Datu ievads valodā Java

Vienkāršs ievads un izvads

Visvienkāršākā klase datu ievadam ir `Scanner`. Ar tās palīdzību var uzreiz lasīt dažādu tipu datus. Izvadam var izmantot `System.out`. Diemžēl, abas šīs klases nenodrošina pietiekami ātru datu ievadu un izvadu un sacensību apstākļos lielāka apjoma datiem var izrādīties par lēnu.

```
import java.util.Scanner;

public class IOTest {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);

        int a = in.nextInt();
        int b = in.nextInt();
        System.out.println(a + b);

        System.out.flush();
    }
}
```

Ātrs ievads un izvads

Ātram datu ievadam ieteicams izmantot klasi `StreamTokenizer`. Šī klase automātiski sadala ievadu skaitļu un teksta daliņās. Lai nolasītu kārtējo daļiņu, jāizsauc komanda `nextToken()`. Skaitļu un teksta vērtības tiek saglabātas attiecīgi laukos `nval` un `sval`.

Ātram datu izvadam izmantojiet klasi `PrintWriter`. Nav ieteicams lietot tās metodi `printf()`, kas var strādāt lēni (izmantojiet `print()` var `println()`). Tāpat nav ieteicams izvadīt vairākos mainīgos reizē, kas arī var palēlināt izvadu (piemēram, `out.println(a + " " + b)` vietā labāk lietot `out.print(a); out.print(" "); out.println(b);`).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

public class IOTest {
    static StreamTokenizer in;
    static PrintWriter out;

    static int nextInt() throws IOException {
        in.nextToken();
        return (int) in.nval;
    }

    static String nextString() throws IOException {
        in.nextToken();
        return in.sval;
    }

    public static void main(String[] args) throws IOException {
```

```
        in = new StreamTokenizer(new BufferedReader(new java.io.InputStreamReader(System.in)));
        out = new PrintWriter(System.out);

        int a = nextInt();
        int b = nextInt();
        out.println(a + b);

        String s = nextString();
        out.println(s);

        out.flush();
    }
}
```

Long tipa skaitļu lasīšana

Skaitliskās vērtības `StreamTokenizer` glabā `double` tipa laukā `nval`, kura precizitātes nepietiek, lai saglabātu `long` tipa skaitļus. Lai nolasītu šādus skaitļus, var nodefinēt zīmes un ciparus kā tekstuālas vērtības priekš `StreamTokenizer`. Tad skaitli var nolasīt kā tekstu un nopārsēt iegūto virkni par skaitli. Jāņem vērā, ka šajā gadījumā visi skaitļi tiks lasīti kā teksts (arī `int` lieluma skaitļi).

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StreamTokenizer;

public class IOTest {
    static StreamTokenizer in;
    static PrintWriter out;

    static Long nextLong() throws IOException {
        in.nextToken();
        return Long.parseLong(in.sval);
    }

    public static void main(String[] args) throws IOException {
        in = new StreamTokenizer(new BufferedReader(new java.io.InputStreamReader(System.in)));
        in.resetSyntax();
        in.whitespaceChars(0, 32);
        in.wordChars('0', '9');
        in.wordChars('-', '-');
        out = new PrintWriter(System.out);

        long a = nextLong();
        long b = nextLong();
        out.println(a + b);

        out.flush();
    }
}
```

Citas metodes

Dažreiz vēl ātrāk par `StreamTokenizer` strādā pati `BufferedReader` klase, un citos gadījumos var būt efektīvāk pielietot `StringTokenizer` klasi. Par šiem rīkiem sīkāk var izlasīt Java dokumentācijā. Tomēr `StreamTokenizer` ātrdarbība parasti ir pietiekama visos gadījumos.