Prednáška 6: Vstupno/výstupný systém Javy a viacniťovosť

Objektovo-orientované programovanie 2012/13

Valentino Vranić

Ústav informatiky a softvérového inžinierstva
Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

3. aprl 2013

Obsah prednášky

Vstupno/výstupný systém Javy

Vstupno/výstupný systém Javy

- `java.io`
  - Založený na koncepte prúdu údajov (stream)
- `java.nio` (*new* I/O; od verzie 1.4)
  - Založený na kanáloch a vyrovnávacích pamätiach (channels and buffers)
  - Rýchlejší
  - Lepšia podpora znakových sád
  - Transparentná práca s veľkými súbormi
- Pre určité veci je však `java.io` nevyhnutný – napr. pre serializáciu objektov a štandardný vstup a výstup
- `java.io` bol reimplementovaný pomocou `java.nio` – zvýšenie rýchlosti aj samotného `java.io`

Práca s adresármi

- Trieda `File`
  - abstraktná reprezentácia súborov a ciest (*pathname*)
- Abstraktná cesta: systémovo závislý prefix (voliteľne) + postupnosť názvov (žiaden alebo viac)
- Cesta – absolútna alebo relatívna
- Aktuálny adresár je v systémovej vlastnosti `user.dir`

```
System.getProperty(user.dir);
```

Obsah adresára

```
import java.io.*;
import java.util.*;

public class Dir {
    public static void main(String[] args) {
        File path = new File(".");
        String[] list;
        list = path.list();
        for(int i = 0; i < list.length; i++)
          System.out.println(list[i]);
    }
}
```

Filtrovaný obsah adresára

```
import java.io.*;
import java.util.*;

public class FDir {
    public static void main(String[] args) {
        File path = new File(".");
        String[] list;
        list = path.list(new XFilter());
        for(int i = 0; i < list.length; i++)
            System.out.println(list[i]);
    }
}
class XFilter implements FilenameFilter {
    public boolean accept(File dir, String name) {
        return name.charAt(0) == 'x';
    }
}
```

Ďalšie metódy triedy File

- exists()
- delete()
- isFile()
- isHidden(
- mkdirs()
- renameTo()

...

V/V systém Javy založený na prúdoch

- Prúd bajtov (byte stream)
  - Abstraktné triedy `InputStream` a `OutputStream`
  - Bajty – teda len 8-bitové znaky
- Prúd znakov (character stream)
  - Abstraktné triedy `Reader` a `Writer`
  - 16-bitové znaky – podpora Unicode
- Od týchto tried sú odvodené ďalšie pre rôzne typy údajov, ktoré sa čítajú a zapisujú
- Mimo tohto je trieda `RandomAccessFile` – vo verzii 1.4 ju v podstate nahradili tzv. memory-mapped files

Otvorenie a zatvorenie prúdu

- Prúd sa otvára konštrukciou

```
FileReader file = new FileReader("data.txt");
```

- Prúd sa zatvára metódou close()

```
file.close();
```

- Toto by sa malo udiať pri finalizácii, ale nie je to zaručené – lepšie zatvárať prúdy explicitne
- Nikdy však nezatvárajte štandardný vstup, štandardný výstup a štandardný výstup pre chyby (vrátane obaľovacích prúdov)
  - `System.in`, `System.out` a `System.err`
  - Tieto atribúty sú finálne – nedá sa im priradiť nový prúd

java.io cez príklady

- Veľmi užitočný príklad `IOStreamDemo.java` v TiJ
- Nasledujú typické použitia `io` – séria príkladov podľa programu `IOStreamDemo.java` z TiJ
  - pridať: **import** java.io.*;

Čítanie súboru po riadkoch

```
public class Subor {
    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new FileReader("Subor.java"));
        String s, s2 = new String();

        while ((s = in.readLine())!= null)
            s2 += s + "\n";

        in.close();
    }
}
```

- Výnimka `IOException`
- Obalenie vstupného prúdu do objektu triedy `BufferedReader` pre rýchlejší a jednoduchší prístup k údajom

Štandardný V/V systém

- Koncept štandardného V/V systému
- V Jave:
  - štandardný vstup: `System.in`
  - štandardný výstup: `System.out`
  - štandardný výstup pre chyby: `System.err`
- Štandardný výstup a výstup pre chyby sú obalené (wrapped) v objekte typu `PrintStream` – pracujú so znakmi
- Štandardný vstup je však `BufferedInputStream` (bajty) – vhodné je obaliť ho pred použitím
- Na to sa využíva trieda `InputStreamReader`

Čítanie zo štandardného vstupu

```
class C {
    public static void main(String[] args) throws IOException {
        BufferedReader stdin =
            new BufferedReader(new InputStreamReader(System.in));

        System.out.print("Vstup: ");
        String s = stdin.readLine();
        System.out.println(s);
    }
}
```

Štandardný vstup a Scanner

- Java 5 API poskytuje textový skener pre načítavanie s rozborom (parsing)[1]
- Dá sa použiť na hocijaký vstupný prúd, reťazec a kanál
- Presnejšie zadanie formátu je možné pomocou triedy java.util.regex.Pattern
- Rozsiahlejšie možnosti na rozbor reťazcov poskytujú regulárne výrazy
  - Pozrite v TiJ pre potreby projektu

```
Scanner sc = new Scanner(System.in);
System.out.print("Ulica a cislo: ");
String ulica = sc.next();
int cislo = sc.nextInt();
System.out.println(ulica + " " + cislo);
```

[1] http://java.sun.com/j2se/1.5.0/docs/api/java/util/Scanner.html

Formatovaný výstup

- Java 5 API poskytuje aj formatovaný výstup
- Prúdy typu `PrintStream` poskytujú metódu printf()

```
double a = 10000.0;
System.out.printf("a = %e%n", a);
```

  - `%n` namiesto `\n` zaručuje správnu platformovo-špecifickú interpretáciu nového riadku
- Použitie je podobné ako v jazyku C
- Ak sa rozhodnete pre použitie formatovaného výstupu, preštudujte dokumentáciu triedy java.util.Formatter[2]

[2] http://java.sun.com/j2se/1.5.0/docs/api/java/util/Formatter.html

Čítanie z pamäte po znakoch

- Niekedy je vhodné pristupovať k reťazcu znakov ako k prúdu
- Na to je možné použiť triedu `StringReader`

```
class C {
    public static void main(String[] args) throws IOException {
        String s = new String("12356789");
        StringReader in = new StringReader(s);

        int c;
        while ((c = in.read()) != -1)
            System.out.print((char)c);
        in.close();
    }
}
```

Čítanie z pamäte podľa formátu údajov (1)

- Ak je potrebné z prúdu vyberať podľa formátu údajov, treba použiť triedu `DataInputStream`
- Potom možno vyberať bajty, znaky, čísla a pod.

Čítanie z pamäte podľa formátu údajov (2)

```
class C {
    public static void main(String[] args) throws IOException {
        String s = new String("123456789");
        DataInputStream in = null;
        try {
            in = new DataInputStream(new ByteArrayInputStream(s.getBytes()));

            while (true)
                System.out.print((char)in.readByte());
        } catch(EOFException e) {
            System.err.println("\nKoniec prúdu");
        } finally {
            in.close();
        }
    }
}
```

Výstup do súboru (1)

- Príklad vytvorenia textového súboru
- Pri súboroch treba zvlášť dbať na explicitne zatvorenie prúdu

```
class C {
    public static void main(String[] args) throws IOException {
        String s = new String("123\nabc");
        BufferedReader in = null;
        PrintWriter out = null;
        try {
            in = new BufferedReader(new StringReader(s));
            out = new PrintWriter(new BufferedWriter(new FileWriter("demo.out")));
            int l = 1;
            ...
```

Výstup do súboru (2)

```
            ...
            while ((s = in.readLine()) != null )
                out.println(l++ + ": " + s); // c. riadku: obsah
        } catch(EOFException e) {
            System.err.println("Koniec prúdu");
        } finally {
            in.close();
            out.close();
        }
    }
}
```

Ukladanie a obnovovanie údajov (1)

- Príklad vytvorenia súboru s binárnymi údajmi
- Nezávisle od platformy

```
class C {
    public static void main(String[] args) throws IOException {
        DataOutputStream out = null;
        DataInputStream in = null;
        try {
            out = new DataOutputStream(new BufferedOutputStream(
                new FileOutputStream("data.txt")));
            out.writeUTF("Text");
            out.writeDouble(1.1);
            ...
```

Ukladanie a obnovovanie údajov (2)

```
            ...
            in = new DataInputStream(new BufferedInputStream(
               new FileInputStream("data.txt")));
            System.out.println(in.readUTF());
            System.out.println(in.readDouble());
        } catch(EOFException e) {
            throw new RuntimeException();
        } finally {
            in.close();
            out.close();
        }
    }
}
```

Súbory s voľným prístupom (1)

- Random access – najčastejšie sa prekladá ako *priamy prístup*
- Trieda `RandomAccessFile` – pomerne izolovaná od zvyšku V/V systému
- Možnosť otvorenia pre čítanie a zápis súčasne

Súbory s voľným prístupom (2)

```
class C {
    public static void main(String[] args) throws IOException {
        RandomAccessFile f =
            new RandomAccessFile("test.dat", "rw");

        for(int i = 0; i < 10; i++)
            f.writeInt(i);

        f.seek(3*4); // nastavenie za tretí int
        f.writeInt(-1);

        f.seek(0); // nastavenie na začiatok
        for(int i = 0; i < 10; i++)
            System.out.println(i + ": " + f.readInt());
        f.close();
    }
}
```

Kanály a vyrovnávacie pamäte

- java.nio
- Flexibilnejšia a rýchlejšia práca so súbormi
- Kanály pre prácu so súbormi možno získať z tried FileInputStream, FileOutputStream a RandomAccessFile
- Vyrovnávacie pamäte sú založené na bajtoch – pri práci s reťazcami znakov treba použiť metódu getBytes() triedy String

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

class C {
   public static void main(String[] args) throws Exception {
      FileChannel fc = new FileOutputStream("data.txt").getChannel();
      fc.write(ByteBuffer.wrap("Some text ".getBytes()));
      fc.close();
   }
}
```

Súbory zobrazené do pamäte (1)

- Memory-mapped files
- Založené na kanáloch a vyrovnávacích pamätiach
- Novšia a lepšia alternatíva súborov s voľným prístupom

Súbory zobrazené do pamäte (2)

```
import java.io.*;
import java.nio.*;
import java.nio.channels.*;

class C {
   static int length = 0x6400000; // 100 MiB
   public static void main(String[] args) throws Exception {
      MappedByteBuffer out =
         new RandomAccessFile("test.dat", "rw").getChannel().
            map(FileChannel.MapMode.READ_WRITE, 0, length);

      for(int i = 0; i < length; i++)
         out.put((byte)'x');

      for(int i = length/2; i < length/2 + 6; i++)
         System.out.print((char)out.get(i));
   }
}
```

Kompresia

- Java V/V API obsahuje podporu kompresie (ZIP/GZIP)
- Založená na bajtovo orientovaných triedach `InputStream` a `OutputStream`
- Príklady v TiJ
- *Java Archive* (JAR)
  - Zoskupenia **class** súborov
  - Vytvárajú sa pomocou programu `jar`
  - **class** súbory v JAR archívoch sa dajú priamo používať
  - Pritom `classpath` musí zahŕňať príslušný JAR archív (ako adresár)

Serializácia objektov

Serializácia objektov

- Spôsob transformácie objektov do postupností bajtov
  - umožňuje ukladanie objektov do súborov a ich neskoršiu obnovu
  - umožňuje prenos objektov cez sieť a ich obnovu na strane prijímateľa
  - prenos argumentov pri volaní vzdialených metód (*remote method invocation*, RMI)

Serializácia objektov (2)

- Java API umožňuje serializáciu objektov – podpora perzistencie
- Aby sa dal serializovať, objekt musí implementovať rozhranie `Serializable`
  - Toto rozhranie neobsahuje metódy – slúži len na značenie
- Serializácia sa uskutočňuje pre celú spleť objektov, na ktoré sa daný objekt odkazuje
- Pre obnovenie objektov sú potrebné **class** súbory
- Atribúty (*fields*) sa ukladajú bez ohľadu na modifikátory prístupu
- `ObjectOutputStream` / `ObjectInputStream`

Príklad: serializácia objektov (1)

```
import java.io.*;

class Data implements Serializable {
    int n;
    Data next;
    public Data(int n) { this.n = n; }
}
```

Príklad: serializácia objektov (2)

```
class C {
   public static void main(String[] args) throws ClassNotFoundException, IOException {
      Data d1 = new Data(1);
      d1.next = new Data(2);

      ObjectOutputStream out =
         new ObjectOutputStream(new FileOutputStream("d.out"));
      out.writeObject(d1);
      out.close();

      ObjectInputStream in = new ObjectInputStream(new FileInputStream("d.out"));
      Data d2 = (Data)in.readObject();
      System.out.println(d2.n + ", next " + d2.next.n);
   } // 1, next 2
}
```

Riadenie serializácie

- Použitím rozhrania `Externalizable` namiesto `Serializable` objekt preberá zodpovednosť za serializáciu
  - Objekt musí implementovať serializáciu v metódach `writeExternal()` a `readExternal()`
- Iná možnosť je použitie rozhrania `Serializable` a metód `writeObject()` a `readObject()`
  - V rámci týchto metód objekt môže zavolať implicitnú serializáciu (`defaultWriteExternal()` a `defaultReadExternal()`
- Zabrániť uloženiu hodnôt pri jednotlivých atribútoch sa dá použitím kľúčového slova **transient** (teda *prechodný* – opak *persistent*)
- Podrobné príklady v TiJ

Serializácia zoskupení objektov

- Zoskupenia objektov sa tiež dajú serializovať
- Pole priamo predstavuje spleť objektov
- Zoskupenia objektov dosahujú serializovateľnosť implementáciou metódy `writeObject()` – pre programátora je to transparentné

```
class C {
   public static void main(String[] args) throws ClassNotFoundException, IOException {
      List<Data> l1 = new ArrayList<>(Arrays.asList(new Data(1), new Data(2)));

      ObjectOutputStream out =
         new ObjectOutputStream(new FileOutputStream("d.out"));
      out.writeObject(l1);
      out.close();

      ObjectInputStream in = new ObjectInputStream(new FileInputStream("d.out"));
      List<Data> l2 = (List<Data>)in.readObject();

      for (Data e : l2)
         System.out.println(e.n);
   }
```

Viacniťovosť

Niť

- Proces
- Niť – *thread*
  - bežne sa prekladá ako *vlákno*, ale potom sa stráca alúzia na niť v zmysle *niť príbehu/myšlienok*
- *Concurrent* (súbežné) / *distributed* (distribuované) / *parallel* (paralelné) *programming*
- *Multithreading* (viacniťovosť) – priama podpora v Jave
- Význam pre používateľské rozhranie – zrýchlenie odozvy

Vytváranie nití

- Dva spôsoby vytvárenia nití:
  - rozšírením triedy `Thread`
  - implementáciou rozhrania `Runnable` (na spustenie sa následne použije trieda `Thread`)

Niť rozšírením triedy Thread

```
class Nit extends Thread {
    int n = 0;
    public Nit(int n) { this.n = n; }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("T" + n + ": " + i);
        }
    }
    public static void main(String[] args) {
        for(int i = 0; i < 5; i++)
            new Nit(i).start();
    }
}
```

Niť implementáciou rozhrania Runnable

```
class Nit implements Runnable {
    int n = 0;
    public Nit(int n) { this.n = n; }
    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println("T" + n + ": " + i);
        }
    }
    public static void main(String[] args) {
        for(int i = 0; i < 5; i++)
            new Thread(new Nit(i)).start();
    }
}
```

Riadenie nití

- Metódy triedy `Thread`:
  - yield()
  - sleep()
  - interrupt()
  - wait()
  - join()
  - setPriority()
  - setDaemon()

Ďalšie súvislosti

- *Daemon thread*
  - Vykonávanie v pozadí
  - JVM ukončí program ak všetky nite, ktoré bežia sú typu *daemon*
- Kľúčové slovo **volatile** – zabezpečenie atomickosti
  - Aj samotné čítanie a zápis premennej prebiehajú v niekoľkých krokoch
  - Atribúty deklarované ako **volatile** zaručujú atomicky prístup pri čítaní a zápise
- Synchronizácia
- Spolupráca nití (*pipes* – rúry)
- *Deadlock* – uviaznutie

Synchronizácia nití

- K prepnutiu medzi niťami môže dôjsť aj pri prístupe k zdieľaným zdrojom
- Tomu sa dá zabrániť tzv. synchronizáciou nití
- Podstata je v uzamknutí objektu pre kritický región (critical section) programu
- Zabezpečuje sa kľúčovým slovom **synchronized** pred synchronizovaným blokom (kritickým regiónom) s uvedením referencie objektu, ktorý má byť uzamknutý:

```
synchronized (referenciaObjektu) {
    . . . // kritický región
}
```

Synchronizované metódy (1)

- Synchronizovaná môže byť aj celá metóda

```
class Trieda1 {
    synchronized void op1() {
        ...
    }
    static synchronized void op2() {
        ...
    }
}
```

- Pri metóde inštancie sa uzamyká aktuálny objekt, t.j. **this**

```
void op1() {
    synchronized (this) {
        ...
    }
}
```

Synchronizované metódy (2)

- Pri statických metódach sa uzamyká objekt triedy – dostupný prostredníctvom metódy `getClass()` príslušnej triedy

```
static void op2() {
    synchronized (getClass()) {
        ...
    }
}
```

Príklad: synchronizácia nití (1)

```
class Nit1 extends Thread {
    public void run() {
        for (int i = 0; i < Integer.MAX_VALUE; i++) {
            Test.increment();
        }
    }
}

class Nit2 extends Thread {
    public void run() {
        for (int i = 0; i < Integer.MAX_VALUE; i++) {
            Test.print();
        }
    }
}
```

Príklad: synchronizácia nití (2)

```
class Test {
    static int i = 0, j = 0;
    static void increment() {
        i++;
        j++;
    }
    static void print() {
        if (i != j)
            System.out.println("i=" + i + " j=" + j);
    }
    public static void main(String[] args) {
        new Nit1().start();
        new Nit2().start();
    }
}
```

- Nie je isté, že sa i a j budú inkrementovať zároveň
- Občas sa teda uskutoční výpis v metóde print()

Príklad: synchronizácia nití (3)

```
class Test {
    static int i = 0, j = 0;
    static synchronized void increment() {
        i++;
        j++;
    }
    static synchronized void print() {
        if (i != j)
            System.out.println("i=" + i + " j=" + j);
    }
    public static void main(String[] args) {
        new Nit1().start();
        new Nit2().start();
    }
}
```

- Synchronizácia zaručuje, že sa inkrementácia `i` a `j` vykoná bez prerušenia
- Obidvé metódy musia byť synchronizované

Sumarizácia

Sumarizácia

Sumarizácia

- V/V systém Javy:
  - starý (`java.io`) – prúdy údajov
  - nový (`java.nio`) – kanály a vyrovnávacie pamäte
- Serializácia objektov (pomocou `java.io`) – podpora perzistencie
- Viacniťovosť
  - Dva spôsoby tvorby nití
  - Riadenie nití
  - Synchronizácia nití

Čítanie

- Dnešná prednáška: OJA, kapitoly 11 a 12
  - Dobrý prehľad vrátane vecí týkajúcich sa Javy 5 je na `http://java.sun.com/docs/books/tutorial/essential/io/`
  - TiJ, kapitola 12 podľa potrieb projektu – napríklad Preferences (uchovanie nastavení) alebo Regular expressions
- Na ďalšiu prednášku: OJA, kapitola 13
  - Užitočné príklady v TiJ, kapitola 14 – grafický rámec Swing možno využijete v projekte