# Lekcja: Praca z plikami i szyfrowanie

## FileReader

FileReader to klasa w języku Java używana do odczytywania danych z plików jako strumień znaków. Działa bardzo podobnie do FileInputStream, z tą różnicą, że FileInputStream czyta bajty, a FileReader znaki. FileReader jest przeznaczony do odczytywania strumieni znaków z plików, co czyni go idealnym wyborem do odczytywania plików tekstowych.

**Główne cechy:**

Prosty w użyciu przy odczycie plików tekstowych.

Odczytuje dane znak po znaku.

Można go łatwo połączyć z BufferedReader dla efektywniejszego odczytu.

## BufferedReader

BufferedReader to klasa w języku Java, która zapewnia buforowanie znaków podczas odczytu, co znacząco zwiększa wydajność przez zmniejszenie liczby operacji wejścia/wyjścia. Owiń FileReader (lub inny Reader) obiektem BufferedReader, aby zapewnić buforowanie danych wejściowych i efektywnie czytać większe bloki danych naraz.

Główne cechy:

Zwiększa wydajność odczytu przez buforowanie znaków.

Umożliwia odczyt linii tekstu za pomocą metody readLine().

Zmniejsza koszty związane z dostępem do dysku poprzez odczytywanie większych bloków danych.

Przykład użycia:

Poniżej znajdziesz przykład, który demonstruje, jak używać FileReader i BufferedReader do odczytania pliku tekstowego w Java:

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

public class FileReadExample {

public static void main(String[] args) {

// Ścieżka do pliku, który chcesz odczytać

String filePath = "path/to/your/file.txt";

try {

// Utworzenie FileReader do odczytu pliku

FileReader fileReader = new FileReader(filePath);

// Owinięcie FileReader w BufferedReader.

BufferedReader bufferedReader = new BufferedReader(fileReader);

String line;

// Odczytywanie pliku linia po linii

while ((line = bufferedReader.readLine()) != null) {

System.out.println(line); // Wyświetlenie odczytanej linii

}

// Zamknięcie BufferedReader i FileReader

bufferedReader.close(); // Zamyka również fileReader

} catch (IOException e) {

e.printStackTrace();

}

}

}

W powyższym kodzie FileReader jest używany do otwarcia pliku do odczytu, a BufferedReader jest używany do buforowania odczytanych znaków. Metoda readLine() klasy BufferedReader czyta każdą linię tekstu z pliku aż do osiągnięcia końca pliku. Po zakończeniu odczytu pliku, zarówno BufferedReader jak i FileReader są zamykane, aby zwolnić zasoby systemowe.

Zmodyfikuj ścieżkę filePath na odpowiednią dla twojego systemu i pliku, który chcesz odczytać. Ta metoda jest wydajna i szeroko stosowana do odczytywania plików tekstowych w aplikacjach Java.

## Try-catch

try-catch w Javie to mechanizm obsługi wyjątków, który pozwala na przechwycenie i obsłużenie problemów, które mogą pojawić się podczas wykonania programu. Wyjątki to zdarzenia (zazwyczaj błędy), które zakłócają normalny przepływ instrukcji w programie. Oto jak działa try-catch:

## Try

**Blok try:** Tutaj umieszczasz kod, który może wygenerować wyjątek. Java wykonuje instrukcje wewnątrz bloku try aż do momentu napotkania wyjątku. Jeśli wszystko przebiegnie pomyślnie, i żaden wyjątek nie zostanie wyrzucony, bloki catch są pomijane, a wykonanie programu kontynuowane jest poza strukturą try-catch.

Wyrzucanie wyjątków: Gdy w bloku try pojawia się problem, Java "wyrzuca" wyjątek. Wyrzucenie wyjątku natychmiast przerywa wykonanie bieżącej metody i przekazuje kontrolę do pierwszego pasującego bloku catch.

## Catch

**Blok catch:** Jest to miejsce, gdzie definiujesz, jak program powinien reagować na konkretny typ wyjątku. Blok catch zawiera parametr, który odpowiada typowi wyjątku, który chcesz obsłużyć. Możesz mieć wiele bloków catch dla różnych typów wyjątków.

Obsługa wyjątku: W bloku catch, możesz zdecydować, co zrobić z wyjątkiem — możesz wydrukować komunikat o błędzie, spróbować naprawić problem, zarejestrować wyjątek w logu, ponownie wyrzucić wyjątek lub podjąć inne działania.

Wielokrotne bloki catch: Możesz mieć wiele bloków catch po bloku try, każdy z nich przeznaczony do obsługi innego rodzaju wyjątku. Java sprawdzi je po kolei i wejdzie do pierwszego bloku catch, którego typ pasuje do wyjątku.

## Finally (opcjonalnie)

**Blok finally:** Jest to opcjonalny blok, który zawsze jest wykonywany po bloku try/catch, niezależnie od tego, czy wyjątek został wyrzucony, czy nie. Jest to dobre miejsce do umieszczania kodu, który musi być wykonany bez względu na to, co się stanie (na przykład zamknięcie plików lub zwolnienie zasobów systemowych).

**Przykład:**

try {

// Kod, który może wygenerować wyjątek

int division = 10 / 0; // To spowoduje ArithmeticException

} catch (ArithmeticException e) {

// Kod do wykonania, gdy wystąpi ArithmeticException

System.out.println("Nie można dzielić przez zero!");

} finally {

// Kod do wykonania zawsze po bloku try/catch

System.out.println("Blok finally - wykonuję się zawsze.");

}

**Podsumowanie:**

**Obsługa wyjątków:** try-catch pozwala na eleganckie i kontrolowane obsługiwanie błędów i innych nieoczekiwanych sytuacji.

**Kontrola przepływu:** Pomaga utrzymać płynność działania programu, nawet gdy coś pójdzie nie tak.

**Zasoby i czyszczenie:** Używanie finally lub try-with-resources zapewnia, że zasoby są odpowiednio zwalniane.

Pamiętaj, że nadmierne i nieodpowiednie używanie obsługi wyjątków może prowadzić do kodu trudnego do zrozumienia i utrzymania, dlatego zawsze dąż do pisania klarownego i przemyślanego kodu.

## Zadanie 1

Wczytujemy dane z pliku CSV (data.csv) zawierający dane studentów: imię, nazwisko i zaszyfrowane hasło.

Plik wygląda mniej więcej tak:

Jan,Kowalski,Qiwjs.

Każdy wiersz reprezentuje jednego studenta.

Następnie parsujemy dane i zamieniamy na listę obiektów klasy Student.

Następnie dodajemy do klasy student metodę, która pozwoli na rozszyfrowanie szyfrogramów.

Odszyfrowanie hasła: Hasło zostało zaszyfrowane przy użyciu szyfru Cezara, co oznacza, że każda litera w haśle została przesunięta o pewną liczbę miejsc w alfabecie.

W naszym wypadku przesuniecie wynosi 5.

**Elementy języka Java, które będą potrzebne:**

Odczyt plików: FileReader, BufferedReader.

Operacje na stringach: Metody klasy String, takie jak charAt(), substring(), length().

Pętle: Użycie pętli for lub while do iteracji przez znaki w ciągu.

Instrukcje warunkowe: if lub switch do obsługi różnych przypadków alfabetu (np. przesunięcie ponad 'z').

Konwersje typów: Rzutowanie i konwersje między typami, szczególnie przy operacjach na charach i intach.

Obsługa wyjątków: Try-catch do obsługi błędów przy wczytywaniu plików.