Java
Obsługa strumieni

Strumień

Według Słownika Encyklopedycznego - Informatyka, Wyd. Europa (1999):

> Strumień (angielskie stream), połączenie między nadawcą a odbiorcą w postaci nieprzerwanej sekwencji danych, nie dzielonych na komunikaty. Komunikacja strumieniowa jest realizowana za pomocą buforów pozwalających nadawcy wyprzedzać odbiorcę.
>
> Strumienie znajdują zastosowanie przy zdalnych rejestracjach, przesyłaniu plików i są podstawą organizowania komunikacji typu producent-konsument.

Strumienie standardowe

Strumienie standardowe:

- System.in
  - Srumień wejściowy – domyślnie podłączony pod klawiaturę
- System.out
  - Strumień wyjściowy – domyślnie podłączony pod konsolę
- System.err
  - Strumień związany z obsługą błędów – domyślnie podłączony pod konsolę

- Konieczność obsługi wyjątku:
  - IOException – błędna lub przerwana operacja we/wy (np. w sieci zerwanie połączenia, brak miejsca na dysku itp.)

Strumienie standardowe

- Możliwość przedefiniowania domyślnych strumieni:

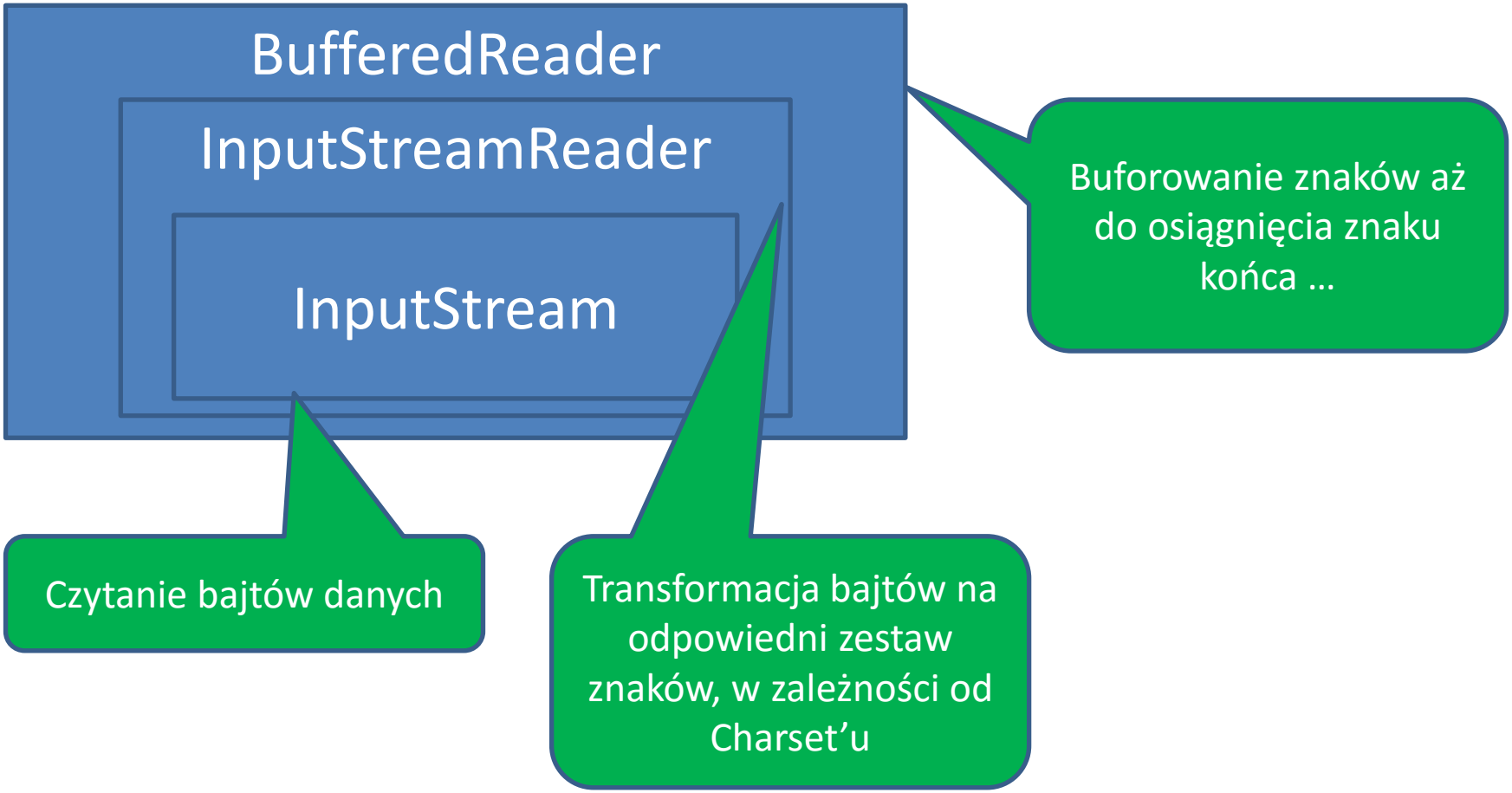
static void	**setErr**(PrintStream err) Reassigns the "standard" error output stream.
static void	**setIn**(InputStream in) Reassigns the "standard" input stream.
static void	**setOut**(PrintStream out) Reassigns the "standard" output stream.

Strumień wejściowy
Klasa **InputStream**

Klasy implementujące **InputStream:**

- AudioInputStream,
- ByteArrayInputStream,
- FileInputStream,
- FilterInputStream,
- ObjectInputStream,
- PipedInputStream,
- SequenceInputStream,
- StringBufferInputStream

Struktura strumieni wejściowych



InputStream

Konstruktor

- **InputStream**()

Metody

- Int **available**()
  Returns the number of bytes that can be read (or skipped over) from this input stream without blocking by the next caller of a method for this input stream.
- Void **close**()
  Closes this input stream and releases any system resources associated with the stream.
- Void **mark**(int readlimit)
  Marks the current position in this input stream.
- Boolean **markSupported**()
  Tests if this input stream supports the mark and reset methods.abstract
- Int **read**()
  Reads the next byte of data from the input stream.
- Int **read**(byte[] b)
  Reads some number of bytes from the input stream and stores them into the buffer array b.
- Int **read**(byte[] b, int off, int len)
  Reads up to len bytes of data from the input stream into an array of bytes.
- Void **reset**()
  Repositions this stream to the position at the time the mark method was last called on this input stream.
- Long **skip**(long n)
  Skips over and discards n bytes of data from this input stream.

InputStreamReader

InputStreamReader jest mostem pomiędzy strumieniem bajtów a strumieniem znaków. InputStreamReader czyta bajty i dekoduje je na odpowiednie znaki używając specjalnego charset (zbioru znaków).

Konstruktory

InputStreamReader(InputStream in)
Create an InputStreamReader that uses the default charset.

InputStreamReader(InputStream in, Charset cs)
Create an InputStreamReader that uses the given charset.

InputStreamReader(InputStream in, CharsetDecoder dec)
Create an InputStreamReader that uses the given charset decoder.

InputStreamReader(InputStream in, String charsetName)
Create an InputStreamReader that uses the named charset.

Metody

Void **close**()
Close the stream.

String **getEncoding**()
Return the name of the character encoding being used by this stream.

Int **read**()
Read a single character.

Int **read**(char[] cbuf, int offset, int length)
Read characters into a portion of an array.

Boolean **ready**()
Tell whether this stream is ready to be read.

BufferedReader

Czyta tekst z wejścia będącego znakowym strumieniem wejściowym, buforuje je pozwalając na wydajne czytanie znaków, tablic i linii.

Konstruktory

- **BufferedReader**(Reader in)
  Create a buffering character-input stream that uses a default-sized input buffer.
- **BufferedReader**(Reader in, int sz)
  Create a buffering character-input stream that uses an input buffer of the specified size.

Metody

- Void **close**()
  Close the stream.
- Void **mark**(int readAheadLimit)
  Mark the present position in the stream.
- Boolean **markSupported**()
  Tell whether this stream supports the mark() operation, which it does.
- Int **read**()
  Read a single character.
- Int **read**(char[] cbuf, int off, int len)
  Read characters into a portion of an array.
- String **readLine**()
  Read a line of text.
- Boolean **ready**()
  Tell whether this stream is ready to be read.
- Void **reset**()
  Reset the stream to the most recent mark.
- Long **skip**(long n)
  Skip characters.

Przykład

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Strumien {

    public static void main(String[] args) {
        InputStreamReader inp = new InputStreamReader(System.in);
        BufferedReader inbr = new BufferedReader(inp);
        try {
            String line = inbr.readLine();
            System.out.println(line);
        } catch (IOException e) {
            System.out.println("Bąd odczytu");
        }
    }
}
```

Problemy

- Musimy obsługiwać wyjątki!
- Rozwiązanie -> klasa StreamTokenizer

StreamTokenizer

- Automatyczne parsowanie strumieni danych (uwaga na specyfikę kodowanie)
- Wartości odczytywane z pól publicznych
  - nval – pole publiczne zwraca liczbę (double)
  - sval – pole publiczne zwraca tekst (String)

- int nextToken() – sprawdzenie i odczetuje następny ciąg ze strumienia, zwraca wartość ttype - typu tokena:
  - TT_NUMBER – odebrał liczbę
  - TT_WORD – odebrał słowo (wyraz)
  - TT_EOL – koniec lini
  - TT_EOF – koniec łańcucha
- Typ tokena można odczytać również z pola publicznego ttype

StreamTokenizer – cd.

```
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.StreamTokenizer;

public class Strumien {
    public static void main(String[] args) {
        try {
            InputStreamReader inp = new InputStreamReader(System.in);
            StreamTokenizer stk = new StreamTokenizer(inp);
            System.out.println("Podaj a:");
            while (stk.nextToken() != StreamTokenizer.TT_NUMBER) {
                System.out.println("Jeszcze raz a:");
            }
            double a = stk.nval;
        } catch (IOException ex) {}
    }
}
```

Klasa scanner

- JDK co najmniej 1.5
- Obsługuje strumienie których źródłem jest: InputStream, File, String, inne typu Readable
- Trzeba zaimportować java.util.Scanner;
- Bogaty zestaw metod odczytu i weryfikacji typu tokena:

```
String next()
String next(String pattern)
BigDecimal nextBigDecimal()
BigInteger nextBigInteger()
boolean nextBoolean()
byte nextByte()
double nextDouble()
float nextFloat()
int nextInt()
String      nextLine()
long nextLong()
short nextShort()
```

oraz odpowiednie metody sprawdzania typu wartości zwracające boolean:

```
hasNext()
 hasNext(String pattern)
 hasNextBigDecimal()
 hasNextBigInteger()
 hasNextBoolean()
 hasNextByte()
 hasNextDouble()
 hasNextFloat()
 hasNextInt()
 hasNextLine()
 hasNextLong()
 hasNextShort()
```

Uwaga:
Wywołanie **next…()** powoduje odczytanie, **hestNext..()** sprawdza jedynie stan bitu

Uwaga:
Skaner czyta poszczególne tokeny identyfikowane przez białe znaki

Scanner przykład 1

```
public class StrumienSkanerA {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String takNie;
        do  {
            System.out.println("Wybierz Tak lub Nie (T/N)");
            takNie = sc.next();
        } while ("t".equalsIgnoreCase(takNie)
                || "n".equalsIgnoreCase(takNie));
        System.out.println("Wybrałeś: " + takNie);
    }
}
```

Scanner przykład 2

```
import java.util.Scanner;

public class Strumien {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (!sc.hasNextInt()) {
            System.out.println("Podaj liczbę jeszcze raz");
            sc.next();
        }
        int i = sc.nextInt();
        System.out.println("Podałeś: " + i);
    }
}
```

Obsługa plików
Odczyt

- Import java.io.*;
- Przydatne klasy:
  - File – opis pliku
  - FileInputStream – strumień danych wejściowych
- Oraz:
  - BufferedReader
  - Scanner
  - DataInputStream – Pozwala aplikacji na odczyt danych prymitywnych (int, boolean, double etc.) z odpowiedniego strumienia wejściowego.

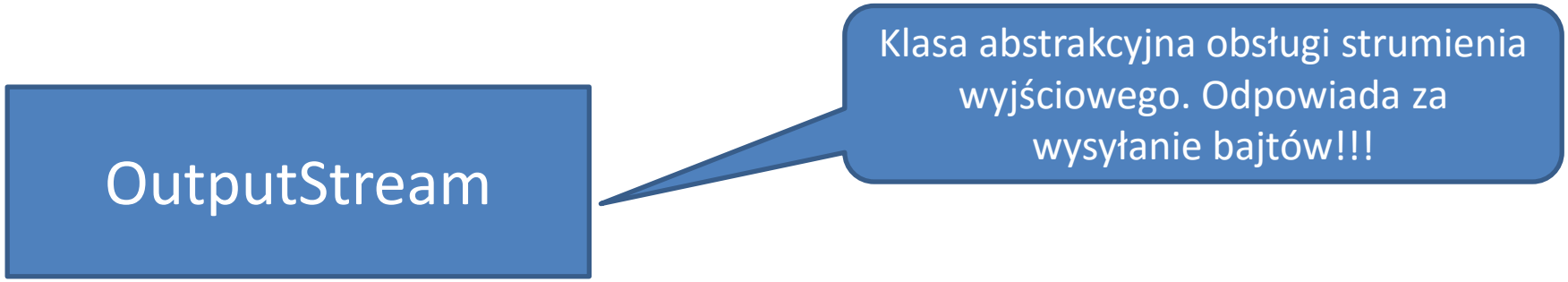
Przykład – odczyt z pliku BufferedReader

```
public class PlikZapis1 {
    public static void main(String[] arg) {
        String line = "";
        FileInputStream fIn = null;
        File file = new File("test.txt");
        BufferedReader inbr = null;
        try {
            fIn = new FileInputStream(file);
            inbr = new BufferedReader(new InputStreamReader(fIn));
            while ((line = inbr.readLine()) != null) {
                //null wystąpi jesli koniec pliku
                System.out.print(line + "\n");
            }
        } catch (IOException e) {
            System.out.println("Read/write error");
        } finally {
            if (inbr != null)
                try { inbr.close(); } catch (IOException ex) {}
        }
    }
}
```

Przykład - Odczyt pliku Scanner

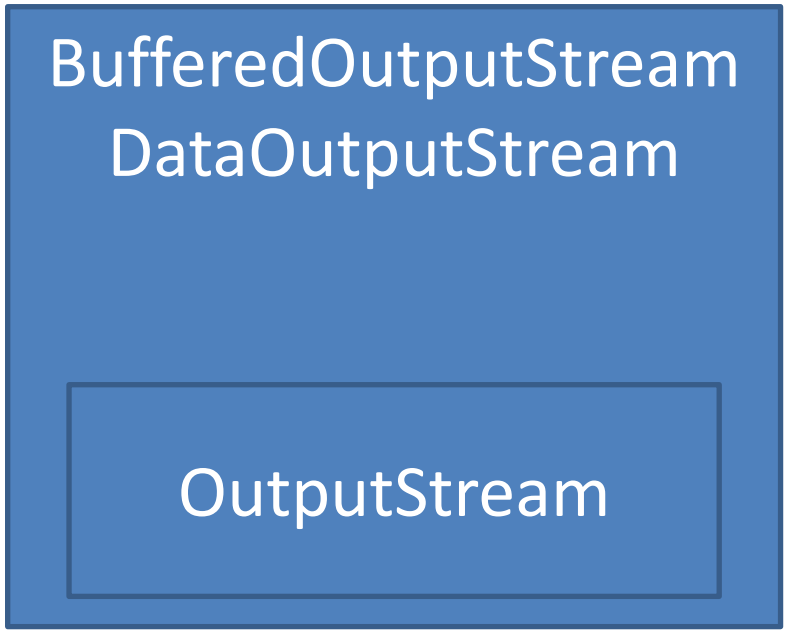
```
public class PlikZapis2 {
    public static void main(String[] arg) {
        File file = new File("file1.txt");
        try (Scanner fileScanner = new Scanner(file);){
            do {
                String line = fileScanner.nextLine();
                System.out.println(line);
            } while (fileScanner.hasNextLine());
        } catch (FileNotFoundException e) {
            System.out.println("Brak pliku!");
        }
    }
}
```

Strumienie wyjściowe



Podklasy klasy OutputStream:

- ByteArrayOutputStream,
- FileOutputStream,
- FilterOutputStream,
- ObjectOutputStream,
- PipedOutputStream



Klasy obsługi strumieni wyjściowych

- PrintStream – pozwala na wysyłanie danych, wraz z ich sformatowaniem, nie wyrzuca wyjątków, wyjątki sprawdzane przez metdę chceckError, umożliwia autoflush
- PrintWriter – klasa przeznaczona do obsługi „text output stream”
- DataOutputStream – klasa przeznaczna do wysyłania danych binarnych (np. liczby zmiennoprzecinkowe przesyłane sa jako zmiennoprzecinkowe i zajmują np. 8 bajtów dla double, bajt dla byte itd.)

Klasa PrintStream (PrintWriter podobna składnia)

Konstruktory i możliwe rozwiązania:

Constructor Summary

`PrintStream(File file)` Creates a new print stream, without automatic line flushing, with the specified file.
`PrintStream(File file, String csn)` Creates a new print stream, without automatic line flushing, with the specified file and charset.
`PrintStream(OutputStream out)` Creates a new print stream.
`PrintStream(OutputStream out, boolean autoFlush)` Creates a new print stream.
`PrintStream(OutputStream out, boolean autoFlush, String encoding)` Creates a new print stream.
`PrintStream(String fileName)` Creates a new print stream, without automatic line flushing, with the specified file name.

Metody:

PrintStream append(char c) Appends the specified character to this output stream.

void close() Closes the stream.

void flush() Flushes the stream.

void print(Dowolny_typ b) Prints Dowolny_typ

PrintStream printf(String format, Object... args) A convenience method to write a formatted string to this output stream using the specified format string and arguments.

void println(Dowolny_typ b) Prints Dowolny typ i przechodzi do nowej linii

Przykład – zapis do pliku PrintStream

```
public class PlikZapis1 {
    public static void main(String[] args) {
        String str;
        PrintStream filePrintStream = null;
        try {
            filePrintStream = new PrintStream("file1.txt");
            Scanner sc = new Scanner(System.in);
            do {
                str = sc.next();
                filePrintStream.println(str);
            } while ("koniec".equals(str));
        } catch (FileNotFoundException e1) {
            System.out.println("Pliku nie znaleziono");
            System.exit(-1);
        } finally {
            if (filePrintStream != null){
                filePrintStream.close();
            }
        }
    }
}
```

Przykład – zapis do pliku PrintStream

```
public class PlikZapis1 {
    public static void main(String[] args) {
        String str;
        try (PrintStream filePrintStream =
                new PrintStream("file1.txt")) {
            Scanner sc = new Scanner(System.in);
            do {
                str = sc.next();
                filePrintStream.println(str);
            } while ("koniec".equals(str));
        } catch (FileNotFoundException e1) {
            System.out.println("Pliku nie znaleziono");
            System.exit(-1);
        }
    }
}
```

Przykład – Bezpośrednia obsługa strumieni

```
public static void main(String[] args) {
    try (FileInputStream fileIn = new FileInputStream("dane.txt");
            FileOutputStream fileOut = new FileOutputStream("wynik.txt");) {
        int bufferSize = 1024;
        byte[] buffer = new byte[bufferSize];
        while (true){
            int leng = fileIn.read(buffer);
            if (leng == -1){
                break;
            }
            fileOut.write(buffer,0,leng);
        }
    } catch (IOException ex) {
        System.out.println("Brak pliku");
    }
}
```

Obsługa plików cd.

- Klasa RandomAccessFile – dostęp do pliku jako tablicy typy byte – sekwencyjny odczyt bajtów
- Konstruktor
  - *RandomAccessFile(String fname, String AccessMethod)*
  - *RandomAccessFile(File file, String AccessMethod)*
    - *"r" Otwarcie pliku tylko do odczytu, próba zapisu zostanie zakończona wyjątkiem IOException.*
    - *„rw" Otwarcie pliku do zapisu i odczytu*
- *Metody:*
  - *read(byte[] b)* – odczytuje dane z pliku do tablicy *b*
  - *double ReadDouble() – odczytuje z pliku wartość typu double*
  - *String readLine()* - odczytuje następną linię z pliku
  - writeDouble(double d) – zapisuje wartość typu doble
  - writeChars(String s) – zapisuje wartość typu string
  - seek(long pos) – wskaźnik pliku – podaje miejsce z którego należy rozpocząć czytanie z pliku

Zapis

Odczyt

```
public class RAF {

    public static void main(String[] args) {
        RandomAccessFile rf = null;
        try {
            rf = new RandomAccessFile("rtest.dat", "rw");
            for (int i = 0; i < 10; i++) {
                rf.writeDouble(i * 2);
            }
        } catch (IOException e) {
        } finally {
            if (rf != null) {
                try {
                    rf.close();
                } catch (IOException ex) {}
        } }

        try {
            rf = new RandomAccessFile("rtest.dat", "r");
            for (int i = 0; i < 10; i++) {
                System.out.println("Value " + i + ": " + rf.readDouble());
            }
        } catch (IOException e) {
            System.out.println("IO Exception 2");
        } finally {
            if (rf != null) {
                try {
                    rf.close();
                } catch (IOException ex) { }
        } }
} }
```