Strumienie

Hierarchie klas strumieniowych, strumienie bajtowe - klasy InputStream i OutputStream i ich metody, klasa File, strumienie plikowe, strumienie buforowane, strumienie danych, strumienie znakowe, klasy strumieni znakowych, metody klas Reader i Writer.

FUW

Strumienie

- **Do transmisji danych zwykle wykorzystuje się strumienie.**
- **Strumień jest pojęciem abstrakcyjnym. Oznacza tor komunikacyjny pomiędzy dowolnym źródłem danych a ich miejscem przeznaczenia.**
- **Istnieją dwa rodzaje strumieni:**
  - **wejściowe** - do odczytu danych ze źródła
  - **wyjściowe** - do zapisu danych w ich miejscu docelowym.
- **Ponieważ operacje wejścia-wyjścia można przeprowadzać na danych binarnych lub na znakach, więc strumienie dzielą się także na strumienie bajtowe i strumienie znakowe.**
- **Wszystkie klasy strumieniowe, składające się na system wejścia-wyjścia w Javie, zgrupowane są w pakiecie java.io.**
- **Tworzą one cztery hierarchie klas, wywodzące się z następujących, abstrakcyjnych klas bazowych:**

	bajty	znaki Unicode
Wejście	InputStream	Reader
Wyjście	OutputStream	Writer

Klasy strumieni bajtowych

Klasa	Opis
BufferedInputStream **BufferedOutputStream**	Strumienie buforowane
ByteArrayInputStream **ByteArrayOutputStream**	Klasy przeznaczone do transmisji tablicy bajtów
DataInputStream **DataOutputStream**	Strumienie danych, pozwalające na zapis i odczyt różnych ty═ów danych.
FileInputStream **FileOutputStream**	Strumienie plikowe
FilterInputStream **FilterOutputStream**	Klasy bazowe dla klas umożliwiających filtrowanie strumienia danych
PipedInputStream **PipedOutputStream**	Klasy pozwalające na łączenie strumieni w potoki (pipes)
PrintStream	Drukujący strumień wyjściowy
PushbackInputStream	Klasa pozwalająca cofać odczytane bajty
RandomAccessFile	Obsługuje pliki o dostępie swobodnym
SequenceInputStream	Łączenie weilu strumieni wejściowych

Metody klasy InputStream

int available( )	Podaje liczbę bajtów dostępnych do odczytu
void close( )	Zamyka strumień i odłącza go od źródła
boolean mark(int ile)	Umieszcza znacznik na aktualnej pozycji w strumieniu do momentu odczytania podanej liczby bajtów
boolean markSupported( )	Sprawdza, czy strumień obsługuje metody mark() i reset()
int read( )	Zwraca liczbę całkowitą z przedziału 0-255, która odpowiada odczytanemu bajtowi. Po napotkaniu końca strumienia zwraca -1
int read(byte bufor[ ])	Czyta bajty i umieszcza je w buforze. Zwraca liczbę odczytanych bajtów lub -1
int read(byte bufor[ ], int poz, int ile)	Czyta do bufora określoną liczbę bajtów poczynając od podanej pozycji. Zwraca liczbę odczytanych bajtów lub -1
void reset( )	Resetuje pozycję wskaźnika wejściowego
long skip(long ile)	Pomija podaną liczbę bajtów z wejścia. Zwraca liczbę pominiętych bajtów

Metody klasy OutputStream

void close( )	Zamyka wykorzystywany strumień danych. Jeśli po zamknięciu i odłączeniu strumienia nastąpi próba zapisu danych, zostanie wygenerowany wyjątek IOException
void flush( )	Czyści strumień wyjściowy
void write(int bajt)	Zapisuje pojedynczy bajt do strumienia wyjściowego. Argument jest liczbą całkowitą z przedziału od 0 do 255. W przypadku podania większej liczby, przekazany zostanie jej młodszy bajt
void write(byte bufor[ ])	Zapisuje tablicę bajtów do strumienia wyjściowego
void write(byte bufor[ ] , int poz, int ile)	Zapisuje do tablicy bufor określoną przez parametr ile liczbę bajtów, poczynając od podanej pozycji

Klasa File

Pakiet java.io zawiera klasę File, która reprezentuje odwołania do plików lub katalogów. Posiada ona trzy konstruktory:

- **File(String nazwa)**

odwołanie do pliku lub katalogu o podanej nazwie, może dotyczyć plików i katalogów istniejących i tych, które mają być utworzone

- **File(String katalog, String plik)**
- **File(File katalog, String pliku)**

parametry reprezentują katalog i zawarty w nim plik

Operacje plikowe wykonuje się za pomocą metod klasy File. Niektóre z nich muszą być umieszczone w bloku try...catch. Na przykład tworzenie nowego pliku może spowodować wyjątek IOException:

```
File f = new File("C:\\a.txt");
try {
   f.createNewFile();
} catch (IOException e) {
   System.err.println("Błąd + e");
}
```

FUW

Metody klasy File

boolean exists( )	Sprawdza, czy plik istnieje na dysku
boolean createNewFile( )	Tworzy nowy plik.
boolean delete( )	Usuwa istniejący plik. Zwraca wartość logiczną, która informuje czy operacja się udała
void deleteOnExit( )	Usuwa plik podczas zakończenia programu
String getAbsolutePath()	Zwraca ścieżkę dostępu
String getName( )	Zwraca nazwę pliku lub katalogu
boolean isDirectory( )	Sprawdza, czy obiekt File jest katalogiem
boolean isFile( )	Sprawdza, czy obiekt File jest plikiem
long length( )	Zwraca rozmiar pliku wyrażony w bajtach
boolean mkdir( )	Tworzy katalog
boolean renameTo(File)	Tworzy duplikat istniejącego pliku lub katalogu; nazwa jest przekazywana przez argument File

Strumienie plikowe

Strumienie te są najczęściej używanym rodzajem strumieni bajtowych. Podstawowe ich klasy to FileOutputStream i FileInputStream.

- **Wyjściowe strumienie plikowe tworzymy za pomocą konstruktora FileOutputStream(String adres_pliku)**
- **Na przykład strumień wyjściowy do zapisu utworzymy pisząc:**

```
FileOutputStream f = new FileOutputStream("a.dat");
```

Jeśli plik "a.dat" nie istnieje, to zostanie utworzony. W przypadku istniejącego już pliku, jego zawartość zostanie usunięta.

- **Czasami chcemy dopisać bajty na końcu istniejącego już pliku. Używamy wtedy konstruktora dwuargumentowego:**

```
FileOutputStream f = new FileOutputStream("a.dat",true);
```

- **Po utworzeniu strumienia wyjściowego umieszcza się w nim dane:**

```
f.write(x); //x - liczba całkowita od 0 do 255
```

- **Zaraz po zapisaniu ostatniego bajtu danych zamykamy strumień:**

```
f.close();
```

FUW

Strumienie plikowe

Odczytywanie bajtów danych ze strumienia przebiega podobnie:

I. **Zadeklarowanie strumienia i dowiązanie go do pliku:**

```
FileInputStream f = new FileInputStream("a.dat");
```

II. **Pobranie danych za pomocą metody read():**

```
while (true)
{
   int x = f.read();
   if (x==-1) break;
   System.out.print(x + " ");
}
```

- **Powyższa pętla odczytuje kolejne bajty danych ze strumienia reprezentowanego przez obiekt f aż do napotkania znaku końca pliku, czyli wartości -1.**

III. **Zamknięcie strumienia i odłączenie go od pliku:**

```
f.close();
```

Przykład - FileOutputStream

Zapisywanie bajtów danych w strumieniu.

```
import java.io.*;
public class WriteStream {
  public static void main(String[] args) {
    int[] v = {1,0,127,4,9,32,25,36,255,16};
    try {
     FileOutputStream f = new
FileOutputStream("C:\\a.dat");
     for (int i=0; i<v.length; i++) {
        f.write(v[i]);
     }
     f.close();
    }
    catch (IOException e) {
     System.err.println("Błąd " + e);
    }
  }
}
```

Dana jest tablica zawierająca liczby całkowite z zakresu od 0 do 255. Program zapisuje elementy tablicy do pliku binarnego.

Przykład - FileInputStream

Program odczytuje bajty danych ze strumienia.

```
import java.io.*;
public class ReadStream {
  public static void main(String[] args) {
    try {
      FileInputStream f = new FileInputStream("C:\\a.dat");

          while (true) {
            int x = f.read();
            if (x==-1) break;
            else System.out.print(x + " ");
          }
      f.close();
    }
    catch (IOException e) {
      System.err.println("Błąd " + e);
    }
  }
}
```

Strumienie buforowane

Buforowanie odczytu i zapisu danych znacznie przyspiesza działanie programu, ponieważ umożliwia transmisję większych porcji danych. Buforowanie operacji zapisu i odczytu danych odbywa się w sposób automatyczny i kod programu zmienia się nieznacznie.

I. **Tworzymy strumień plikowy tak jak poprzednio:**

```
FileOutputStream f1 = new FileOutputStream("a.dat");
FileInputStream f2 = new FileInputStream("a.dat");
```

II. **Deklarujemy bufor i łączymy go ze strumieniem:**

```
BufferedOutputStream b1 = new BufferedOutputStream(f1);
BufferedInputStream b2 = new BufferedInputStream(f2);
```

III. **Od tej pory wszystkie operacje wykonujemy na buforze:**

```
b1.write(127);
b1.close();
```

```
int x = b2.read();
b2.close();
```

Buforowany zapis w strumieniu

```
import java.io.*;
public class WriteStream {
  public static void main(String[] args) {
    int[] v = {0,1,4,9,16,25,36,127,254,255};
    try {
     FileOutputStream f = new
FileOutputStream("C:\\a.dat");
     BufferedOutputStream bf = new BufferedOutputStream(f);
     for (int i=0; i<v.length; i++) {
        bf.write(v[i]);
     }
     bf.close();
    }
    catch (IOException e) {
     System.err.println("Błąd " + e);
    }
  }
}
```

Buforowany odczyt ze strumienia

```
import java.io.*;
public class ReadStream
{
  public static void main(String[] args) {
     try {
     FileInputStream f = new FileInputStream("C:\\a.dat");
     BufferedInputStream bf = new BufferedInputStream(f);
     while (true) {
         int x= bf.read();
         if (x==-1) break;
         else System.out.print(x + " ");
     }
     bf.close();
    }
    catch (IOException e) {
     System.err.println("Błąd " + e);
    }
  }
}
```

Strumienie danych

- **Strumienie danych umożliwiają posługiwanie się różnymi typami danych. Filtrują one połączone z nimi strumienie bajtów, pozwalając na zapis i odczyt prostych typów danych.**
- **Podstawowe klasy to: DataOutputStream i DataInputStream.**
- **Tworzenie wyjściowego strumienia danych przebiega według schematu:**

I. **Deklarujemy strumień plikowy i łączymy go z plikiem**

```
FileOutputStream f = new FileOutputStream("a.dat");
```

II. **Deklarujemy bufor i łączymy go ze strumieniem:**

```
BufferedOutputStream bf = new BufferedOutputStream(f);
```

III. **Deklarujemy strumień danych i łączymy go z buforem:**

```
DataOutputStream df = new DataOutputStream(bf);
```

IV. **Zapisujemy dane do strumienia danych:**

```
df.writeDouble( 12.5 );
```

FUW

Metody strumieni danych

Konstruktor i metody klasy **DataOutputStream**	Konstruktor i metody klasy **DataInputStream**
DataOutputStream(OutputStream f)	DataInputStream(InputStream f)
void writeBoolean(boolean b)	boolean readBoolean( )
void writeByte(int n)	byte readByte( )
void writeChar(int n)	char readChar( )
void writeDouble(double x)	double readDouble( )
void writeFloat(float x)	float readFloat( )
void writeInt(int x)	int readInt( )
void writeLong(long x)	long readLong( )
void writeShort(int x)	short readShort( )

Przykład - DataOutputStream

Zapis prostych typów danych w strumieniu.

```
import java.io.*;
public class WriteStream {
  public static void main(String[] args) {
    double[] v = {0.0, 1.2, 9.1, 2.5, 3.6, 12.4, 5.5};
    try {
     FileOutputStream f = new
FileOutputStream("C:\\a.dat");
     BufferedOutputStream bf = new BufferedOutputStream(f);
     DataOutputStream df = new DataOutputStream(bf);
     for (int i=0; i<v.length; i++) {
        df.writeDouble(v[i]);
     }
     df.close();
    } catch (IOException e) {
     System.err.println("Błąd " + e);
    }
  }
}
```

Dana jest tablica zawierająca liczby rzeczywiste typu double. Program zapisuje elementy tablicy do pliku binarnego.

Przykład - DataInputStream

Odczytywanie wartości typu double ze strumienia

```
import java.io.*;
public class ReadStream {
 public static void main(String[] args) {
   try {
     FileInputStream f = new FileInputStream("C:\\a.dat");
     BufferedInputStream bf = new BufferedInputStream(f);
     DataInputStream df = new DataInputStream(bf);
     try {
       while (true) {
       double x = df.readDouble();
       System.out.print(x + " ");
       }
     } catch (EOFException e) {
           df.close();}
   } catch (IOException e) {
     System.err.println("Błąd " + e);}
 }
}
```

Metoda **readDouble()** nie zwraca żadnej wartości, która wskazywałaby na osiągnięcie końca pliku. Wykorzystujemy więc fakt, że po napotkaniu końca pliku wystąpi wyjątek **IOException**.

Strumienie znakowe

- Reader i Writer - to klasy bazowe strumieni znakowych.
- Służą one do przesyłania znaków Unicode o 16-bitowych kodach.
- Wszystkie znakowe operacje wejścia-wyjścia powinny być wykonywane na strumieniach znakowych, które poprawnie obsługują znaki Unikodu.
- Przykład - strumienie FileWriter i BufferWriter:

I. Deklarujemy strumień plikowy i łączymy go z plikiem

```
FileWriter f = new FileWriter("a.dat");
```

II. Deklarujemy bufor i łączymy go ze strumieniem:

```
BufferedWriter bf = new BufferedWriter(f);
```

III. Umieszczamy dane w strumieniu buforowanym:

```
bf.write('A');
```

IV. Zamykamy strumień buforowany

```
bf.close();
```

FUW

Klasy strumieni znakowych

Klasa	Opis
BufferedReader **BufferedWriter**	Buforowane strumienie znakowe
CharArrayReader **CharArrayWriter**	Klasy przeznaczone do transmisji tablicy znaków
FileReader **FileWriter**	Strumienie plikowe
FilterReader **FilterWriter**	Strumienie filtrujące
InputStreamReader **OutputStreamWriter**	Konwersja odczytywanych bajtów na znaki, konwersja zapisywanych znaków na bajty
PipedReader **PipedWriter**	Potoki łączące wyjście z jednego strumienia z wejściem dla innego strumienia
PrintWriter	Klasa zawierająca metody print() i println()
PushbackReader	Klasa pozwalająca cofać odczytane bajty
StringReader **StringWriter**	Klasy pozwalajace na operacje wejścia i wyjścia na łańcuchach znaków.

Metody klasy Reader

abstract void close()	Zamyka strumień wejściowy
void mark(int ile)	Umieszcza znacznik na aktualnej pozycji w strumieniu do momentu odczytania podanej liczby znaków
boolean markSupported()	Sprawdza, czy strumień obsługuje metody mark() i reset()
int read()	Zwraca liczbę całkowitą z przedziału 0-255, która reprezentuje odczytany znak. Po napotkaniu końca strumienia zwraca -1
int read(char bufor[ ])	Czyta znaki i umieszcza je w tablicy bufor. Zwraca liczbę odczytanych znaków lub -1
int read(char bufor[ ], int poz, int ile)	Czyta do bufora określoną liczbę znaków, poczynając od podanej pozycji. Zwraca liczbę odczytanych znaków lub -1
boolean ready( )	Sprawdza gotowość do odczytu
void reset( )	Resetuje pozycję wskaźnika wejściowego
long skip (long ile)	Pomija podaną liczbę znaków z wejścia. Zwraca liczbę pominiętych znaków

Metody klasy Writer

abstract void close( )	Zamyka strumień wyjściowy. Jeśli po zamknięciu nastąpi próba zapisu danych, zostanie wygenerowany wyjątek IOException
abstract void flush( )	Czyści strumień wyjściowy
void write(int znak)	Zapisuje pojedynczy znak do strumienia wyjściowego. Argument jest liczbą całkowitą z przedziału od 0 do 255
void write(char bufor[ ])	Zapisuje tablicę znaków do strumienia wyjściowego
void write(char bufor[ ], int poz, int ile)	Zapisuje do tablicy bufor określoną przez parametr ile liczbę znaków, poczynając od podanej pozycji
void write(String s)	Zapisuje łańcuch znaków do strumienia wyjściowego
void write(String s, int poz, int dlugosc)	Zapisuje łańcuch znaków o określonej długości, poczynając od podanej pozycji

Przykład 1

Na wejściu dany jest plik tekstowy a.txt. Ułóż program, który przepisze ten plik na nowy plik o nazwie b.txt, poprzedzając kolejne wiersze ich numerami. Każdy numer powinien być oddzielony spacją od początku linii.

Rozwiązanie

Klasa **BufferedReader** dostarcza metodę readLine(), która zwraca odczytaną ze strumienia linię tekstu.

```
import java.io.*;
public class NowyPlik
{
   public static void main(String args[])
   {
      try {
         FileReader fr = new FileReader("a.txt");
         BufferedReader br = new BufferedReader(fr);
         FileWriter fw = new FileWriter("b.txt");
         BufferedWriter bw = new BufferedWriter(fw);
```

Przykład 1 - c.d.

```
    int i=0;
    String line;
    while (true)
    {
      line = br.readLine();
      if ( line == null ) break;
      i++;
      bw.write(i + " " + line + "\r\n");
    }
    br.close();
    bw.close();
  }
  catch(IOException e) {
    System.err.println("Błąd " + e);
  }
}
```

W przypadku napotkania znaku końca pliku metoda readLine() zwraca **null**, co wykorzystujemy do przerwania pętli.

Przykład 2

W pliku tekstowym a.txt umieszczony jest n-elementowy wektor. Sposób zapisu jest następujący:

- **w pierwszym wierszu umieszczona jest litera 'n';**
- **w drugim wierszu zapisana jest liczba całkowita -należy ją interpretować jako wartość parametru n.**
- **ostatni wiersz zawiera n liczb całkowitych, oddzielonych spacjami. Należy go interpretować jako wektor.**

Napisz aplikację, która obliczy sumę elementów tego wektora.

Rozwiązanie

```
import java.io.*;
public class Suma {
   public static void main(String args[]) {
      try {
         FileReader f = new FileReader("a.txt");
         BufferedReader b = new BufferedReader(f);
         StreamTokenizer in = new StreamTokenizer(b);
```

Przykład 2 - c.d.

```
    String linia = b.readLine();
    in.nextToken();
    int n = (int) in.nval;
    int[] v = new int[n];
    for (int i=0; i<n; i++)
    {
      in.nextToken();
      v[i] = (int) in.nval;
    }
    int suma = 0;
    for (int i=0; i<n; i++)
    {
    }
    b.close();
    System.out.println("Suma = " + suma);
  } catch (IOException e) {
      System.err.println("Błąd"+e);
  }
 }
}
```

- W rozwiązaniu korzystamy z klasy **StreamTokenizer**. Dzieli ona strumień wejściowy na tokeny, czyli jednostki leksykalne, takie jak słowa, liczby, znaki itp.
- Do pobrania kolejnego leksemu służy metoda **nextToken**.
- Pole **nval** klasy StreamTokenizer zapamiętuje wartości typu double.

Przykład 3

Aplikacja odczytuje kolejne bajty danego pliku binarnego i zapisuje je w nowym pliku. Można spowodować, że niektóre bajty zostaną zmienione.

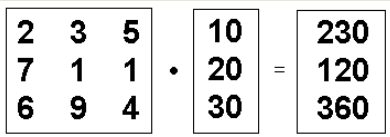
```
import java.io.*;

public class NowyPlik
{
   public static void main(String args[])
   {
      try
      {
         FileInputStream fi = new
                              FileInputStream("a.gif");
         BufferedInputStream bi = new
                              BufferedInputStream(fi);
         FileOutputStream fo = new
                              FileOutputStream("b.gif");
         BufferedOutputStream bo = new
                              BufferedOutputStream(fo);
```

Przykład 3-c.d.

```
      int x;
      boolean eof = false;
      int i=0;
      while (!eof)
      {
         x = bi.read();
         if (x==127 && i>200) x=x/8;
         bo.write(x);
         if (x==-1) eof=true;
         else i++;
      }
      bi.close();
      bo.close();
      System.out.println("\nZapisano bajtów: " + i);
    }
    catch (IOException e)
    {
       System.err.println("Błąd " + e);
    }
  }
}
```

Zadanie



W pliku tekstowym a.txt zapisanym na dysku C umieszczona jest macierz nxn oraz n-elementowy wektor. Sposób zapisu jest następujący:

- W pierwszym wierszu umieszczona jest litera n.
- W drugim wierszu zapisana jest liczba, należy ją interpretować jako wartość parametru n. Liczba ta jest całkowita i dodatnia.
- Następny wiersz zawiera słówko: "macierz".
- Każdy z następnych n wierszy składa się z n liczb całkowitych oddzielonych spacjami bądź tabulatorami. Każdy z tych wierszy należy interpretować jako wiersz macierzy.
- Następny wiersz zawiera słówko: "wektor".
- Kolejny, ostatni wiersz zawiera n liczb całkowitych, oddzielonych spacjami. Należy go interpretować jako wektor.

Zakładamy, że w pliku nie występują błędy, zapisane liczby mieszczą się w dostępnej arytmetyce. Napisać program, który mnoży macierz przez wektor zapisane w pliku a.txt. Następnie wynik umieszcza w tym samym pliku, dopisując go na końcu, w osobnej linijce, która powinna być poprzedzona słówkiem: "wynik". Liczby należy oddzielić spacjami.