardzo złożonych strukturach	`Gson`, `JsonObject`, `JsonArray`
Jackson	FasterXML	- Bardzo wydajna- Wiele funkcji- Obsługuje JSON, YAML, XML	- Więcej konfiguracji, większy kod	`ObjectMapper`, `JsonNode`
org.json	JSON.org	- Prosta, intuicyjna- Bez potrzeby tworzenia klas	- Mniej wygodna przy dużych danych	`JSONObject`, `JSONArray`

Na co uważać przy JSON w Javie

- Klucze w JSON muszą odpowiadać nazwom pól w klasie
- Nie można mieć przecinka na końcu ostatniego elementu
- Typy danych muszą się zgadzać (np. `int` vs `"tekst"`)
- Plik JSON musi mieć poprawną strukturę (zawsze `{` i `}` zamknięte)
- Zawsze zamykaj pliki po odczycie/zapisie, inaczej dane mogą się nie zapisać

Zalety używania JSON w Javie

- Bardzo łatwe mapowanie danych z/do obiektów
- Możliwość pracy z plikami konfiguracyjnymi
- Idealny format do zapisu danych użytkownika
- Współpraca z API
- Można łatwo przetwarzać dane z aplikacji mobilnych, serwerów i stron WWW

Główne cechy pliku TXT

- Przechowuje czysty tekst (litery, liczby, znaki specjalne).
- Każda linia może zawierać inny wpis.
- Nie zawiera informacji o strukturze danych (tak jak JSON czy XML).
- Idealny do prostych danych lub logów.

Klasa	Pakiet	Zastosowanie
`File`	`java.io`	Sprawdzenie, czy plik istnieje, utworzenie, usunięcie
`FileReader`	`java.io`	Odczyt znak po znaku
`FileWriter`	`java.io`	Zapis tekstu do pliku
`BufferedReader`	`java.io`	Wydajny odczyt linia po linii
`BufferedWriter`	`java.io`	Wydajny zapis tekstu linia po linii
`Files`	`java.nio.file`	Nowoczesne, uproszczone metody odczytu i zapisu
`Scanner`	`java.util`	Prosty odczyt danych z pliku

Importowanie potrzebnych bibliotek

- import java.io.File;
- import java.io.FileReader;
- import java.io.FileWriter;
- import java.io.BufferedReader;
- import java.io.BufferedWriter;
- import java.io.IOException;
- import java.util.Scanner;
- import java.nio.file.Files;
- import java.nio.file.Paths;
- import java.util.List;

Sposoby odczytu pliku tekstowego

a) `FileReader`

Odczytuje znak po znaku, np. litery jednego po drugim.

b) `BufferedReader`

Odczytuje całe linie tekstu — szybciej i wygodniej:

```
BufferedReader reader = new BufferedReader(new
FileReader("dane.txt"));

String linia = reader.readLine();
```

d) `Files.readAllLines()`

Nowoczesne podejście z pakietu `java.nio.file`:

```
List<String> linie =
Files.readAllLines(Paths.get("dane.txt"));
```

Sposoby zapisu do pliku

a) `FileWriter`

Zapisuje tekst znak po znaku lub całe linie:

```
FileWriter writer = new FileWriter("wynik.txt");
writer.write("Pierwsza linia\nDruga linia");
writer.close();
```

b) `BufferedWriter`

Wydajniejsza metoda (buforuje dane):

```
BufferedWriter writer = new BufferedWriter(new
FileWriter("wynik.txt"));

writer.write("To jest linia 1");

writer.newLine();

writer.write("To jest linia 2");

writer.close();
```

c) Files.write()

Nowoczesne i krótkie:

```
Files.write(Paths.get("wynik.txt"),
"Zawartość pliku".getBytes());
```

Zalety	Wady
Łatwe w odczycie i edycji	Brak struktury danych
Mały rozmiar	Trudne do przetwarzania przy dużych plikach
Nie wymaga specjalnych bibliotek	Brak typów danych (wszystko tekst)
Działa na każdym systemie	Potrzebna ostrożność z kodowaniem znaków

Na co uważać

- Zawsze zamykaj pliki po użyciu (`close()`)
- Sprawdzaj, czy plik istnieje, zanim go odczytasz
- Obsługuj wyjątki (`IOException`)
- Pamiętaj o kodowaniu znaków (UTF-8 vs Windows-1250)
- Przy zapisie używaj `\n` do przejścia do nowej linii

Dobre praktyki

- Używaj `try-with-resources` — automatycznie zamyka plik
- Unikaj ręcznego `close()` jeśli używasz `try-with-resources`
- Buforuj dane (`BufferedReader`, `BufferedWriter`)
- Używaj `Paths` i `Files` w nowych projektach
- W logach zawsze dodawaj datę/godzinę zapisu