



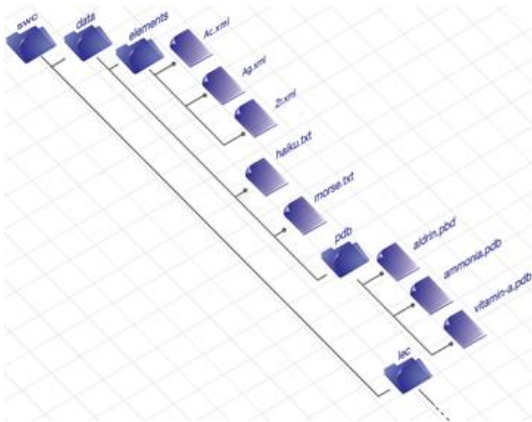
# 7. prednáška (4.11.2010)

```
Exception in thread "main" java.lang.NullPointerException  
    at Vynimkarka.kladnyPriemer(Vynimkarka.java:9)  
    at Spustac.main(Spustac.java:10)
```

## Výnimky (1.časť), adresáre a súbory

alebo

Pomaly opúšťame  
korytnačky





# Čo sú to výnimky?

- Výnimky sú špeciálne objekty, ktoré vznikajú vo výnimočných stavoch, keď nejaké metódy nemôžu prebehnúť štandardným spôsobom alebo nevedia vrátiť očakávanú hodnotu
- Jedna z najlepších vecí v OOP
- Už žiadne “segmentation fault“, či “Fatal error“ známe z neobjektových jazykov



# Testovacia metóda

- Vytvorme si nasledujúcu metódu, ktorá má:
- vrátiť **true** ak je priemer prvých  $k$  prvkov poľa pole väčší ako 0,
- inak má vrátiť **false**.

```
public boolean kladnyPriemer(int[] pole, int k)
```



# Testovacia metóda

- Testujeme odolnosť metódy na zákerné vstupy:
- V premennej `pole` pošleme **null**
- Pošleme `pole` dĺžky nula
- Pošleme hodnotu pre `k` väčšiu ako posledný index `poľa`
- Všetky tieto vstupy “vyhodia” nejakú výminku



# Stack trace

- Keď sa vyhodí výnimka môžeme obdivovať stack trace:

Názov vyhodenej výnimky

```
Exception in thread "main" java.lang.NullPointerException  
at Vynimkarka.kladnyPriemer(Vynimkarka.java:9)  
at Spustac.main(Spustac.java:10)
```

kladnyPriemer bol volaný  
z metódy main v triede  
Spustac z 10. riadku

Program skončil vykonávanie na 9. riadku  
v triede Vynimkarka a bolo to v metóde  
kladnyPriemer



# *java.lang.NullPointerException*

- Snád' najčastejšia výnimka.
- Typické situácie:

```
private Turtle t;  
t.step(100);
```

`null.step(100)`

```
private int[] pole;  
for(int i=0; i < pole.length; i++)
```

`null.length`

```
Turtle[] korytnacky = new Turtle[10];  
korytnacky[0].turn(90);
```

`null.turn(90)`



# Ďalšie výnimky

- `java.lang.ArithmeticException`: / by zero
  - Delili sme nulou
- `java.lang.NegativeArraySizeException`
  - `int[] pole = new int[-5];`
- `java.lang.ArrayIndexOutOfBoundsException`: 10
  - Použili sme index poľa 10, čo je mimo rozsahu poľa, ktoré malo veľkosť 10 alebo menej.
- Všetky tieto výnimky sa dajú ošetriť `if`-mi.



# Sčítavame čísla zo Stringu

- Chceme sčítať všetky čísla v danom reťazci

"125 26 1587 11 0 15"

- Potrebujeme :
  - Rozdeliť vstup na slová
  - Každé slovo parsovať na čísla

```
int cislo = Integer.parseInt("...");
```





# Sčítavame čísla zo Stringu

- Čo ak sa mi stane niečo z nasledovného ?

```
int cislo = Integer.parseInt("Java");
```

```
int cislo2 = Integer.parseInt("");
```

- Vyletí `NumberFormatException`
- Prostý `if` by nepomohol



# Odchytávame výnimky

- Výnimky vieme odchytiť
- Môžeme sa vysporiadať s daným stavom a zabrániť ukončeniu aplikácie

```
try {  
    // blok príkazov z ktorého odchytávame výnimky  
} catch (TypVýnimky1 e) {  
    // vysporiadanie sa s daným typom výnimky  
} catch (TypVýnimky2 e) {  
    // vysporiadanie sa s daným typom výnimky  
}
```



# Situácia 1

```
príkaz1;  
try {  
    príkaz2;  
    príkaz3;  
}  
catch (TypVýnimky1 e) {  
    príkazE1;  
}  
catch (TypVýnimky2 e) {  
    príkazE2;  
}  
príkaz4;
```

V prípade normálneho priebehu sa vykonajú:

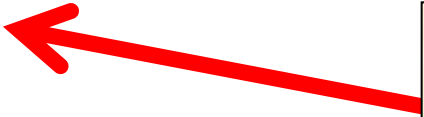
```
príkaz1;  
príkaz2;  
príkaz3;  
príkaz4;
```



## Situácia 2

⚡ príkaz1;

```
try {  
    príkaz2;  
    príkaz3;  
} catch (TypVýnimky1 e) {  
    príkazE1;  
} catch (TypVýnimky2 e) {  
    príkazE2;  
}  
príkaz4;
```



Nech je výnimka akákoľvek,  
končíme, nič viac sa nevykoná a  
obdivujeme stack trace



# Situácia 3

```
príkaz1;
```

```
try {
```

```
⚡ príkaz2;
```

```
    príkaz3;
```

```
} catch (TypVýnimky1 e) {
```

```
    príkazE1;
```

```
} catch (TypVýnimky2 e) {
```

```
    príkazE2;
```

```
}
```

```
príkaz4;
```

Ak sa vyhodí výnimka typu  
TypVýnimky1 vykonáme  
príkazE1 a príkaz4



# Situácia 4

```
príkaz1;
```

```
try {
```

```
⚡ príkaz2;
```

```
    príkaz3;
```

```
} catch (TypVýnimky1 e) {
```

```
    príkazE1;
```

```
} catch (TypVýnimky2 e) {
```

```
    príkazE2;
```

```
}
```

```
príkaz4;
```

Ak sa vyhodí výnimka typu  
TypVýnimky2 vykonáme  
príkazE2 a príkaz4



# Situácia 5

```
príkaz1;
```

```
try {
```

```
⚡ príkaz2;
```

```
    príkaz3;
```

```
} catch (TypVýnimky1 e) {
```

```
    príkazE1;
```

```
} catch (TypVýnimky2 e) {
```

```
    príkazE2;
```

```
}
```

```
príkaz4;
```

Ak sa vyhodí výnimka rôzna od  
TypVýnimky1 aj TypVýnimky2  
končíme a obdivujeme stack  
trace



# Blok finally

- V bloku **finally** môžeme napísať príkazy, ktoré sa vykonajú vždy, ak už program vošiel do bloku **try** - bez ohľadu na to, či v bloku **try** výnimka nastala alebo nenastala, bez ohľadu na to, či sme ju odchytili alebo nie

```
try {  
    // blok príkazov z ktorého odchyťavame výnimky  
} catch (TypVýnimky1 e) {  
    // vysporiadanie sa s daným typom výnimky  
} catch (TypVýnimky2 e) {  
    // vysporiadanie sa s daným typom výnimky  
} finally {  
    // príkazy, ktoré sa vykonajú bez ohľadu na to, čo sa stalo  
}
```





# *Blok finally*

- Ak v bloku **try** realizujeme nejakú komunikáciu s okolím, často chceme korektne uzavrieť komunikáciu bez ohľadu na to, či sme riešili alebo neriešili výnimočné stavy
- Typické použitie bloku **finally**:
  - Na uzavretie súboru
  - Na ukončenie sieťového pripojenia
  - Na ukončenie pripojenia na databázu
  - Na zápis vykonanej operácie do logovacieho súboru



# Slajd pre špekulantov

- Ak nastane výnimka v bloku **catch** alebo v bloku **finally** obdivujeme stack trace
- Môžeme to riešiť vnorením ďalších **try-catch** blokov






# Nie sú výnimky ako výnimky

- V Java existujú dva typy výnimiek
- **RuntimeException** - behové (neočakávané) výnimky
  - Nemusíme ich odchytať ak nechceme
  - `NullPointerException`, `ArrayIndexOutOfBoundsException`, `ArithmeticException`, `NumberFormatException`, ...
  - Obvykle spôsobené programátorom, ťažko sa z nich zotaviť bez zmeny kódu (napr. vhodným ošetrením if-mi)
- **Exception** - očakávané výnimky
  - Musíme ich odchytať (inak Eclipse protestuje)
  - `FileNotFoundException`, ...
  - Obvykle spôsobené používateľom (zlý vstup), zmenou kódu im nevieme predchádzať, vieme ale poprosiť používateľa o nový vstup
- Na fórach sú búrlivé diskusie o tom, ktorá výnimka má byť akého druhu



# Sumár

- Čo sa dá ošetriť `if`-mi, ošetrujeme `if`-mi !
- Ak nastane výnimka mimo `try` bloku,
  - program končí a pozeráme na stack trace 
- Ak nastane výnimka v `try` bloku a neodchytíme ju v `catch` bloku,
  - program skočí do `finally` bloku, ten sa vykoná
  - a pozeráme na stack trace 
- Ak nastane výnimka v `try` bloku a odchytíme ju v `catch` bloku,
  - program skočí do príslušného `catch` bloku,
  - potom do `finally` bloku
  - a potom pokračujeme v programe ďalej 



# Vstupno-výstupné operácie

- Programy potrebujú komunikovať s okolím
  - získavať z neho údaje
  - odovzdávať/zobrazovať mu údaje
- Potrebujú vstupy a výstupy
  - **vstup:** klávesnica, súbor, myš, internet, databáza, ...
  - **výstup:** monitor, súbor, internet, tlačiareň, databáza, ...



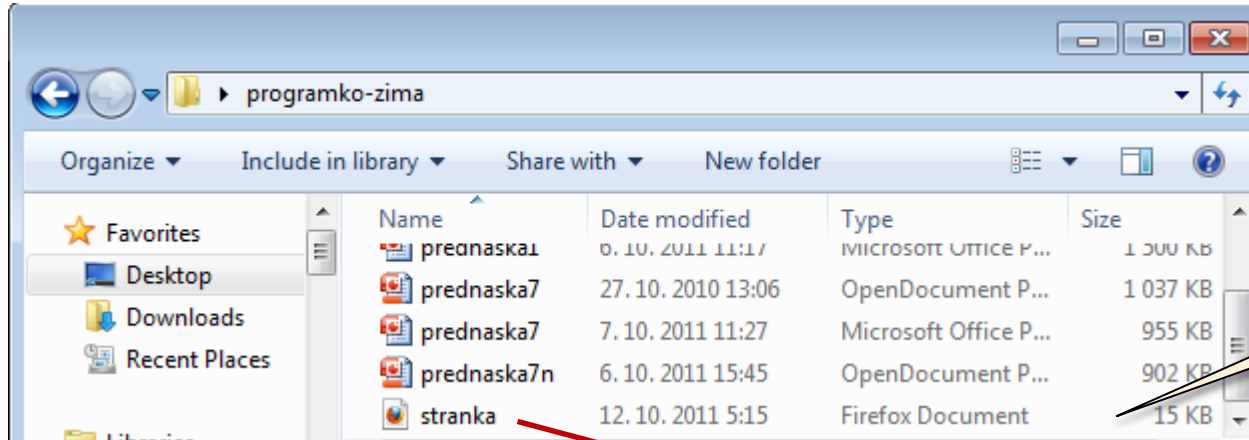


# Adresáre a súbory

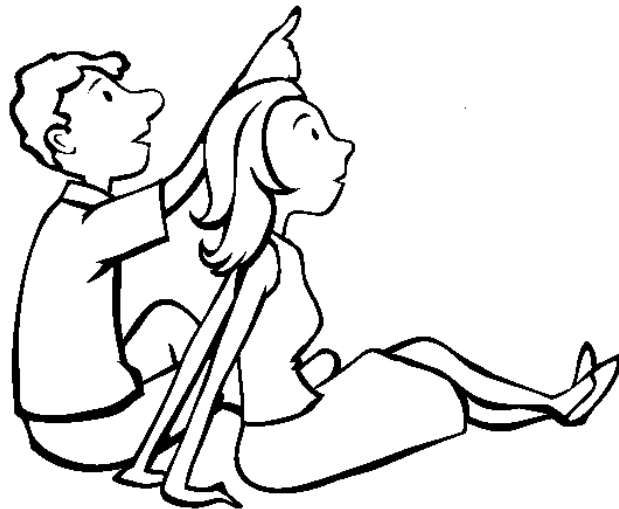
- Adresáre a súbory sú dôverne známe z operačných systémov
- V Jave (aj Linuxe)
  - súbor a adresár splývajú do jedného pojmu
  - adresár je tiež súbor
- Súbor to je:
  - **Dáta** (pohľad z NotePadu): postupnosť 0 a 1 tvoriaci obsah súboru
  - **Metadáta** (pohľad z Total Commandera): názov, veľkosť, umiestnenie, oprávnenia, vlastník, ...



# Pohľad na súbor



metadáta



dáta

```
<?xml
version="1.0"
encoding="utf-8"?>
<!DOCTYPE html
PUBLIC "-//
//W3C//DTD
XHTML 1.0
Transitional//EN"
...
```



# Adresáre a súbory

- Adresáre tvoria stromovú hierarchiu
- Vo Windowse:
  - Úplný názov súboru:  
`C:\Windows\system32\shell32.dll`
  - Cesta k súboru: `C:\Windows\system32`
  - Názov súboru: `shell32.dll`
- V Linuxe
  - Úplný názov súboru: `/home/gursky/textovy.subor`
  - Cesta k súboru: `/home/gursky`
  - Názov súboru: `textovy.subor`





# Cesta k súborom

- Vo Windowse:

- Položky sú oddelené spätnou lomkou \, ale je možné používať aj /, len o tom málokto vie
- **POZOR, častá chyba:** ak chcete používať spätné lomky, v reťazcoch ich musíte zdvojiť ( \ je špeciálny znak)

“C:\\Windows\\system32”

- V Linuxe

- Položky sú oddelené lomkou /



# Cesta k súborom

- Absolútna cesta: **D:\projekty\paz\obrazok.png**
- Relatívna cesta: vzhľadom k nejakému adresáru
  - **..\paz\obrazok.png** je relatívna vzhľadom k **D:\projekty\programko**



# Aktuálny adresár

- Relatívna cesta môže byť aj k aktuálnemu adresáru
- Ak spúšťame program z Eclipse, aktuálny adresár je adresár projektu
- Ak spúšťame program z príkazového riadku, aktuálny adresár je adresár z ktorého spúšťame program
- Pre fajšmekrov:
  - Aktuálny adresár sa dá získať cez:

```
String menoAktAdresara=System.getProperty("user.dir");
```



# *java.io.File*

- Trieda na prácu s metadátaami o súboroch alebo adresároch
- Objekty triedy `File` sú definované cestou k súboru alebo adresáru
- Tento súbor alebo adresár **nemusí reálne existovať** !
- Analógia:

Jožko Turtlák  
Javovská 42  
040 01 Košice

Taká adresa neexistuje,  
ale možno časom bude a  
možno na nej bude aj bývať  
Jožko Turtlák



# *java.io.File*

// úplná cesta k adresáru s použitím spätných lomiek

```
File adresar = new File("C:\\Windows\\System32");
```

// úplná cesta k súboru s použitím obyčajných lomiek

```
File subor1 = new File("C:/Windows/system.ini");
```

// relatívna cesta k súboru C:\Windows\System32\shell32.dll  
vzhľadom k adresáru C:\Windows\System32

```
File subor2 = new File(adresar, "shell32.dll");
```

// relatívna cesta k súboru vzhľadom k aktuálnemu adresáru

```
File subor3 = new File("heslo.txt");
```



# Niektoré užitočné metódy

<code>String getPath()</code>	Vráti úplný názov súboru
<code>String getName()</code>	Vráti názov súboru alebo adresára (bez cesty)
<b>boolean</b> <code>exists()</code>	Vráti <b>true</b> , ak súbor/adresár existuje
<b>boolean</b> <code>isDirectory()</code>	Zistí, či inštancia zodpovedá adresáru
<b>boolean</b> <code>isFile()</code>	Zistí, či inštancia zodpovedá súboru
<b>long</b> <code>length()</code>	Vráti veľkosť súboru
<b>void</b> <code>createNewFile()</code>	Vytvorí súbor, ak neexistuje. Môže vyvolať výnimku <code>IOException</code> ak nemáme dostatočné práva, alebo neexistuje adresár, v ktorom by sa mal tento súbor vytvoriť.



# Niektoré užitočné metódy

```
void mkdir()
```

Vytvorí adresár, zodpovedajúci poslednej položke v ceste, nadadresár musí existovať.

```
void mkdirs()
```

Vytvorí celú adresárovú štruktúru v ceste.

```
void renameTo(File)
```

Premenuje súbor podľa inej inštancie triedy `File`.

```
void delete()
```

Odstráni súbor.

```
String[] list()
```

Vráti pole názvov súborov/podadresárov v adresári. Vráti **null**, ak nejde o existujúci adresár.

```
File[] listFiles();
```

Vráti pole inštancií triedy `File` zodpovedajúcich súborom/podadresárom v adresári. Vráti **null** ak nejde o existujúci adresár.



# *Testujeme File*

- Vytvorme si metódu, ktorá nám vypíše všetky súbory a adresáre v danom adresári
- Metódu, ktorá vypíše mená a veľkosti všetkých mp3-ák v danom adresári





# Práca s textovými súbormi

- Samotná trieda `File` nám neumožňuje pracovať s dátami v súbore
- Práca s **obsahom textových súborov** sa vždy skladá z 3 krokov.
  - otvorenie súboru, ktoré sa udeje pri vzniku nejakého čítača (napr. `Scanner`) alebo zapisovača (napr. `PrintWriter`).
  - práca s obsahom súboru, teda čítanie alebo zapisovanie
  - zatvorenie súboru



# Zápis do textového súboru

- Najjednoduchší zápis do súboru je cez objekty triedy `java.io.PrintWriter`
- Pri otvorení sa súbor premaže
- Na zapisovanie používame metódy `print(...)` a `println(...)`, ktoré fungujú rovnako ako v prípade `System.out.print(...)` a `System.out.println(...)` len sa nevypisuje do konzoly, ale do súboru.
- Prácu s `PrintWriter`-om budeme **vždy** končiť metódou `close()`



# Zápis do textového súboru

- Prácu s textovým súborom budeme vždy realizovať v rámci **try-catch** bloku
- **Musíme** odchytať možnú výnimku `FileNotFoundException`
  - Vyhodí sa, keď adresár v ktorom má nový súbor vzniknúť neexistuje (spomeňte si, že objekty triedy `File` nemuseli pracovať s existujúcim súborom alebo adresárom)
  - Vyhodí sa, aj keď existuje adresár s týmto menom
- V bloku **finally** zatvárame súbor !



# Schéma práce s *PrintWriter*-om

```
File subor = new File("C:\\adresare\\subor");
PrintWriter pw = null;
try {
    pw = new PrintWriter(subor);

    // píšeme do pw

} catch (FileNotFoundException e) {
    System.out.println("Súbor " +
        subor.getName() + " som nenašiel");
} finally {
    if (pw != null)
        pw.close();
}
```



## *Zapisujeme do súboru*

- Vytvorme si metódu, ktorá vypíše do zadaného súboru v prvom riadku veľkosť poľa čísiel a v druhom riadku obsah poľa čísiel.



# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
    }
}
```



náš program

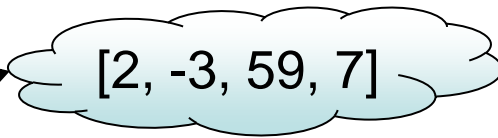
objekty

reálny svet

pole



[2, -3, 59, 7]



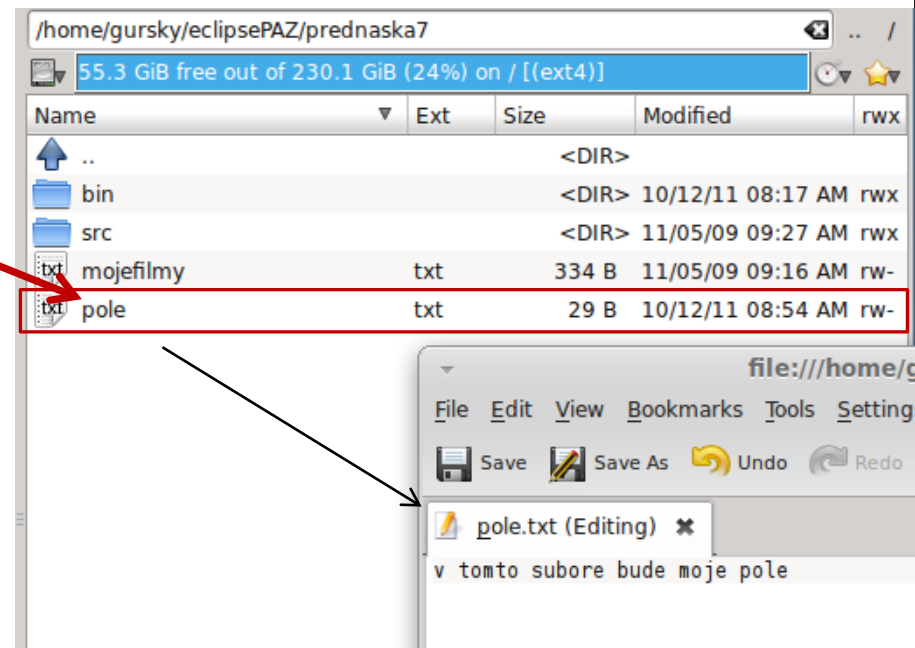
subor



pw



null



# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
```



naš program

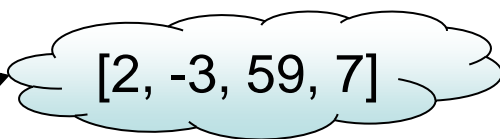
objekty

reálny svet

pole



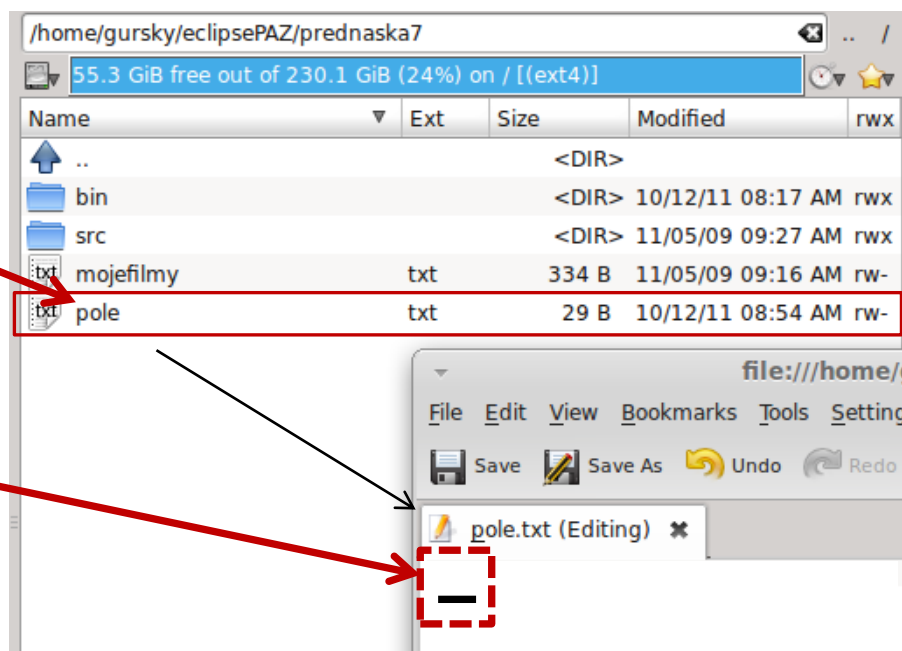
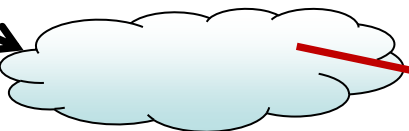
[2, -3, 59, 7]



subor



pw





# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
        pw.println(pole.length);
        for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
        }
    }
```



náš program

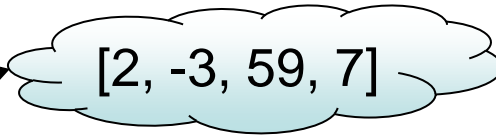
objekty

reálny svet

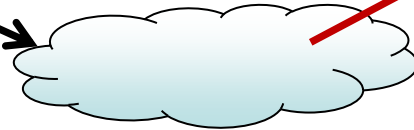
pole



[2, -3, 59, 7]



pw







# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
        pw.println(pole.length);
        for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
        }
    }
}
```



naš program

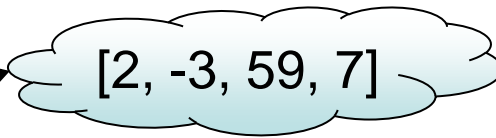
objekty

reálny svet

pole



[2, -3, 59, 7]



pw



4





# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
        pw.println(pole.length);
        for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
        }
    }
```



naš program

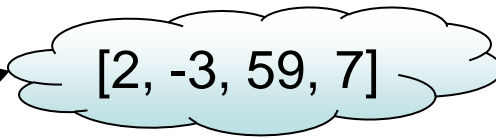
objekty

reálny svet

pole



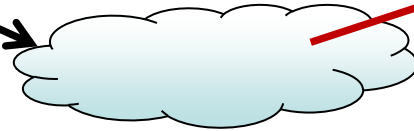
[2, -3, 59, 7]



pw



i



4

2





# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
        pw.println(pole.length);
        for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
        }
    }
```



naš program

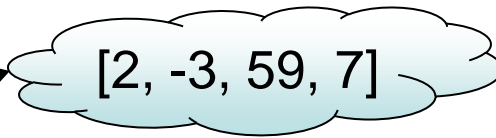
objekty

reálny svet

pole



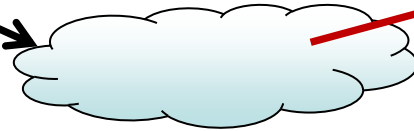
[2, -3, 59, 7]



pw



i



4

2 -3





# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {
    PrintWriter pw = null;
    try {
        pw = new PrintWriter(subor);
        pw.println(pole.length);
        for (int i = 0; i < pole.length; i++) {
            pw.print(pole[i]+" ");
        }
    }
```



naš program

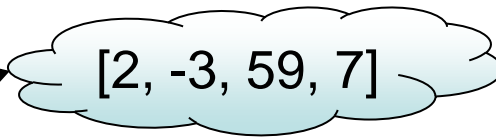
objekty

reálny svet

pole



[2, -3, 59, 7]



pw



i



4

2 -3 59 7 -





# Zapisujeme do súboru

```
public void zapisDoSuboruPole(File subor, int[] pole) {  
    PrintWriter pw = null;  
    try {  
        pw = new PrintWriter(subor);  
        pw.println(pole.length);  
        for (int i = 0; i < pole.length; i++) {  
            pw.print(pole[i]+" ");  
        }  
    } catch (FileNotFoundException e) {  
        System.err.println("Súbor " + subor.getName() + " sa nenašiel");  
    } finally {  
        if (pw!=null)  
            pw.close();  
    }  
}
```



# *java.util.Scanner*

- Dokáže čítať textové vstupy:

- z textového súboru

- `File subor = new File("D:\\vstup.txt");`
    - `Scanner sZoSuboru = new Scanner(subor);`

- z konzoly

- `Scanner sZKonzoly = new Scanner(System.in);`

- z reťazca

- `Scanner sZRetazca1 = new Scanner("Ahoj Java");`
    - `Scanner sZRetazca2 = new Scanner("D:\\x.txt");`



# *System. [in|out|err]*

- **System.in** - Vstup z klávesnice v konzole
  - Reálne využitie biedne: shellové dialógy
- **System.out** - Výpis na konzolu
  - Už poznáme cez jeho metódy `print()` a `println()`
- **System.err** - Chybový výpis na konzolu
  - Pracuje sa s ním rovnako ako so `System.out`
  - Výpis v Eclipse sa vypisuje červeným písmom



# *java.util.Scanner*

- Po vytvorení je jedno z čoho sa číta
- Po čítaní zo súboru nesmieme zabudnúť súbor zatvoriť cez `close()`
- Filozofia `Scanner`-a: fungovanie cez dvojice metód
  - **boolean** `hasNextXXX()` ;
    - Vráti **true**, ak je možné zo vstupu prečítať hodnotu typu `XXX`
  - `XXX nextXXX()` ;
    - Vráti hodnotu typu `XXX`





# *java.util.Scanner*

- Scanner metódami `nextXXX()` a `hasNextXXX()` číta po najbližší oddelovač
- Prednastaveným oddelovačom je ľubovoľný znak, ktorý je tzv. whitespace znakom
- Typickými whitespace znakmi sú `'\t'`, `' '` a `'\n'`
- `next()` a `hasNext()` sa chápu akoby `nextString()` a `hasNextString()`
- Pre dvojicu `hasNextLine()` a `nextLine()` je oddelovačom vždy `'\n'`



## Schéma práce so Scanner-om pri čítaní zo súboru

```
File subor = new File("C:\\adresare\\subor");
Scanner scanner = null;
try {
    scanner = new Scanner(subor);

    // čítame zo scannera

} catch (FileNotFoundException e) {
    System.out.println("Súbor " +
        subor.getName() + " som nenašiel");
} finally {
    if (scanner != null)
        scanner.close();
}
```



## Čítame zo súboru

- Vytvorme si metódu, ktorá načíta zo zadaného súboru v prvom riadku veľkosť poľa čísiel a v druhom riadku obsah poľa čísiel.



# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
```



naš program

objekty

reálny svet

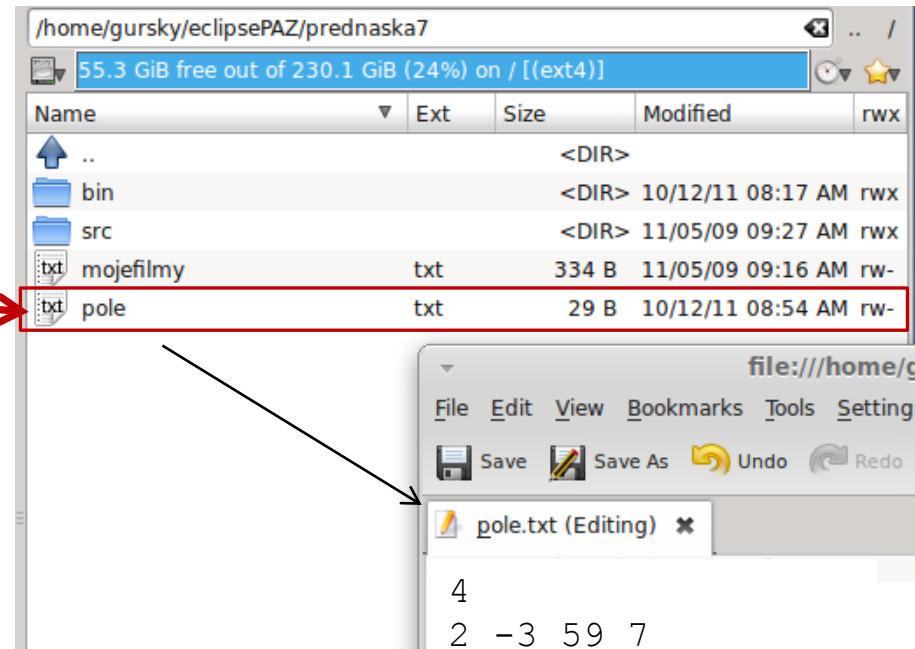
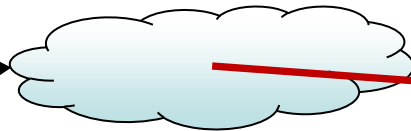
pole

null

subor

citac

null





# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
```



naš program

objekty

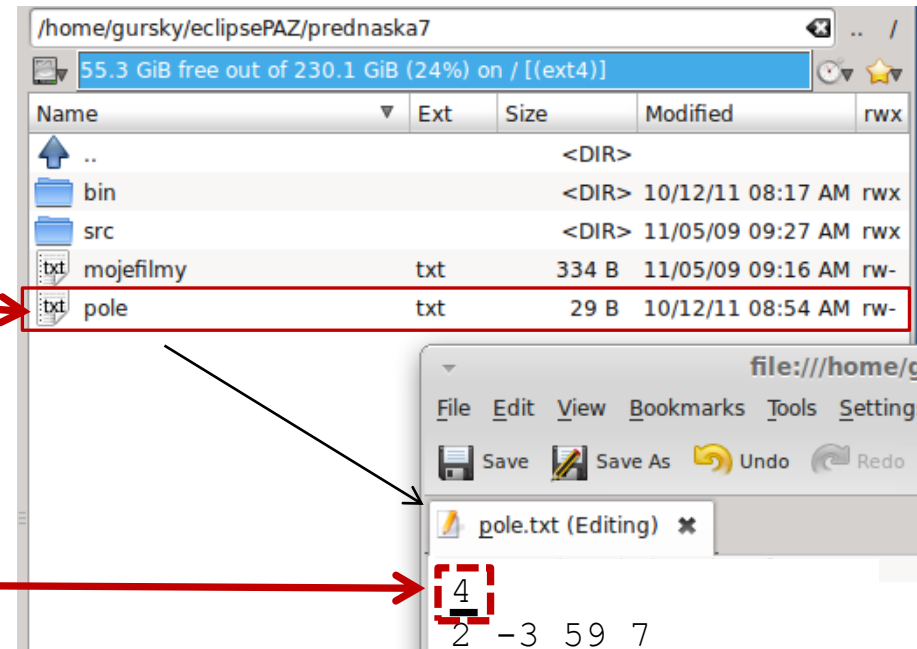
reálny svet

pole

null

subor

citac





# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



náš program

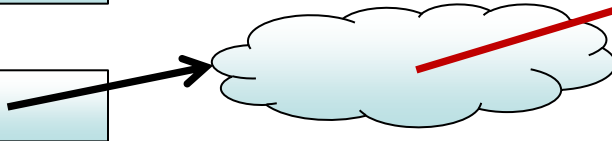
objekty

reálny svet

pole

null

citac



4  
2 -3 59 7



# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

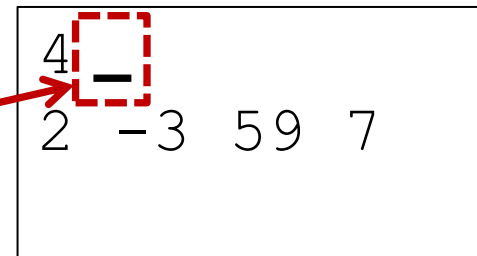
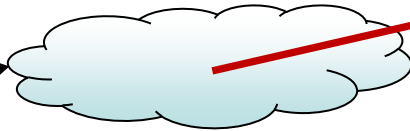
pole

null

citac

pocet

4





# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

pole

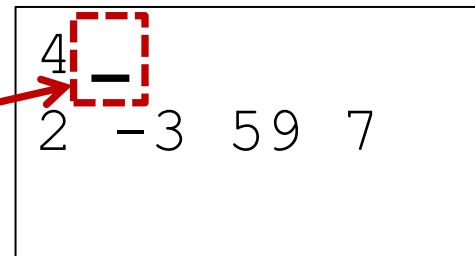
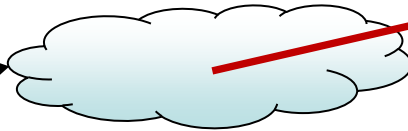
null

citac

pocet i

4

0







# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

pole



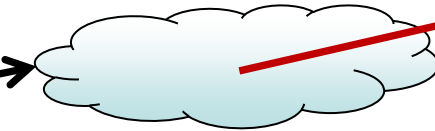
citac



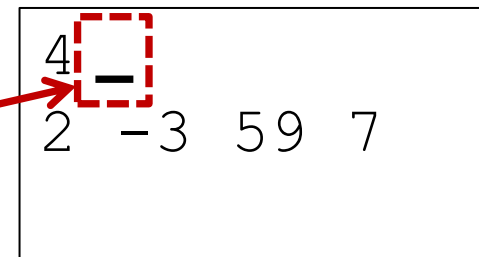
pocet i



objekty



reálny svet





# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

pole



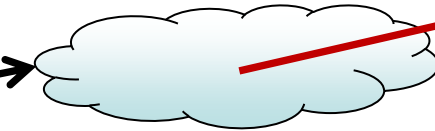
citac



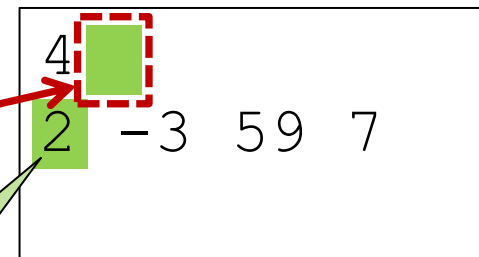
pocet i



objekty



reálny svet



hasNextInt() ???



# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

pole

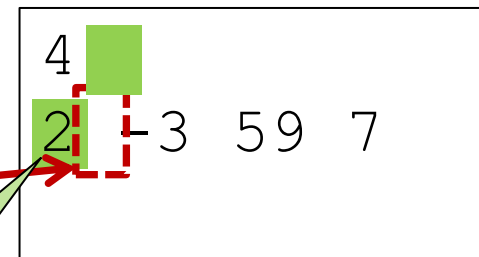


[ 2, 0, 0, 0 ]

citac



pocet i



nextInt() ???



# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

pole



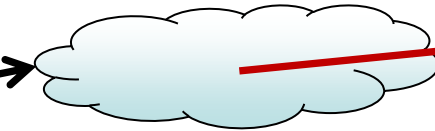
citac



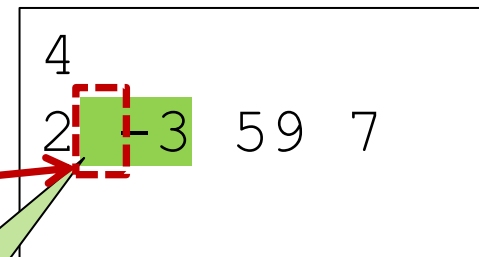
pocet i



objekty



reálny svet



true

hasNextInt() ???



# Čítame zo súboru

```
public int[] nactajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

pole

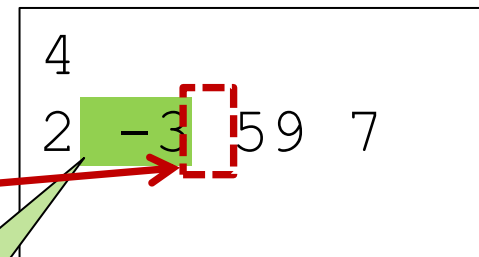


[ 2, -3, 0, 0 ]

citac



pocet i



-3

nextInt() ???



# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

pole

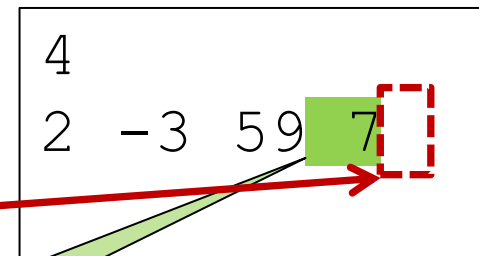


[ 2, -3, 59, 7 ]

citac



pocet i



7

nextInt() ???



# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {
    int[] pole = null;
    Scanner citac = null;
    try {
        citac = new Scanner(subor);
        int pocet = citac.nextInt();
        int i = 0;
        pole = new int[pocet];
        while (citac.hasNextInt()) {
            pole[i++] = citac.nextInt();
        }
    }
}
```



naš program

objekty

reálny svet

pole



[ 2, -3, 59, 7 ]

citac



pocet

i

4

4

false

hasNextInt() ???



# Čítame zo súboru

```
public int[] nacistajZoSuboruPole(File subor) {  
    int[] pole = null;  
    Scanner citac = null;  
    try {  
        citac = new Scanner(subor);  
        int pocet = citac.nextInt();  
        int i = 0;  
        pole = new int[pocet];  
        while (citac.hasNextInt()) {  
            pole[i++] = citac.nextInt();  
        }  
    } catch (FileNotFoundException e) {  
        System.err.println("Súbor " + subor.getName() + " sa nenašiel");  
    } finally {  
        if (citac != null)  
            citac.close();  
    }  
    return pole;  
}
```



# Ďakujem za pozornosť !

