Soubory

- Soubor je množina údajů uložená ve vnější paměti počítače, obvykle na disku
- Pro soubor jsou typické tyto operace.
  - otevření souboru
  - čtení údaje
  - zápis údaje
  - uzavření souboru
- Přístup k údajům (čtení nebo zápis) může být:
  - sekvenční
  - nahodilý
- Soubory se sekvenčním přístupem (sekvenční soubory) umožňují pouze postupné (sekvenční) čtení nebo zápis údajů
- Soubory s nahodilým přístupem umožňují nahodilé čtení nebo zápis udaje (podobně jako pole)
- Způsob přístupu k údajům v souboru není zakódován v souboru, ale je dán programem
- Zde se budeme zabývat pouze sekvenčními soubory

Soubor jako posloupnost bytů

- V jazyku Java slouží pro práci se soubory třídy definované v balíku *java.io*
- Soubor jako posloupnost bytů reprezentují třídy:
  - *FileInputStream* vstupní soubor (bude se z něj číst)
  - *FileOutputStream* výstupní soubor (bude se do něj zapisovat)
- Příklad: kopie souboru

```
package alg8;
import java.io.*;

public class Kopie1 {
  public static void main(String[] args) throws IOException {
    FileInputStream in = new FileInputStream("vstup.txt");
    FileOutputStream out = new FileOutputStream("vystup.txt");
    int b = in.read();
    while (b!=-1) {
      out.write(b);
      b = in.read();
    }
    out.close();
    in.close();
  }
}
```

Soubor jako posloupnost bytů

- Komentář k příkazům

```
FileInputStream in = new FileInputStream("vstup.txt");
```

  – vytvořený objekt *in* reprezentuje vstupní soubor uložený na disku v aktuálním adresáři pod názvem *vstup.txt*; pokud takový soubor neexistuje, nastane chyba

```
FileOutputStream out = new FileOutputStream("vystup.txt");
```

  – vytvořený objekt *out* reprezentuje výstupní soubor, který bude uložen do aktuálního adresáře pod názvem *vystup.txt*; pokud by soubor nebylo možné vytvořit, nastane chyba

```
b = in.read();
```

  – ze souboru *in* se přečte jeden byte (číslo v rozsahu 0..255) a uloží do *b*; není-li v souboru žádný nepřečtený byte, výsledkem metody je –1

```
out.write(b);
```

  – do souboru *out* se zapíše jeden byte s hodnotou *b*

```
out.close();
in.close();
```

  – uzavření souborů *in* a *out*

Výjimky

- Při operacích se soubory (ale i při jiných operacích) mohou nastat chyby
- Chyba při provádění programu v jazyku Java nemusí znamenat ukončení programu – chybu lze ošetřit a pokračovat dál
- K ošetření chyb slouží mechanismus výjimek (exceptions)
- Princip výjimek v jazyku Java:
  - libovolná metoda (konstruktor, funkce) může skončit standardně (proběhla-li operace bez chyby) nebo nestandardně vyhozením (vyvoláním) výjimky určitého typu (v případě, že nastala chyba)
  - pokud příkaz skončil vyhozením výjimky, další příkazy se neprovedou a řízení se předá konstrukci ošetřující výjimku daného typu (s touto konstrukcí se seznámíme později)
  - pokud taková konstrukce v těle funkce (metody, konstruktoru) není, skončí funkce nestandardně a výjimka se šíří na dynamicky nadřazenou úroveň
  - není-li výjimka ošetřena ani ve funkci *main*, vypíše se a program skončí
  - pro rozlišení různých typů výjimek je v jazyku Java zavedena řada knihovních tříd, výjimky jsou instancemi těchto tříd

Neošetření výjimek

- Příklady operací, které mohou skončit vyhozením výjimky:
  `FileInputStream in = new FileInputStream(“vstup.txt”);`
  – pokud soubor neexistuje, vyhodí se výjimka typu *FileNotFoundException*
  `FileOutpuyStream out = new FileOtputStream(...);`
  – pokud soubor nelze vytvořit (např. je zadaná špatná cesta), vyhodí se výjimka *FileNotFoundException*
  `b = in.read();`
  `out.write(b);`
  – nepodaří-li se data přečíst nebo zapsat, vyhodí se výjimka *IOExcpetion*
- Jestliže funkce obsahuje deklarace a příkazy, které mohou vyhodit výjimky a tyto výjimky nejsou ve funkci ošetřeny, musí být seznam typů výjimek, které se mohou z funkce šířit, uveden v hlavičce funkce

```
public static void main(String[] args)
  throws FileNotFoundException, IOException { ... }
```

- Výjimka *FileNotFoundException* je speciálním druhem (podtřídou) výjimky *IOException* – předchozí hlavičku můžeme zkrátit:

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

- Všechny třídy výjimek jsou podtřídami třídy *Exception*

Ošetření výjimek

- Chceme-li ošetřit výjimku, která může vzniknout při provádění určité operace, je třeba operaci vložit do bloku *try,* ke kterému je připojen blok (klauzule) *catch* ošetřující (zachytávající) výjimky daného typu
- Příklad: funkce, která si vyžádá zadání jména vstupního souboru z klávesnice a pokud soubor s daným jménem neexistuje, proces se opakuje (při zadání prázdného jména program skončí)


```
static FileInputStream vstup() {
   for (;;) {
     Sys.p("zadejte vstupní soubor: ");
     String jmeno = Sys.readLine();
     if (jmeno.equals("")) System.exit(0);
     try {
       FileInputStream in = new FileInputStream(jmeno);
       return in;
     } catch (FileNotFoundException e) {
       Sys.pln("soubor neexistuje");
     }
   }
 }
```

Ošetření výjimek

- Pokračování příkladu: napíšeme podobnou funkci pro zadání výstupního souboru a obě funkce použijeme ve druhé verzi programu pro kopii souboru (třída Kopie2)

```
static FileOutputStream vystup() {
  for (;;) {
    Sys.p("zadejte výstupní soubor: ");
    String jmeno = Sys.readLine();
    if (jmeno.equals("")) System.exit(0);
    try {
      FileOutputStream in = new FileOutputStream(jmeno);
      return in;
    } catch (FileNotFoundException e) {
      Sys.pln("soubor nelze vytvořit");
    }
  }
}
```

Ošetření výjimek

- V hlavní funkci *main* ošetříme výjimku typu *IOException*, která může vzniknout při provádění metod *read, write* a *close*

```
public static void main(String[] args) {
   FileInputStream in = vstup();
   FileOutputStream out = vystup();
   try {
     int b = in.read();
     while (b!=-1) {
       out.write(b);
       b = in.read();
     }
     in.close();
     out.close();
   } catch (IOException e) {
     e.printStackTrace();
   }
 }
```

Argumenty příkazového řádku

- Provedení programu v jazyku Java se zadá příkazovým řádkem
  `java Program`
- Příkazový řádek může obsahovat další parametry (argumenty)
  `java Program arg0 arg1 ...`
- Argumenty příkazového řádku jsou přístupné hlavní funkcí *main* prostřednictvím jejího parametru, kterým je reference pole řetězců
  `public static main(String[] args)`
- Příklad: je-li provedení programu zadáno příkazovým řádkem
  `java Program prvni druhy treti`

  pak v hlavní funkci *main* třídy *Program* platí:
  - hodnotou *args*[0] je řetězec “prvni”
  - hodnotou *args*[1] je řetězec “druhy”
  - hodnotou *args*[2] je řetězec “treti”
  - hodnotou *args.length* je 3

Zadání souborů příkazovým řádkem

```
public class Kopie3 {
  public static void main(String[] args) {
    if (args.length<2) {
      Sys.pln("použití: Kopie3 <vstup> <výstup>");
      return;
    }
    try {
      FileInputStream in = new FileInputStream(args[0]);
      FileOutputStream out = new FileOutputStream(args[1]);
      int b = in.read();
      while (b!=-1) {
        out.write(b);
        b = in.read();
      }
      out.close();
      in.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Další verze kopie souboru

```
public class Kopie4 {
  public static void main(String[] args) {
    if (args.length<2) {
      System.out.println("použití: Kopie4 <vstup> <výstup>");
      return;
    }
    try {
      FileInputStream in = new FileInputStream(args[0]);
      FileOutputStream out = new FileOutputStream(args[1]);
      while (in.available()>0) {
        out.write(in.read());
      }
      out.close();
      in.close();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```


- Poznámka: metoda *available* vrací počet nepřečtených bytů souboru

Soubor jako posloupnost primitivních typů

- Soubor typu *FileOutputStream* je z hlediska operací posloupnost prvků typu *byte*
- Soubor jako posloupnost primitivních typů reprezentují třídy *DataOutputStream* a *DataInputStream*
- Příklad: program, který vytvoří soubor obsahující 100 náhodných čísel typu *double* a pak soubor přečte a vypíše součet čísel

```
public class Cisla {
  public static void main(String[] args)
                      throws Exception {
    DataOutputStream out = new DataOutputStream(
                    new FileOutputStream("temp.bin"));
    for (int i=0; i<100; i++)
      out.writeDouble(Math.random());
    out.close();
    DataInputStream in = new DataInputStream(
                    new FileInputStream("temp.bin"));
    double soucet = 0;
    while (in.available()>0)
      soucet = soucet + in.readDouble();
    Sys.pln(soucet);
  }
}
```

Operace se souborem primitivních typů

- DataOutputStream
  `void write`*`Typ`*`(`*`Typ`*` x) throws IOException`
  - do souboru zapíše hodnotu *x* primitivního typu *Typ*
  - příklad:

```
void writeInt(int x) throws IOException
void writeDouble(double x) throws IOException
...
```

- DataInputStream
  ***`Typ`*` `**`readTyp() throws EOFException, IOException`**
  - ze souboru přečte hodnotu primitivního typu *Typ* a vrátí ji jako výsledek
  - výjimka *EOFException* vznikne tehdy, když v souboru již není dostatečný počet bytů tvořících hodnotu daného typu
  - příklad:

```
int readInt() throws EOFException, IOException
double readDouble()
                        throws EOFException, IOException
...
```

- Při čtení souboru je třeba dodržet stejné pořadí typů, jaké bylo při vytváření souboru

Soubor primitivních typů a objektů

- Soubor obsahující jak primitivní typy tak objekty reprezentují třídy *ObjectOtputStream* a *ObjectInputStream*
- Příklad: program, který vytvoří soubor obsahující dvě hodnoty typu *int* a dva řetězce, a pak soubor přečte a vypíše

```
public class CislaRetezce {
  public static void main(String[] args) throws Exception {
    ObjectOutputStream out = new ObjectOutputStream(
                        new FileOutputStream("temp.bin"));
    out.writeInt(1);
    out.writeInt(2);
    out.writeObject("prvni retez");
    out.writeObject("druhy retez");
    out.close();
    ObjectInputStream in = new ObjectInputStream(
                             new FileInputStream("temp.bin"));
    Sys.pln(in.readInt()+" "+in.readInt());
    String s1 = (String)in.readObject();
    String s2 = (String)in.readObject();
    Sys.pln(s1+" "+s2);
  }
}
```

Operace se souborem primitivních typů a objektů

- Pro zápis hodnot primitivních typů do souboru typu *ObjectOutputStream* slouží stejné metody, jako pro zápis do souboru typu *DataOutputStream*
- Pro čtení hodnot primitivních typů ze souboru typu *ObjectInputStream* slouží stejné metody, pro pro čtení ze souboru *DataInputStream*
- Zápis objektu do souboru *ObjectOutputStream*:

```
void writeObject(Object o)
                    throws IOException a další
```

- Připomenutí: hodnotou parametru (proměnné) typu *Object* může být reference objektu libovolné třídy
- Čtení objektu ze souboru *ObjectInputStream*:

```
Object readObject()
                    throws IOException a další
```

  – obvykle je třeba znát typ přečteného objektu a vrácenou referenci přetypovat
- Při čtení souboru je třeba dodržet stejné pořadí typů, jaké bylo při vytváření souboru
- Uvedenými metodami lze zapisovat a číst pouze tzv. serializovatelné objekty (patří mezi ně např. řetězce a pole primitivních typů)