POLITECHNIKA GDAŃSKA

Język Java podstawy

Jacek Rumiński

Język Java podstawy

Jacek Rumiński

Katedra Inżynierii Biomedycznej,
Wydział Elektroniki, Telekomunikacji i Informatyki
Politechnika Gdańska

1. Operacje wejścia/wyjścia
2. Operacje na strumieniach znaków
3. Operacje w systemie plików

Operacje wejścia - operacje na danych, które są wprowadzane do programu

Operacje wyjścia - operacje na danych, które są wyprowadzane z programu

Dane można traktować jako dane proste, np. 1 bajt, wiele bajtów, itp. lub dane sformatowane, np. liczba całkowita, ciąg znaków, obraz (czyli zestawy bajtów).

W najprostszym podejściu możemy traktować dane jako sekwencję bajtów.

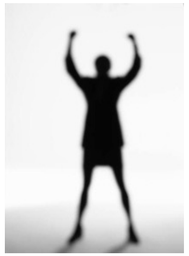
Taką sekwencję bajtów nazwano roboczo (tworząc pewną abstrakcję) jako STRUMIEŃ (w języku angielskim STREAM).

Dlatego czasami mówimy również o strumieniach wejścia (INPUT STREAM) lub o strumieniach wyjścia (OUTPUT STREAM).

Wprowadzenie do operacji wejścia/wyjścia: strumienie

Dlaczego nazwa strumień (stream)? Strumień oznacza sekwencję bajtów, które są transportowane z jednego miejsca do drugiego. Jest więc bardzo wygodna abstrakcja (cokolwiek się przemieszcza, zawsze są to bajty).

Podstawowe klasy strumieni InputStream/OutputStream



WY

strumień wody

strumień bajtów

sekwencja bajtów

WE

OutputStream

InputStream

Podstawowe klasy strumieni InputStream

Klasa abstrakcyjną zawierająca podstawowe metody odczytu i kontroli bajtów ze strumienia.

Jedyną metodą abstrakcyjną (czyniącą z tej klasy klasę abstrakcyjną) jest metoda read() oznaczająca czytanie kolejnego bajtu ze strumienia wejścia.

Obiekt tej klasy można uzyskać poprzez odwołanie się do standardowego wejścia zainicjowanego zawsze w polu in klasy System, czyli System.in.

```
import java.io.*;
public class EchoJedi{
 public static void main(String args[]){
   byte b[] = new byte[100];
            try{        //wczytaj dane ze standardowego wejścia (klawiatura) do tablicy
                        System.in.read(b);
                        /* System.in to tworzony przez Maszynę Wirtualną obiekt jakiejś
                         * klasy, rzutowany na typ klasy InputStream (upcasting). */
                        //wyślij bajty z tablicy do standardowego wyjścia (ekran)
                        System.out.write(b);
                        /* System.out to tworzony przez Maszynę Wirtualną obiekt klasy
                         * PrintStream, która dziecziczy po OutputStream. */
                         System.out.flush();
                        /* jeśli strumień jest buforowany (operacje na bloku danych, celem
                         * zwiększenia wydajności dane zapisywane są do bloku; jeśli blok
                         * jest pełny, cały blok wysyłany jest do strumienia) to wymuś
                         * przesłanie danych z bloku do strumienia.*/
                        } catch (IOException ioe){ //funkcje odczyty mogą generować wyjątki
                        System.out.println("Błąd wejścia-wyjścia: "+ioe);
            }//koniec try
   }//koniec main()   }//koniec public class EchoJedi
```

Podstawowe klasy strumieni InputStream

Pozostałe metody (poza read()) umożliwiają:

- odczyt bajtów do zdefiniowanej tablicy:
  - int read(byte b[]);
  - int read(byte b[], int offset, int length);
- pominięcie określonej liczby bajtów w odczycie:
  - long skip(long n);
- kontrolę stanu strumienia (czy są dane):
  - int available();
- zamknięcie strumienia:
  - void close().

Podstawowe klasy strumieni InputStream

Wybrane klasy dziedziczące po klasie InputStream to:

- ByteArrayInputStream – strumień o źródle w tablicy bajtów (pamięć, wygodny przy operacjach wymagających wysokiej wydajności, np. kodowanie),
- FileInputStream – strumień umożliwiający odczyt z pliku,
- FilterInputStream – strumień umożliwiający operacje na bajtach w sekwencji poprzez klasy dziedziczące, np. DataInputStream, która umożliwia programowi odczyt danych zgodnie z podstawowymi typami danych, char, int, long, double, a nie byte[].
- ObjectInputStream – odtworzenie obiektu (readObject()) z jego zapisu w formie sekwencji bajtów (z pliku, z sieci, itd.) dokonuje rekonstrukcji obiektu z sekwencji bajtów

Podstawowe klasy strumieni OutputStream

W podobny sposób, niemniej dotyczący obsługi wyjścia, definiowane są klasy dziedziczące z klasy OutputStream.
Klasa ta jest również klasą abstrakcyjną z jedyną abstrakcyjną metodą write() zapisująca kolejny bajt do strumienia. Podstawowe metody tej klasy to:

- void close() - zamknięcie strumienia,
- void flush() - przesuwa buforowane dane do strumienia,
- void write(byte[] b, int off, int len), zapisuje len bajtów z tablicy b począwszy od off, do strumienia wyjścia.
- void write(byte[] b) - zapisują dane z tablicy b do strumienia wyjścia.

Klasy dziedziczące jak dla InputStream, tylko człon nazwy "Input", należy zamienić na "Output", np. FileOutputStream.

Podstawowe klasy strumieni OutputStream

Interesującą klasą dziedziczącą po klasie FilterOutputStream (a ta po OutputStream) jest klasa PrintStream.
Właściwie znamy już tę klasę. Pole System.out jest obiektem tej klasy, a najczęściej używana przez nas metoda to System.out.println(), czyli metoda println() klasy PrintStream.
Ponadto klasa ta ma m.in. metody:

- print(); wysłanie wartości danego typu podstawowego oraz wartości ciągu znaków dla klasy String,
- println(); jak dla print(), tylko dodatkowo przesłany znak nowej linii (na końcu)
- printf(); formatowanie przesyłanego ciągu znaków jak w języku "C", np.:

```
printf("Jacek ma %d lat, co w kodzie szesnastkowym wynosi %x.", 38, 38);
```

Wynik: "Jacek ma 38 lat, co w kodzie szesnastkowym wynosi 26."

```
import java.io.*;
public class TransferJedi{
  static String nazwaPliku="test.dat";
  public static void main(String args[]){
    byte b[] = new byte[100]; double liczba;
      try{
        System.out.println("Wprowadź liczbę i naciśnij ENTER:");          System.in.read(b);
        liczba=Double.parseDouble(new String(b));//bajty na ciąg znaków; ciąg na liczbę
        System.out.println("LICZBA wprowadzona:\t\t "+liczba);
        //otwórz/stwórz plik o nazwie "test.dat", otwarty strumień filtruj
        DataOutputStream dos=new DataOutputStream(new FileOutputStream(nazwaPliku));
       dos.writeDouble(liczba); dos.close(); //filtruj formatując bajty jako liczby double
       //otwórz plik, otwarty strumień filtruj
       DataInputStream dis=new DataInputStream(new FileInputStream(nazwaPliku));
       liczba=dis.readDouble(); dis.close(); //filtruj czytając bajty jako liczbę double
       System.out.println("LICZBA odczytana z pliku:\t "+liczba);
     } catch (NumberFormatException nfe){//wyjątek, ktoś podał ciąg znaków inny niż liczba
        System.out.println("Błędnie podana wartość (format) liczby rzeczywistej.");
     } catch (IOException ioe){ //funkcje odczytu mogą generować wyjątki
       System.out.println("Błąd wejścia-wyjścia: "+ioe);        }//koniec try   }//koniec main()
}//koniec public class TransferJedi
```

```
import java.io.*;
class PlanyJedi implements Serializable{

  private String tajneHaslo="mojeHaslo";
  public String pobierzPlany(String haslo){
              if(haslo.equals(tajneHaslo)){
                          return "Takie sobie tajne plany...";
              }else return "Brak planow";
  }//koniec pobierzPlany()

}//koniec class PlanyJedi
```

c.d.n.

```
public class MisjaJedi{
  static String nazwaPliku="plany.dat";
  public static void main(String args[]){
    byte b[] = new byte[100];
     try{
       //otwórz/stwórz plik o nazwie "plany.dat", otwarty strumień filtruj
       ObjectOutputStream oos=new ObjectOutputStream(new FileOutputStream(nazwaPliku));
       oos.writeObject(new PlanyJedi()); oos.close(); //zapisz obiekt klasy PlanyJedi i zamknij
       //otwórz plik, otwarty strumień filtruj
       ObjectInputStream ois=new ObjectInputStream(new FileInputStream(nazwaPliku));
       PlanyJedi pj=(PlanyJedi)ois.readObject(); ois.close();//z pliku czytamy obiekt!
       System.out.println("Podaj hasło:");
       System.in.read(b);    String haslo=new String(b);
       haslo=haslo.trim();//usuń puste znaki (zerowe bajty z b)
       String plany=pj.pobierzPlany(haslo);
       System.out.printf("ODCZYTANE PLANY: %s",plany);
     } catch (Exception ioe){ //funkcje odczytu mogą generować wyjątki
       System.out.println("Błąd wejścia-wyjścia: "+ioe);
     }//koniec try
    }//koniec main()
}//koniec public class MisjaJedi
```

1. Operacje wejścia/wyjścia
2. Operacje na strumieniach znaków
3. Operacje w systemie plików

Podstawowe klasy strumieni znaków

W związku z problemem wynikającym z konwersji znaków Javy (Unicode) na bajty i odwrotnie występujących we wczesnych (JDK 1.0) realizacjach klas obsługi strumieni począwszy od wersji JDK1.1 wprowadzono dodatkowe klasy Reader i Writer.

Obie abstrakcyjne klasy są analogicznie skonstruowane (dziedziczenie z klasy Object i deklaracja metod) jak klasy InputStream oraz OutputStream.

Klasy dziedziczące po Reader i Writer zapewniają ciekawe możliwości zastosowania.

Odczyt danych odbywa się poprzez zastosowanie metod read() lub readLine() natomiast zapis danych do strumienia poprzez wykorzystanie metod write().

Wybrane klasy dziedziczące po klasie Reader/Writer:

- BufferedReader – buforuje otrzymywany tekst (czytaj linię tekstu!),
- InputStreamReader – czyta bajty zamieniające je na tekst według podanego systemu kodowania znaków,
- FileReader – odczyt danych tekstowych z pliku dla domyślnego systemu kodowania znaków, poprzez podanie ścieżki zależnej systemowo (String) lub abstrakcyjnej (File)
- StringReader – obsługa strumienia pochodzącego od obiektu klasy String.

Dla klasy Writer analogicznie, tylko …Writer, np. FileWriter.

Dodatkowa klasa to PrintWriter (analog klasy PrintStream).

```
import java.io.*;
public class NoweEchoJedi{
  public static void main(String args[]){
    PrintWriter pw=null;
    try{
            pw= new PrintWriter(new OutputStreamWriter(System.out,"Cp852"),true);
            //true na końcu oznacza, że dane są przesyłane od razu (bez flush())
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String test;
            //Jeśli chcemy dodać znak specjalny, np. cudzysłów musimy użyć znaku \"
            pw.println("Wprowadź dane. Linia z \".\" oznacza koniec wprowadzania.");

            //readLine() - czytaj całą linię
            while(!(test=br.readLine()).equals(".")){
                    pw.println("Wprowadzono: \t"+test);
            }
            pw.close();
    }catch (Exception e){ System.out.println("Wyjatek: "+e);}
  }//koniec main()
}//koniec public class NoweEchoJedi
```

1. Operacje wejścia/wyjścia
2. Operacje na strumieniach znaków
3. Operacje w systemie plików

Operacje na plikach

Dostęp do plików zaprezentowany wcześniej wykorzystywał klasy FileInputStream, FileOutputStream, FileReader i FileWriter.

Konstruktory tych klas umożliwiają utworzenie strumienia poprzez podanie ścieżki do pliku jako argumentu.

Ścieżkę można podać stosując dwie metody:

1. poprzez ciąg znaków (obiekt klasy String)
2. poprzez obiekt klasy File, reprezentujący logiczną ścieżkę do plików i katalogów.

Ścieżka dostępu do pliku może być sklasyfikowana ze względu na jej zasięg lub ze względu na środowisko, dla którego jest zdefiniowana (np. Windows, Linux). W pierwszym przypadku dzieli się ścieżki dostępu na absolutne i relatywne. Absolutne to te, które podają adres do pliku względem głównego korzenia systemu plików danego środowiska. Relatywne to te, które adresują plik względem katalogu bieżącego.

Operacje na plikach

Tworząc obiekt klasy File dokonywana jest konwersja łańcucha znaków na abstrakcyjną ścieżkę dostępu do pliku (abstrakcyjna ścieżka dostępu do pliku jest tworzona według określonych reguł podanych w dokumentacji API).
Metody klasy File umożliwiają kontrolę podanej ścieżki i plików (np. isFile(), isDirectory(), isHidden, canRead(), itp.) oraz dokonywania konwersji (np. getPath(), getParent(), getName(), toURL(), itp.), jak i wykonywania prostych operacji (list(), mkdir(), itp.).

Uwaga! Należy pamiętać, że zapis tekstowy ścieżki dostępu dla środowiska MS Windows musi zawierać podwójny separator, gdyż pojedynczy znak umieszczony w ciągu znaków oznacz początek kodu ucieczki (czyli znak specjalny), np. „c:\\java\ \kurs\\wyklad\\np".

```
import java.io.*;  import java.util.*;
public class SystemPlikowJedi{
  public static void main(String args[]){
      //utworzenie obiektu File oznacza adresowanie domniemanego węzła w systemie plików
      File f = new File("DANE_JEDI");
      if(f.exists()){//jeśli dany węzeł istnieje
        if(f.isDirectory()){//jeśli jest to katalog
         System.out.println("Katalog: "+f.getAbsolutePath()+" już istniał.\nData modyfikacji: "+
         new Date(f.lastModified())+". Usuwam go.\n");
         f.delete();//usuń go
        }//if f.isDirectory()
      }//if f.exists()
      if (f.mkdir()) {//jeśli uda się utworzyć katalog
            File g = new File (".");//adresuj bieżący węzeł zawierający nowy katalog
            String s[] = g.list();//generuj listę węzłów (plików i katalogów)
            System.out.println("Zawartość katalogu "+g.getAbsolutePath()+": \n");
            for (int i =0; i<s.length; i++){
                        System.out.println(s[i]);//wyświetl kolejne nazwy          }//for
     } else {       System.out.println("Nie można utworzyć katalogu!");          }//else
            }//koniec main()
}//koniec public class SystemPlikowJedi
```

Zapraszamy na kolejne zajęcia