Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej PP

Java – wejście i wyjście

1. Wprowadzenie

Programy Javy komunikują się z otoczeniem za pomocą strumieni znakowych i bajtowych. Strumienie bajtowe korzystają z dwóch hierarchii klas, na szczycie których znajdują się klasy abstrakcyjne `InputStream` i `OutputStream`. Najważniejszymi metodami w obu klasach są metody abstrakcyjne `read()` i `write()`.

Także strumienie znaków korzystają z dwóch hierarchii klas. Na ich szczycie znajdują się klasy abstrakcyjne `Reader` i `Writer`, a w nich metody `read()` i `write()`.

Importowany automatycznie do programów Javy pakiet `java.lang` zawiera klasę `System`, z predefiniowanymi zmiennymi strumieniowymi `in` (typu `InputStream`), `out` i `err` (oba typu `PrintStream`).

2. Wejście standardowe

Aby odczytać dane z konsoli, otacza się `System.in` obiektem klasy `BufferedReader`, obsługującym buforowany odczyt ze strumienia wejściowego. Podstawowa wersja konstruktora tej klasy to:

```
BufferedReader(Reader czytnik_wejścia)
```

`Reader` (klasa abstrakcyjna) zawiera podklasę `InputStreamReader`, odpowiedzialną za konwersję bajtów na znaki. Utworzenie obiektu `InputStreamReader` powiązanego z `System.in` odbywa się przy wykorzystaniu konstruktora:

```
InputStreamReader(InputStream strumien_wejściowy)
```

Przykład:

```
BufferedReader br = new BufferedReader
                    (new InputStreamReader(System.in));
String Str;
Str = br.readLine();
char c; int liczba;
c = (char) br.read();
liczba = Integer.parseInt(br.readLine());
```

3. Wyjście standardowe

Metody `print()` i `println()` są zdefiniowane w klasie `PrintStream`, będącej typem zmiennej referencyjnej `System.out`.

Przykład:

```
System.out.println("napis " + tablica[0][0]);
```

Innym, eleganckim rozwiązaniem jest zastosowanie do wyprowadzania na konsolę klasy znakowej `PrintWriter`. Najczęściej używany konstruktor tej klasy to:

```
PrintWriter(OutputStream strumień_wyjściowy,
            boolean flaga)
```

Flaga informuje o tym, czy Java ma przesyłać dane po każdym wywołaniu metody `println()`. Standardowo ustawiona jest na `true`. `PrintWriter` dostarcza metody `print()` i `println()`, obsługujące wszystkie typy danych.

Przykład:

```
PrintWriter pw = new PrintWriter(System.out, true);
pw.println(”ciag znakow”);
int i = 5;
pw.println(i);
```

Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej PP

4. Wejście/wyjście niestandardowe

W Javie wszystkie pliki mają postać bajtową. Tradycyjnie, do odczytu bajtów stosuje się klasy `FileInputStream` i `FileOutputStream`, tworzące strumienie bajtów powiązane z plikami.

Przykład:

```
FileInputStream fin;
FileOutputStream fout;
fin = new FileInputStream(”c:/plikwe.txt”);
fout = new FileOutputStream(”c:/plikwy.txt”);
char zn;
zn = fin.read();
fout.write(zn);
//....
fin.close();
fout.close();
```

By umożliwić odczyt/zapis znakowy do pliku, Java oferuje klasy `FileReader` i `FileWriter`. W celu uzyskania wejścia/wyjścia buforowanego, otacza się obiekty tych klas obiektami, odpowiednio typu `BufferedReader` i `BufferedWriter`.

Przykład:

```
try {
      FileReader fr = new FileReader(”c:/plik1.txt”);
      BufferedReader br = new BufferedReader(fr);
      String s;
      while(((s = br.readline()) != null) {
          System.out.println(s);
      }
      BufferedWriter bw =
         new BufferedWriter(
                        new FileWriter(”c:/plik2.txt”));
      bw.write(tablica[0][0] + " " + tablica[1][1]);
      bw.newLine();
      //....
      plik.close();
      br.close();
    }
catch (IOException e) {
       System.err.println("Error: " + e);
    }
```

5. Wejście - skaner

Do wczytywania wartości typów prostych lub łańcuchów znaków można tez użyć skanera – obiektu klasy wbudowanej `Scanner`. Działanie takiego skanera jest oparte na wykorzystaniu wyrażeń regularnych, definiujących dopuszczalne postaci tych wartości. Używając białej spacji w charakterze ogranicznika, skaner wyodrębnia z wejściowego ciągu znaków tokeny, które można poddać konwersji na żądany typ wartości. Do tej konwersji służą metody `nextInt`, `nextDouble`, itd.

Przykład:

```
File file = new File(”sciezka dostępu do pliku”);
Scanner sc = new Scanner(file);
             // Scanner sc = new Scanner(System.in);
int l_calk = sc.nextInt();
```

6. Konwersja

Konwersja tekstu na liczby i odwrotnie odbywa się przy wykorzystaniu metod `parseByte()`, `parseShort()`, `parseInt()`, `parseDouble()` i `toString()`, zdefiniowanych w klasach `Short`, `Byte`, `Integer` oraz `Double`.

Przykład:

```
String str = br.readLine(); //br – wejście buforowane
int i;
i = Integer.parseInt(str);
```

Pamiętaj o konieczności dołączania pakietu `java.io` komendą:

```
import java.io.*
```

Beata Jankowska
Instytut Automatyki, Robotyki i Inżynierii Informatycznej PP

7. Hierarchia wybranych klas Javy

```
Object → InputStream (m. read()) → FileInputStream
       | → OutputStream (m. write())
                    → FileOutputStream
                   | → FilterOutputStream → PrintStream
                                          (m. print(), println())
       | → Reader (m. read())
                    → InputStreamReader → FileReader
                   | → BufferedReader
       | → Writer (m. write())
                    → OutputStreamWriter → FileWriter
                   | → PrintWriter (m. print(), println())
                   | → BufferedWriter
       | → Scanner (m. nextByte, nextDouble, nextFloat,
                      nextInt, ...)
       | → System (z. in:InputStream, out:PrintStream,
                    err:PrintStream)
       | → Number   → Byte   (m. parseByte())
                   | → Double (m. parseDouble())
                   | → Float  (m. parseFloat())
                   | → Integer (m. parseInt())
                   | → Long   (m. parseLong())
                   | → Short  (m. parseShort())
       | → String (m. toString())
```