Programovanie, algoritmy, zložitosť (UINF / PAZ1c)

Diel III.

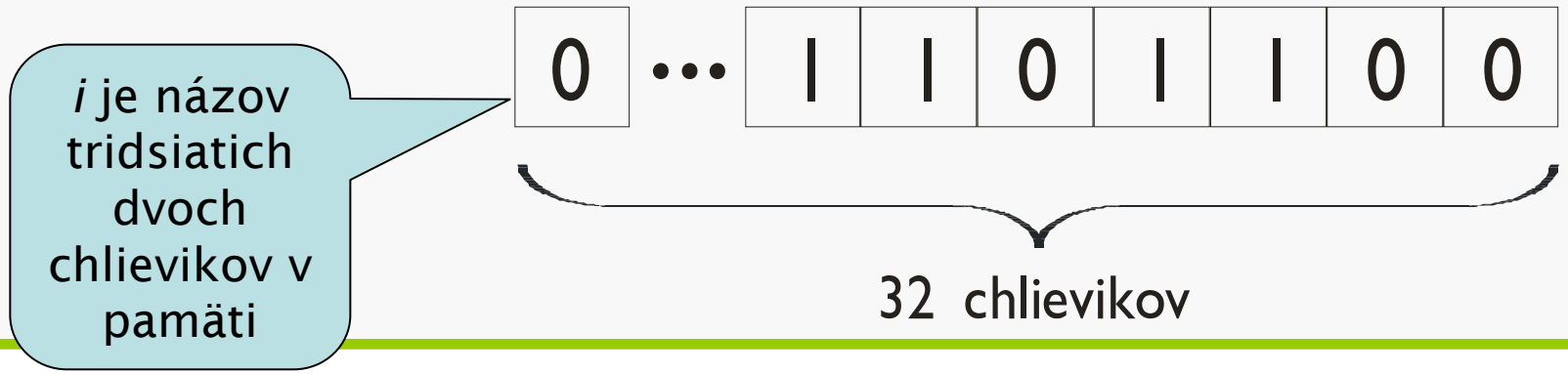
Róbert Novotný
robert.novotny@upjs.sk
8. 10. 2008

Reprezentácia dátových typov v pamäti počítača

- alias „*Bratia* == *a* *equals()* *zasahujú*"
- jestvujú dva druhy dátových typov: primitívy a objekty
  - primitívy: `int`, `boolean`, `float`, `double`,...
    - dátové typy začínajúce malým písmenom
  - objekty: `String`, `Pes`,...
    - začínajúce veľkým písmenom

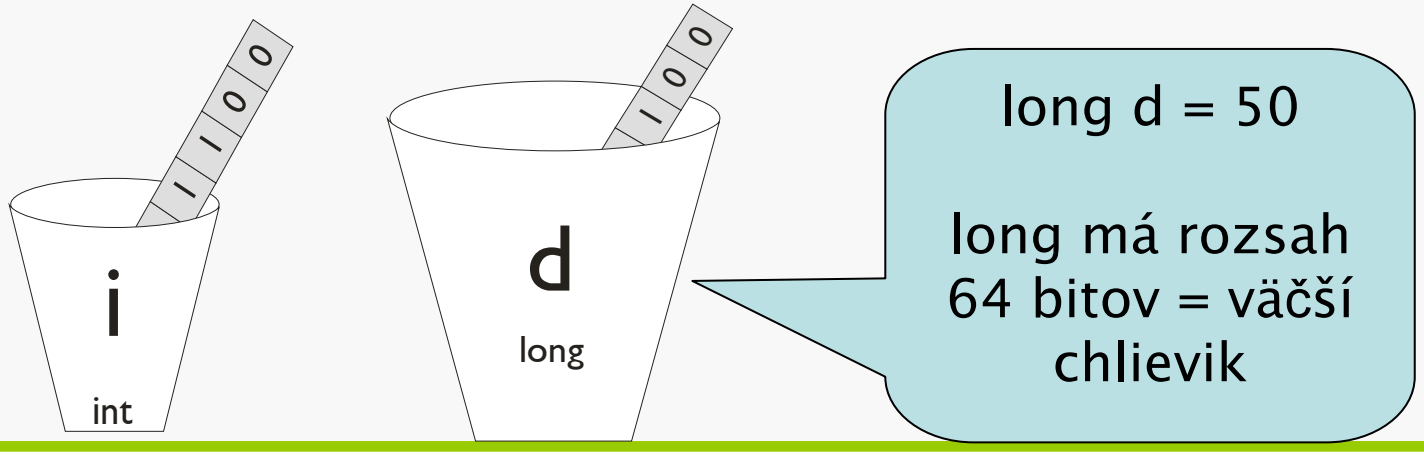
Reprezentácia primitívov v pamäti počítača

- primitívy: presne ako v Pascale
  - premenná je chlievik v pamäti, ktorý má
    - názov (`i`)
    - dátový typ (`int`)
    - veľkosť podľa dátového typu (`int`: 32 bitov)
- príklad: `int i = 52`. V binárnom kóde: 110100



Reprezentácia primitívov v pamäti počítača

- primitívy: presne ako v Pascale
  - premenná je chlievik v pamäti, ktorý má
    - názov (`i`)
    - dátový typ (`int`)
    - veľkosť (rozsah) podľa dátového typu (`int`: 32 bitov)



Reprezentácia primitívov v pamäti počítača

- porovnanie primitívov: výhradne cez ==
- porovnajú sa chlieviky bit po bite.
  - 110100 (50) == 110100 (50)
  - 110101 (51) != 110100 (50)

dátový typ	***veľkosť***
int	32 bitov
float	32 bitov
boolean	ťažko povedať, povedzme 1 bit
double	64 bitov
byte	8 bitov

Reprezentácia primitívov v pamäti počítača - objekt

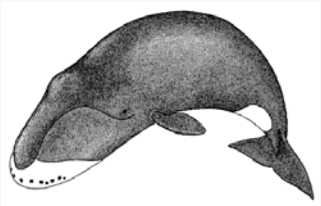
- Objekt je premenná typu špecifikovaného triedou objektu.
- premenná je chlievik v pamäti, ktorý má
  - názov (`i`)
  - dátový typ (`int`)
  - veľkosť podľa dátového typu
- Veľkosť?
  - aký veľký je `String`? A `Pes`? A `Veľryba`?

Je Veľryba väčšia než Mravec?

- nevieme, aký veľký je objekt
- objekty nemôžeme natlačiť do chlievika
  - čo keď sa nezmestia?
  - *jak dostat velrybu do pohárku*
  - nemôžeme mať nekonečne veľký chlievik

Riešenie

- smerní...ehm, referencie



<

Všetky objekty sú na kope... teda halde

Urob si haldu v pamäti počítača, v halde urob priehrady a zvnútra i zvonka ich vymaž smolou! A postav ju takto: tristo MB bude jej dĺžka, päťdesiat MB jej šírka a tridsať MB jej výška.

Do korába vojdeš ty i tvoji synovia, tvoja žena aj ženy tvojich synov s tebou.

Zo všetkých vtákov podľa svojho druhu, z dobytka podľa svojho druhu a z plazov podľa svojho druhu vojdú po dvoch do korába s tebou, aby mohli žiť.

– IT Genesis

Všetky objekty sú na kope... teda halde

- halda (heap) je priestor v pamäti určený pre objekty
  - nemýliť si s heapsortom (triedenie haldovaním)!
- je spravovaný automaticky Javou
- prostý programátor nevie o existencii haldy
- na halde sa dejú kadejaké zverstvá
  - automatické uvoľnovanie pamäte (Garbage Collection)
  - o tom však neskôr

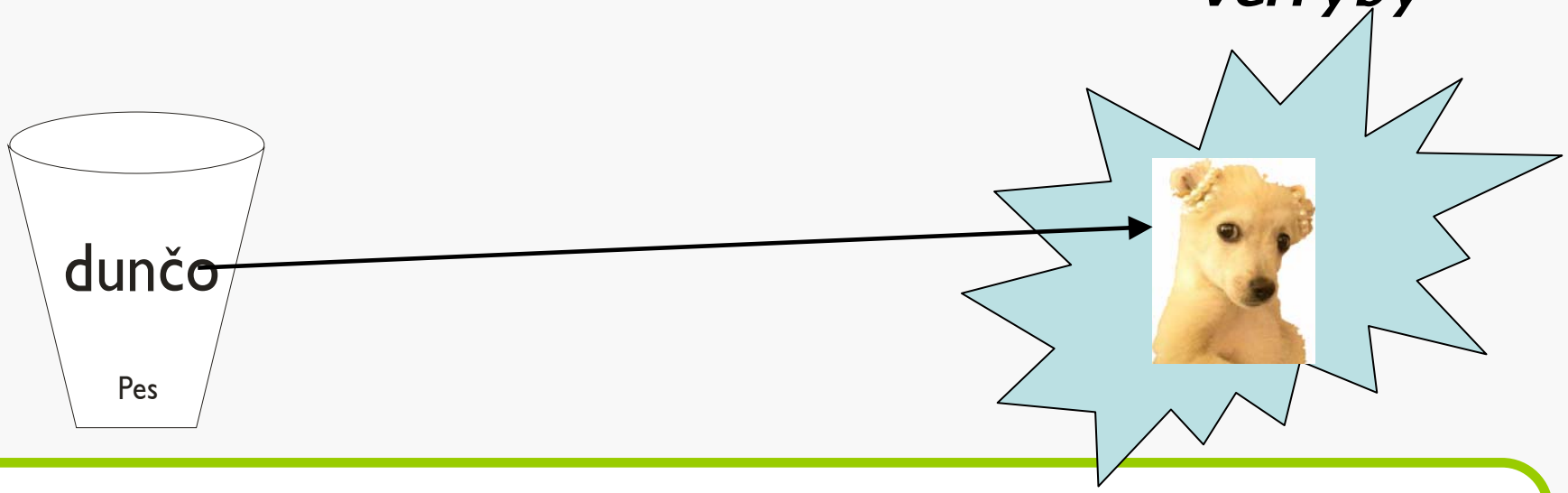
Detaily v útrobách psa

- `Pes dunčo = new Pes()`
- `Pes dunčo;`
  - vytvorí sa nové premenná `dunčo` typu `Pes`
- `new Pes()`
  - na halde sa vytvorí dostatok pamäte pre novú inštanciu

Všetky objekty sú na kope... teda halde

- `Pes dunčo = new Pes()`
- premenná `dunčo` je nasmerovaná na inštanciu psa na halde. Bude obsahovať adresu inštancie na halde

„tretí chlievik zhora, piaty sprava, vedľa veľryby"



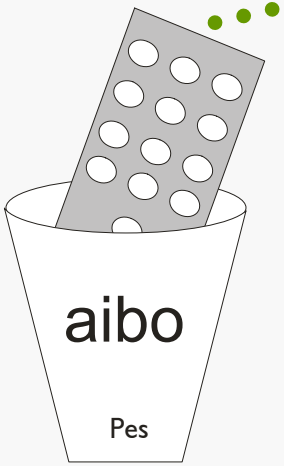
Adresa ja, adresa ty...

- premenná `dunčo` obsahuje adresu inštancie na halde
- to je presne idea smerníkov
- našťastie:
  - smerníky sú v pozadí
  - užívateľ ich nevidí
    - ani nechce vidieť
  - žiadne `^.` ako v Pascale

Podobenstvo s diaľkovým ovládaním

- premenná Pes obsahuje ,,diaľkové ovládanie" inštancie na halde

```
Pes aibo = new Pes();
```



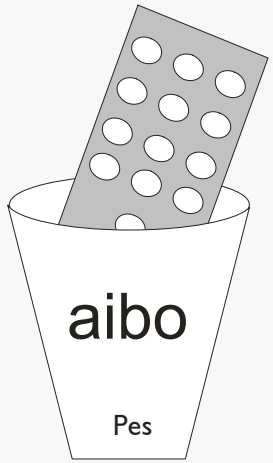
1. **`Pes aibo`** ... vytvoríme novú premennú

   teda pohárik s diaľkovým ovládaním

2. **`new Pes()`** ... vytvorenie novej inštancie na halde
3. **`priradenie`** .. diaľkové ovládanie naprogramujeme na ovládanie konkrétnej inštancie (teda Aiba).

Podobenstvo s diaľkovým ovládaním

- Ak pohárik `int` má veľkosť 32 bitov, akú veľkosť má pohárik typu `Pes`?
- Nevedno, ale ani nás to netrápi.
  - JDK od Sunu: 64 bitov
  - Java od Janka Hraška:
    - 64 bitov na obed
    - 32 bitov v noci



Otázka k diaľkovým ovládaniam

- Ak nadeklarujem premennú a nepriradím jej nič, koho riadi diaľkové ovládanie?

  `Pes aibo;`

  Premenná, ktorej nebola priradená žiadna inštancia ukazuje na **`null`**.

  - **`null`** je analógia **nil** z Pascalu: smerník, ktorý ukazuje nikam.

  Ak premenná ukazuje na null, mám diaľkové ovládanie, ale nemám k nemu televízor.

Dôsledky diaľkového ovládania

- Čo spraví nasledovný kód?

```
Pes dunčo = new Pes();
dunčo.setRasa("čuvač„);
dunčo.setVek(25);
System.out.println(dunčo.getVek());

Pes aibo = dunčo;
aibo.setVek(35);
System.out.println(aibo.getVek());

System.out.println(dunčo.getVek());
```

```
25
35
```

```
35
```

Ale prrrrečo?

```
Pes dunčo=new Pes();
```

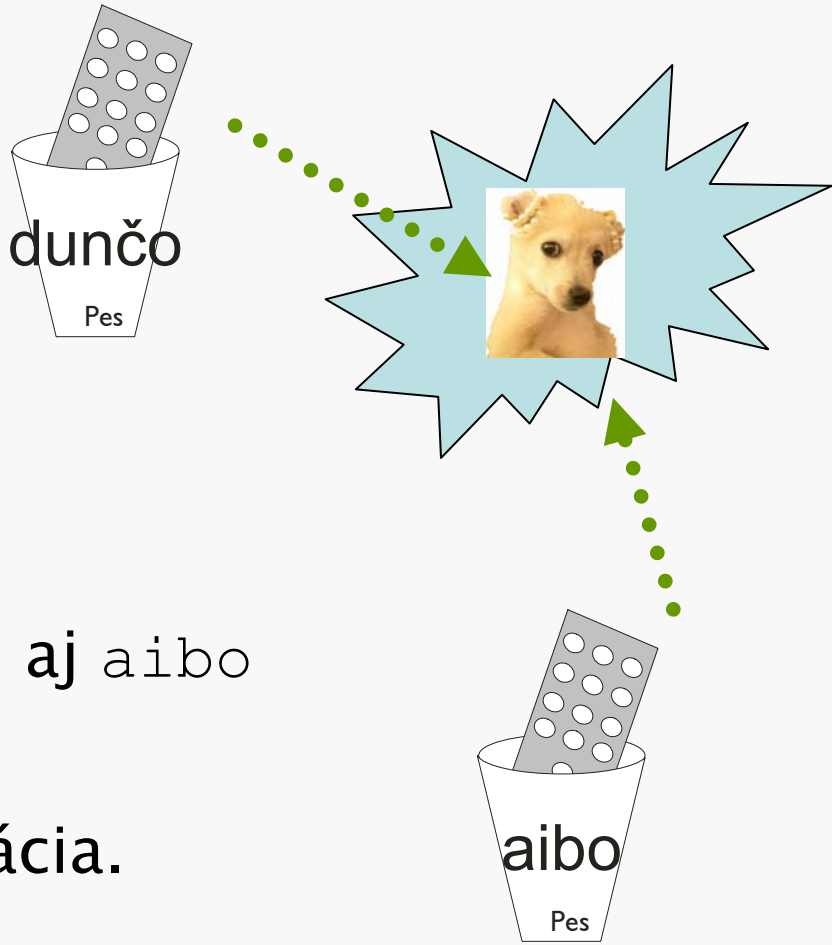
```
Pes aibo = dunčo
```

- obe diaľkové ovládania riadia toho istého psa
- ak zmeníme vek pomocou *aiba*, zmení sa aj vek pre *dunča*

Dôsledky diaľkového ovládania

```
Pes dunčo = new Pes();
Pes aibo = dunčo;

if(dunčo == aibo) {
    //platí
}
```

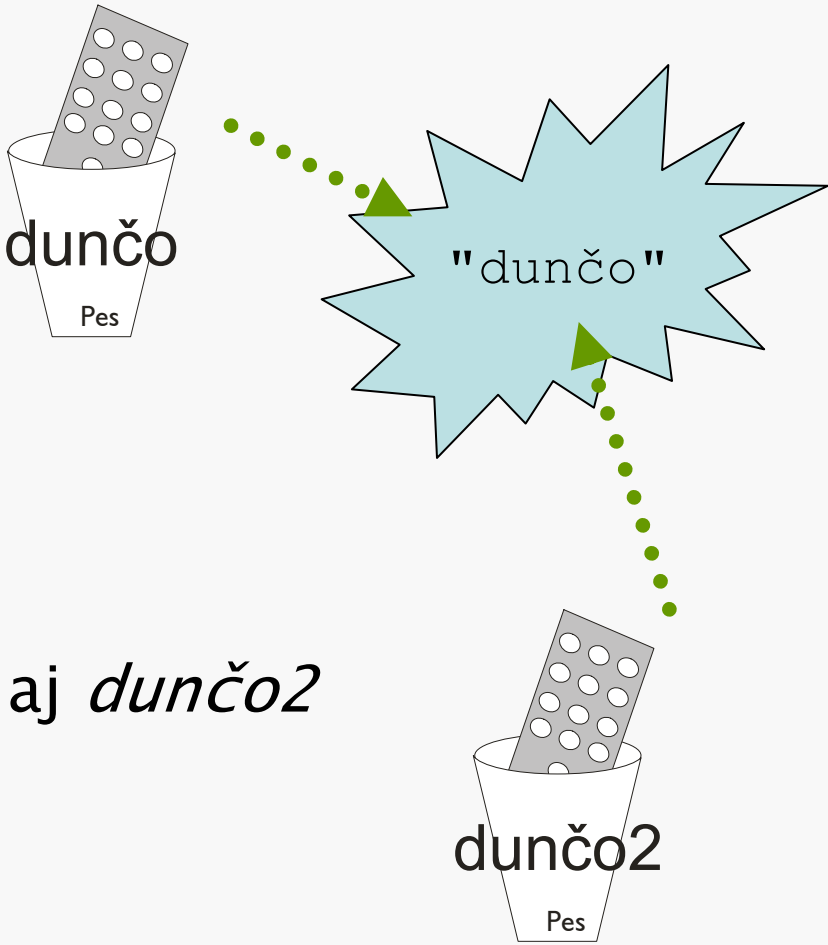


- Podmienka platí, lebo `dunčo` aj `aibo` ukazujú na toho istého psa.
- Toto je však výnimočná situácia.

Dôsledky diaľkového ovládania

```
String dunčo = "dunčo";
String dunčo2 = dunčo;

if(dunčo == dunčo2) {
    //platí
}
```

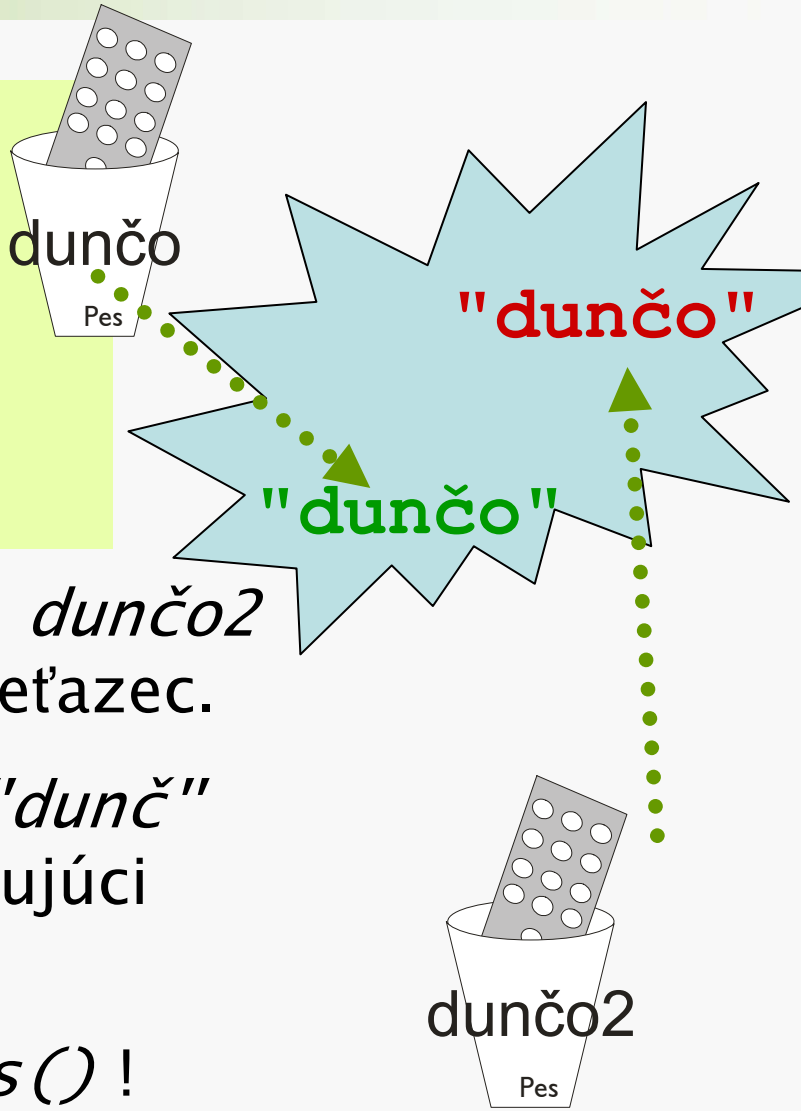


- Podmienka platí, lebo *dunčo* aj *dunčo2* ukazujú na ten istý reťazec.

Dôsledky diaľkového ovládania

```
String dunčo = "dunčo";
String dunčo2 = "dunč" + "o";

if(dunčo == dunčo2) {
    //ráno platí, večer už nie
}
```



- Podmienka platí, lebo *dunčo* aj *dunčo2* nemusia ukazovať na ten istý reťazec.
- Kompilátor nemusí vedieť, že *"dunč" + "o"* má nasmerovať na existujúci reťazec.
- Preto porovnávame cez *equals()* !

Zásada s veľkým Z

Morálne ponaučenie:

- **objekty** porovnávame cez **`equals()`**!

  názov typu objektu sa začína veľkým písmenom

  - *`Stringy sú objekty!`*
- **primitív**ne typy porovnávame cez **==**!

  názov primitívu sa začína malým písmenom

  - primitívy nie je možné porovnávať cez equals()!
  - primitív nemá metódy, nastane kompilačná chyba

Zásada s veľkým Z_2

Morálne ponaučenie 2:

- objekty porovnávame cez **==** jedine v prípade, že ho porovnávame s **null**.

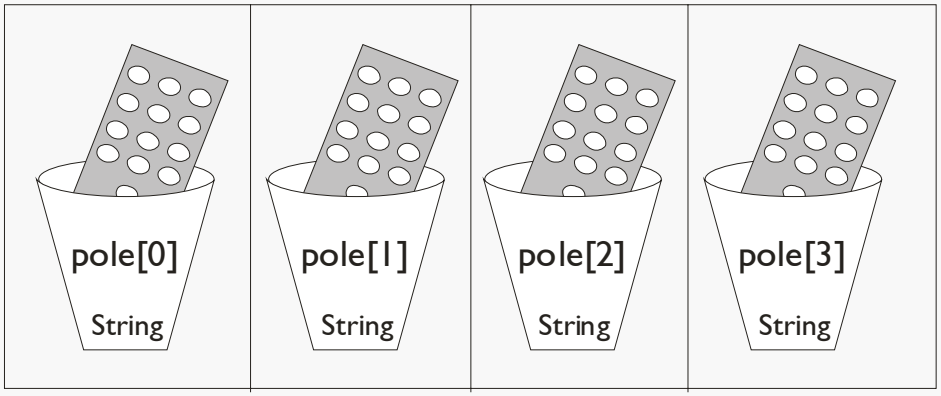
```
Pes pes; // v psovi je null

if(pes == null) {
  System.out.println("Kde je pes?“);
}
```

- nepíšeme ~~`pes.equals(null)`~~
  - vedie to k chybe
  - null znamená *nič* a *nič* nemá žiadne schopnosti (= žiadne metódy!)

Dôsledky diaľkového ovládania - polia

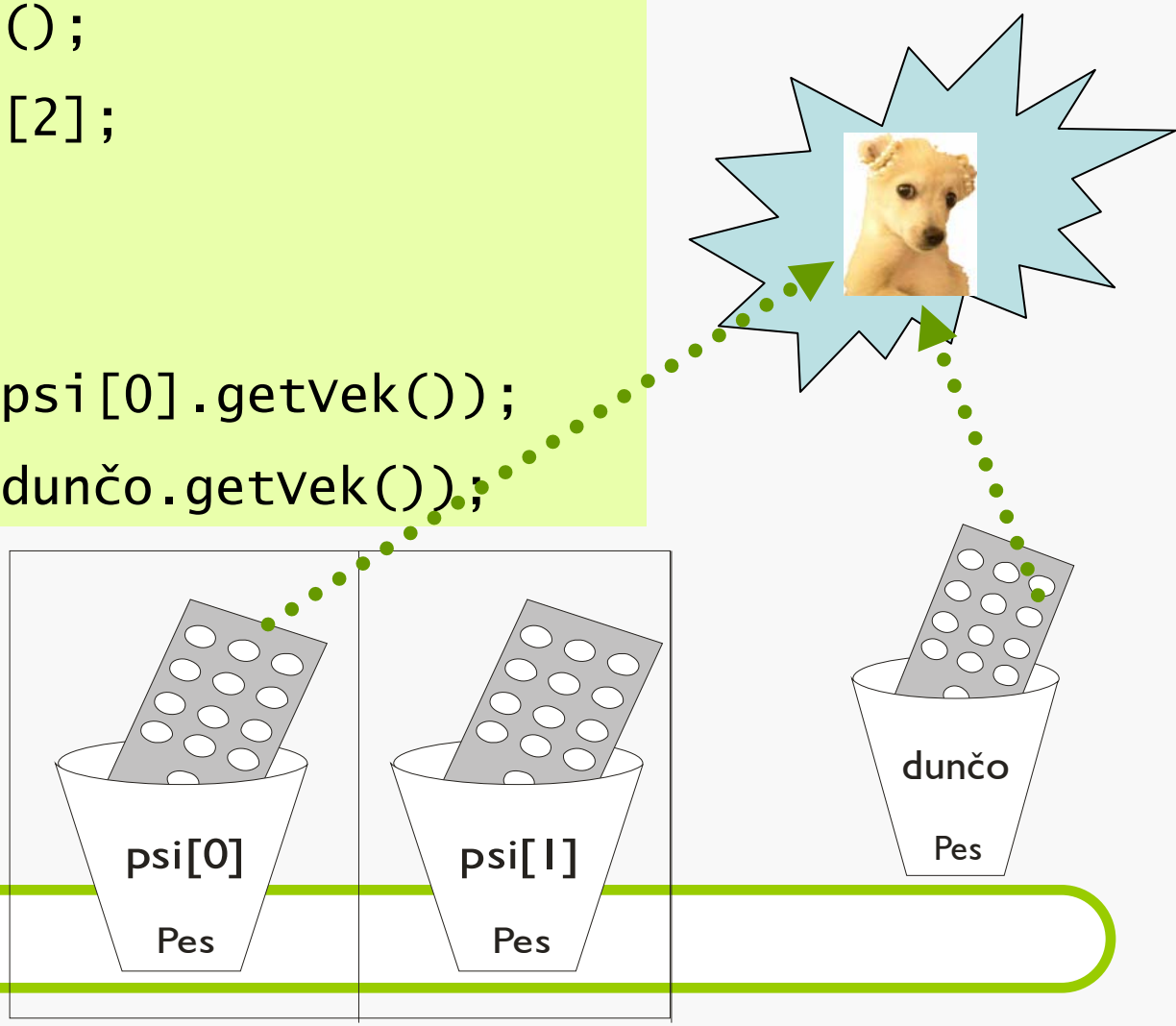
```
String[] pole = new String[4];
```



- Máme pole štyroch pohárikov s diaľkovými ovládaniami, ktoré neriadia žiaden objekt (neukazujú nikam)
- Každý z prvkov má hodnotu **null**.
- Dôsledok: `pole[0].length()` = `NullPointerException` = výbuch = snažíme sa volať metódu na neexistujúcom objekte

Dôsledky diaľkového ovládania – polia a objekty

```
Pes dunčo = new Pes();
Pes[] psi = new Pes[2];
psi [0] = dunčo;
psi[0].setVek(4);
System.out.println(psi[0].getVek());
System.out.println(dunčo.getVek());
```



Národnostné konflikty v triedach

- dosiaľ sme mali *jednoduché názvy* tried – Pes, Veľryba...
- problém nastáva, keď dva projekty pomenujú triedu rovnako
  - `Attribute`: spracovávač webových stránok v HTML
  - `Attribute`: v podpore pre tlač
  - `Attribute`: v projekte IGAP T. Horvátha
- Čo keď chcem používať vo svojej aplikácii aj HTML spracovávač aj IGAP?
  - *Riešenie 1*: dohoda medzi programátormi, premenovanie tried
    - HTMLAttribute, PrintAttribute, IgapAttribute
  - problém: s T. Horváthom sa dohodnete, s autormi Javy už nie

Zabaľme konflikty

Riešenie 2: balíčky

- zaveďme hierarchickú štruktúru á la adresáre pre triedy
- každá trieda má názov získaný z hierarchie
  - ***org.htmlparser.Attribute***: v HTML parseri
  - ***javax.print.attribute.Attribute***: v podpore pre tlač
  - ***uinf.wid.Attribute***: v projekte IGAP T. Horvátha

Zabaľme konflikty

- Ak chceme vytvoriť novú inštanciu HTML atribútu, musíme uviesť celý názov cesty.

```
org.htmlparser.Attribute atribút = new
  org.htmlparser.Attribute();
```

- Problém: dá si niekto na obed toto?

```
org.springframework.aop
.framework.autoproxy.metadata
.AttributesThreadLocalTargetSourceCreator c
   = new org.springframework.aop
        .framework.autoproxy.metadata
        .AttributesThreadLocalTargetSourceCreator();
```

Syndróm karpálneho tunelu a jeho riešenie

- namiesto klepkania názvu triedy použijeme *import*

```
import
public class AttributeTester {
   public static void main(String[] args) {
      uinf.wid.Attribute a = new uinf.wid.Attribute();
      Attribute a = new Attribute();
   }
}
```

- import hovorí toto: ak používam triedu `Attribute`, znamená to, že vlastne používam triedu `uinf.wid.Attribute`;
- *import* nie je *include*: obsah triedy `Attribute` sa **ne**vkladá do triedy `AttributeTester`

Balíčky

- každá trieda prináleží do nejakého balíčka
- výnimka-nevýnimka: existuje ***implicitný balíček*** (*default package*), ktorý je tvorený ,,koreňovým adresárom" hierarchie
- náš `Pes` bol zatiaľ v implicitnom balíčku
- triedy ***v implicitnom balíčku netreba importovať***
  - trieda v balíčku nevidí implicitný balíček
- v implicitnom balíčku nie sú štandardne žiadne triedy (pokiaľ nepoužívame *prasácky* napísané knižnice)

Štábna kultúra

- Odteraz zaradíme každú našu triedu do balíčka
- Príklad: `novotnyr.zvierata.Pes`
- Ale ako?

```
package novotnyr.zvierata;

public class Pes {
     private String rasa;
     private int vek;
     ...
}
```

To však nie je všetko...

Balíčky

- Zabudli sme sa spýtať: ***čo napr. taký String?***
  - String sa volá `java.lang.String`
  - Triedy v balíčku `java.lang` sa nemusia importovať
- Trieda v rovnakom balíčku sa nemusí importovať

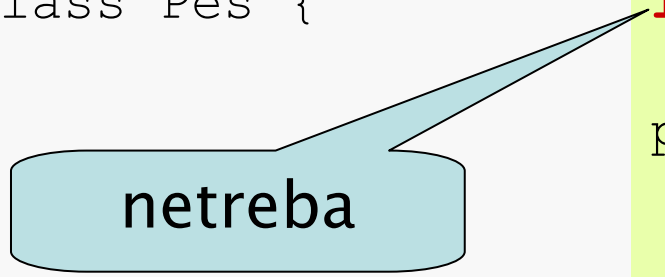
```
package novotnyr.zvierata;

public class Pes {
```

```
package novotnyr.zvierata;

import novotnyr.zvierata.Pes;

public class PesTester {
        ...
        Pes pes = new Pes();
```



Balíčky

- Viacero importov z rovnakého balíčka vieme zapísať skrátene

```
import novotnyr.zvierata.Pes;
import novotnyr.zvierata.Mačka;
import novotnyr.zvierata.Lasica;

public class PesTester {
...
```

```
import novotnyr.zvierata.*;

public class PesTester {
...
```

importuj všetky triedy z balíčka `novotnyr.zvieratá`

Nadradené a podr(i)ad(e)né balíčky

- Trieda automaticky importuje (= netreba písať import) len triedy zo svojho balíčka
- Musíme importovať aj triedy z nadradených, aj z podriadených balíčkov

import triedy z podriadeného balíčka

import triedy z nadriadeného balíčka

```
package novotnyr;

import novotnyr.zvierata.Pes;

public class Búda {
  private Pes obyvateľ;
```

```
package novotnyr.zvierata;

import novotnyr.Búda;

public class Pes {
   private Búda bydlisko
```

Nadradené a podr(i)ad(e)né balíčky

- Hviezdičkový import znamená len ***„importuj triedy z tohto balíčka"***
- Triedy z podriadených balíčkov sa takto **NEimportujú**
  - `import java.util.*` importne java.util.ArrayList, java.util.HashMap, atď
  - NEimportne napr. triedu `java.util.prefs.Preferences`, ani napr. `java.util.regex.Pattern`
- nedá sa ani použiť `import java.util.*.*.*`

Štábna kultúra

- Zdrojové súbory tried sa doteraz váľali v jednom adresári.
- Norma: skompilované triedy musia byť uložené v adresárovej štruktúre zodpovedajúcej hierarchii balíčkov
- Príklad: skompilovaná trieda `novotnyr.zvierata.Pes` musí byť v súbore

***nejakýAdresár*\novotnyr\zvierata\Pes.class**

Balíčky, CLASSPATH a iné potvory

- Ako Java vie, že triedu `novotnyr.zvierata.Pes` má načítať práve z tohto adresára?

  ***nejakýAdresár*`\novotnyr\zvierata\Pes.class`**

- premenná prostredia CLASSPATH

  WinXP: Ovládacie panely | Systém | Upresniť | Premenné prostredia | Systémové premenné

- obsahuje adresáre, v ktorých sa má začať prehľadávať hierarchia balíčkov

Balíčky, CLASSPATH a iné potvory

- Príklad:
  - majme `CLASSPATH=C:\`
  - chceme používať triedu `novotnyr.zvierata.Pes` uloženú v `D:\Java\psi\novotnyr\zvierata\Pes.class`
- Riešenie
  - prehľadávame CLASSPATH:
    - hľadáme súbor `C:\novotnyr\zvierata\Pes.class`
  - taký súbor neexistuje = **chyba!**

```
java.lang.NoClassDefFoundError: novotnyr.zvierata.Pes
```

Balíčky, CLASSPATH a iné potvory

- Riešenie
  - CLASSPATH upravíme na

```
CLASSPATH=C:\;D:\Java\psi
```

  - prehľadávame CLASSPATH
    - `C:\novotnyr\zvierata\Pes.class` nenájdený
    - `D:\Java\psi\novotnyr\zvierata\Pes.class` nájdený = OK!
- Nenastavený CLASSPATH je to isté ako

```
CLASSPATH=.
```

BODKA = Hľadaj v aktuálnom adresári

Balíčky, CLASSPATH a iné potvory

- CLASSPATH môžeme nastaviť aj pri spúšťaní java.exe

```
java -cp C:\;D:\Java\psi
   novotnyr.zvierata.Pes
```

- Integrované vývojové prostredia riešia CLASSPATH sami

Ako nenechať vybuchnúť program pri udeľovaní zápočtu

„V tvojich programoch vznikne v nich mnoho chýb a v pote tváre budeš stále opravovať svoje dielo.!

- Život programátora je smutný
- Všade je kopa chýb
  - chyby syntaktické:
    - hlásené kompilátorom – vieme vyriešiť
  - chyby pri behu programu
    - ako na ne?

Ako nenechať vybuchnúť program pri udeľovaní zápočtu

Behové chyby: čím je bližšie termín odovzdania projektu, tým viac behových chýb vzniká

- programátorský škriatok dá na vstup nesprávne údaje

  *,,Pretože tam, kde očakávate vstup 12, užívateľ just zadá **Britney Spears is not dead!***

- súbory sa záhadne presúvajú po disku
- na disku dôjde miesto

  príčinou nemusí byť nutne DC++/Torrent/eMule...

- administrátor sa rozhodol zmeniť heslá k databáze

  ,,Jsem princezná Rosella a zaspívám vám písničku: Divide by zero overflow"

Ako nenechať vybuchnúť program pri udeľovaní zápočtu

- Riešenie: Výnimky
- Príklad: synček píše domácu úlohu.
- Algoritmus:
  - vezmi učebnicu, písanku a pero
  - vyrieš príklad
  - po skončení ju odlož do aktovky

Ako nenechať vybuchnúť program pri udeľovaní zápočtu

- **Klasické jazyky:**
  - nájdi pero. Podarilo sa?
    - nájdi písanku. Podarilo sa?
      - nájdi zadanie. Podarilo sa?
- **Jazyky s podporou výnimiek:**
  - Nebudeme sa synčeka pýtať, či našiel pero, či má písanku, a či si vôbec zapísal znenie DÚ
  - Necháme ho pracovať a synček sám zahlási, ak mu niečo prekáža vo vyriešení úlohy
  - Všetky prekážky bude musieť niekto (otecko?) odstrániť, resp. sa s nimi vysporiadať

Ako nenechať vybuchnúť program pri udeľovaní zápočtu

- skús toto:
  - nájdi pero
  - nájdi písanku
  - nájdi zadanie
  - vyrieš príklad
  - odlož všetko do tašky
- ak sa niečo nepodarilo...
  - pero nemá atrament, písanku zjedol pes, zadanie je naškrabané...
- ....DÚ asi nedokončíme, musíme sa s tým nejak vysporiadať

Ako nenechať vybuchnúť...

```
try {
    Pero pero = izba.getPero();
    Písanka zošit = izba.getPísanka();
    String zadanie = učebnica.getDÚ();
    dieťa.vyriešDomácuÚlohu(pero, zošit, zadanie)
    taška.odlož(pero, zošit)
} catch (PeroJePrázdneException e) {
    System.out.println("Prepáčte, že som
    nenapísal DÚ, ale nemal som atrament");
} catch (NemámPísankuException e) {
    System.out.println("Prepáčte, že som
    nenapísal DÚ, ale nemal som písanku a
    obchod bol zavretý");
}
```

skús...

riešime chybové stavy

Ako nenechať vybuchnúť...

- Čo je `PeroJePrazdneException`?
- *exception* = výnimka
- ľubovoľná metóda môže v prípade chyby ***vyhodiť výnimku***
- výnimka znamená, že nastala chybová situácia a beh metódy sa ukončil
- kus kódu, ktorý takúto metódu volá, môže výnimku odchytiť a vysporiadať sa s ňou

Reálnejší príklad – čítanie zo súboru

```
public static void main(String[] args) {
   FileReader r = new FileReader("C:/autoexec.bat");
   int znak = 0;
   znak = r.read();
   while(znak != -1) {
       System.out.println(znak);
       znak = r.read();
   }
}
```

Trieda sa ani neskompiluje. Uvidíte prečo...

java.io.FileReader

```
Unhandled exception type FileNotFoundException
Unhandled exception type IOException
```

Reálnejší príklad – čítanie zo súboru

JavaDoc pre java.io.FileReader: hlavička metódy je

```
public int read() throws IOException
```

- Ľudskou rečou:

  Milý užívateľ. Pri čítaní zo súboru pomocou tejto metódy môže nastať nejaká vstupno-výstupná chyba. Ak sa to stane, metóda to oznámi nadradenej metóde výnimkou `IOException`.

- Teda ak voláme metódu `read()`, musíme si uvedomiť, že môže nastať chyba.
- Uvedomenie si = obalenie kódu do *try-catch*.

Reálnejší príklad – čítanie zo súboru

skús

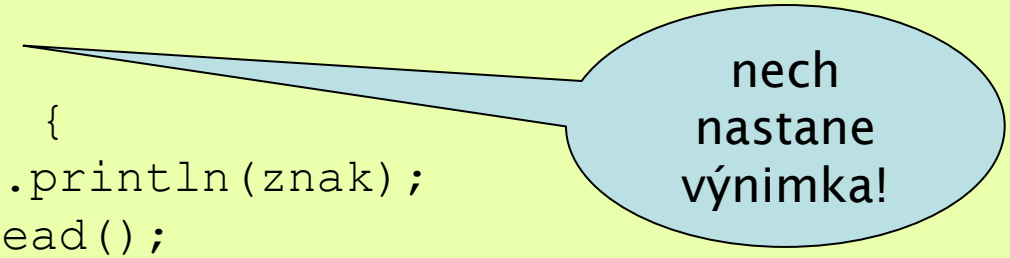
```
public static void main(String[] args) {
    try {
        FileReader r = new FileReader("C:\autoexec.bat");
        int znak = 0;
        znak = r.read();
        while(znak != -1) {
                System.out.println(znak);
                znak = r.read();
        }
    } catch (IOException e) {
        System.out.println("Pri čítaní nastala chyba");
    }
}
```

toto sprav, ak nastane výnimka

Skús čítať zo súboru a ak, božechráň, nastane chyba, vysporiadaj sa s ňou tak, že vypíš hlášku na konzolu a skonči.

Reálnejší príklad – čítanie zo súboru

```
public static void main(String[] args) {
   try {
       FileReader r = new FileReader("C:\autoexec.bat");
       int znak = 0;
       znak = r.read();
       while(znak != -1) {
              System.out.println(znak);
              znak = r.read();
       }
    } catch (IOException e) {
       System.out.println("Pri čítaní nastala chyba");
    }
}
```



Ak nastane v bloku *try* výnimka, zvyšné riadky sa preskočia a začne sa vykonávať *catch* blok.

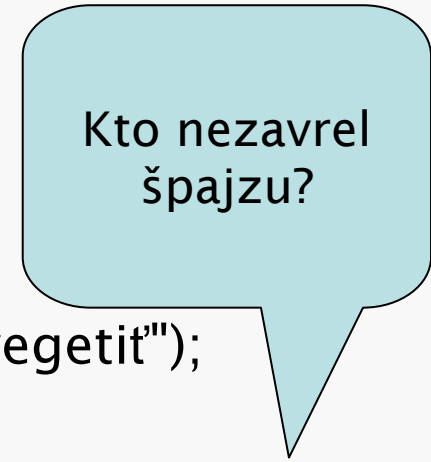
A nakoniec sa všetko dobre skončilo...

```
try {
        ...
} catch (IOException e) {
        System.out.println(e.getMessage());
}
```

- výnimky sú tiež objektami
- tuto odchytávame výnimku `java.io.IOException`
- trieda `java.io.IOException` má všakovaké metódy
  - `getMessage()` – textový popis výnimky
- `e` je objekt typu `IOException`, môžeme ho použiť na výpis
- bližšie informácie viď dokumentácia JavaDoc

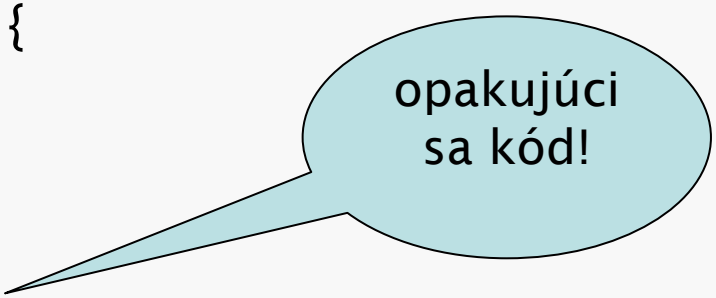
A nakoniec sa všetko dobre skončilo...

```
try {
      otvor špajzu; //otvorŠpajzu throws ŠpajzaVykradnutáException
      vezmi motyku; //getMotyka throws MotykaNenájdenáException
      okop záhradu;
      zatvor špajzu;
} catch (MotykaNenájdenáException e) {
      System.out.println("Nenašiel som motyku. Idem vegetiť");
} catch (ŠpajzaVykradnutáException e) {
    System.out.println("Hm, tak nič.");
}
```



Riešenie č. 1

```
try {
    otvor špajzu; //otvorŠpajzu throws
    ŠpajzaVykradnutáException
    vezmi motyku; //getMotyka throws MotykaNenájdenáException
    okop záhradu;
    zatvor špajzu;
} catch (MotykaNenájdenáException e) {
    System.out.println("Nenašiel som motyku. Idem vegetiť");
} catch (ŠpajzaVykradnutáException e) {
   System.out.println("Hm, tak nič.");
}
if (špajza.jeOtvorena()) zatvor špajzu;
```



Riešenie č. 2 – blok *finally*

```
try {
      otvor špajzu; //otvorŠpajzu throws ŠpajzaVykradnutáException
      vezmi motyku; //getMotyka throws MotykaNenájdenáException
      okop záhradu;
      zatvor špajzu;
} catch (MotykaNenájdenáException e) {
      System.out.println("Nenašiel som motyku. Idem vegetiť");
} catch (ŠpajzaVykradnutáException e) {
   System.out.println("Hm, tak nič.");
} finally {
   zatvor špajzu;
}
```

toto nám už netreba

blok vykoná *nakoniec*, či už nastala výnimka, alebo nie

Konečne .. blok *finally*

- Blok finally sa vykoná **vždy**
- Ak nastane v try bloku výnimka
  - vykoná sa kód v príslušnom catch bloku
  - Potom sa vykoná *finally* blok
- Ak výnimka nenastane
  - dokončí sa *try* blok
  - Potom sa vykoná *finally* blok

Prípad, že je všetko OK

```
try {
      otvor špajzu; //otvorŠpajzu throws ŠpajzaVykradnutáException
      vezmi motyku; //getMotyka throws MotykaNenájdenáException
      okop záhradu;
} catch (MotykaNenájdenáException e) {
      System.out.println("Nenašiel som motyku. Idem vegetiť");
} catch (ŠpajzaVykradnutáException e) {
    System.out.println("Hm, tak nič.");
} finally {
    zatvor špajzu;
}
```

Ak sa veci pokazia...

```
try {
      otvor špajzu; //otvorŠpajzu throws ŠpajzaVykradnutáException
      vezmi motyku; //getMotyka throws MotykaNenájdenáException
      okop záhradu;
} catch (MotykaNenájdenáException e) {
      System.out.println("Nenašiel som motyku. Idem vegetiť");
} catch (ŠpajzaVykradnutáException e) {
    System.out.println("Hm, tak nič.");
} finally {
    zatvor špajzu;
}
```

Praktický príklad – načítanie súboru

```
FileReader súborovýČitateľ = new FileReader("C:/test.txt");
BufferedReader br = new BufferedReader(súborovýČítateľ);
String riadok = null;
riadok = br.readLine();
while(riadok != null) {
  System.out.println(riadok);
  riadok = br.readLine();
}
br.close();
```

červeným písmom sú miesta výnimiek

Praktický príklad – načítanie súboru 2

```
try {
   FileReader súborovýČitateľ = new FileReader("C:/test.txt");
   BufferedReader br = new BufferedReader(súborovýČítateľ);
   while((String riadok = br.readLine()) != null) {
     System.out.println(riadok);
     riadok = br.readLine();
   }
} catch (FileNotFoundException e) {
   System.out.println("Súbor nebol nájdený");
} catch (IOException e) {
   System.out.println("Chyba pri čítaní!");
} finally {
   br.close();
}
```

- Premenná `br` vo finally bloku neexistuje!
- `FileNotFoundException` je vlastne `IOException` (dedičnosť...)

Praktický príklad – načítanie súboru 3

```
BufferedReader br = null;
try {
   br = new BufferedReader(new FileReader("C:/test.txt");
   ...
} catch (FileNotFoundException e) {
   System.out.println("Súbor nebol nájdený");
} catch (IOException e) {
   System.out.println("Chyba pri čítaní!");
} finally {
   if(br != null) {
     try {
         br.close();
     } catch (IOException e) {
     }
   }
}
```

chutné, však?

Ako vyhodiť vlastnú výnimku

- výnimky sú objektami
- aj my môžeme vyhadzovať výnimky

```
void setVek(int vek) throws ZápornýVekException
{
   if(vek < 0) {
      ZápornýVekException e = new ZápornýVekException();
      throw e;
   }
}
```

vyhlasujem, že v tejto metóde môže nastať chyba!

vyhoď výnimku!

- pravidlo: každá výnimka, ktorá môže nastať, musí byť uvedená v hlavičke metódy

Ako vyhodiť vlastnú výnimku

- výnimky sú objektami
- aj my môžeme vyhadzovať výnimky

```
public class ZápornýVekException extends Exception {
   //tu nič nie je
}
```

- vytvorili sme vlastnú výnimku
- výnimka môže mať
  - vlastné inštančné premenné
  - vlastné metódy
  - konštruktory...

Ako odchytiť vlastnú výnimku

- príklad ošetrenia vlastnej výnimky

Unhandled exception type ZápornýVekException!

```
public static void main(String[] args)
  Pes pes = new Pes();
  pes.setVek(-2000);
}
```

```
public static void main(String[] args) {
  try {
    Pes pes = new Pes();
    pes.setVek(-2000);
  } catch (ZápornýVekException e) {
    System.out.println("Psovi nemožno nastaviť záporný vek!");
  }
}
```

Pohadzujeme výnimky hore-dole

- ak naša metóda výnimku neošetruje, môže ju posunúť ďalej volajúcej metóde – výnimka vybuble vyššie
- platí pravidlo:
  - výnimku musíme **buď odchytiť** v `catch` bloku
  - alebo ju môžeme neošetriť a **poslať ďalej**

„systém padajúceho [censored]..."

- výnimku pošleme ďalej tak, že ju uvedieme v `throws` klazule v hlavičke metódy

ak nastane problém, niekto ho vyriešiť musí!

Pohadzujeme výnimky hore-dole

- ak naša metóda výnimku neošetruje, môže ju posunúť ďalej volajúcej metóde – výnimka vybuble vyššie

```
public class Čitateľ {
  void načítaj() throws FileNotFoundException
  {
    String s = "C:/test.txt";
    FileReader r = new FileReader(s);
  }
}
```

táto metóda hádže FileNotFoundException

ošetrenie necháme na niekoho iného

Neošetrená výnimka FileNotFoundException

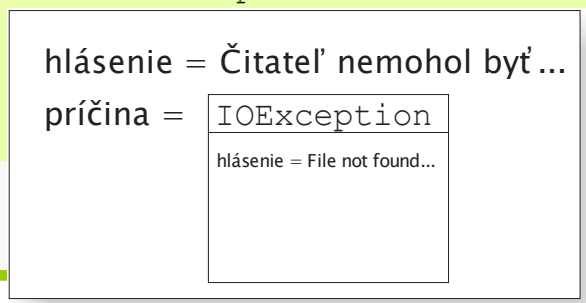
Pohadzujeme výnimky hore-dole

- alternatívne môžeme zabaliť výnimku do novej, popisnejšej

```
public class Čitateľ {
   void načítaj() throws ČitateľException {
     try {
       String s = "C:/test.txt";
       FileReader r = new FileReader(s);//throws FileNotFoundException
     } catch (FileNotFoundException e) {
       throw new ČitateľException("Čitateľ nemohol byť načítaný, e");
     }
   }
}
```

konštruktor hádže ČitateľException

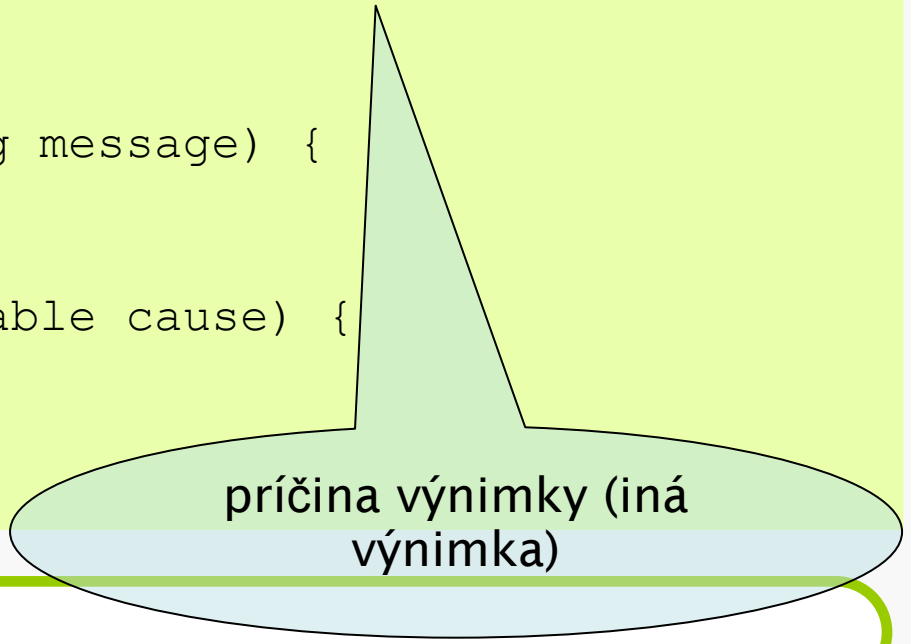
Vytvoríme novú, popisnejšiu výnimku, ktorou obalíme nízkoúrovňovú výnimku



Pohadzujeme výnimky hore-dole

- do výnimky `ČitateľException` musíme samozrejme dodať konštruktory (lebo tie sa nededia)

```
public class ČitateľException extends Exception {
  public ČitateľException() {
    super();
  }
  public ČitateľException(String message, Throwable cause) {
    super(message, cause);
  }
  public ČitateľException(String message) {
    super(message);
  }
  public ČitateľException(Throwable cause) {
    super(cause);
  }
}
```



Prebaľovanie výnimiek

- načo je dobré prebaľovanie výnimiek?
- *Príklad*: sústruh-výrobná linka-továreň
- ***Sústruh***: UrvaloŠeKoľečkoException
  - ***Výrobná linka***: VýrobnáLinkaException
    - ***Továreň***: TováreňException
- Riaditeľa továrne netreba zaťažovať s tým, že sa pokazil sústruh č. 244424/A. Jeho zaujíma hlavne to, či továreň funguje. Konkrétne príčiny chýb ho trápia až v druhom slede.

Pohadzujeme výnimky hore-dole

- príklad bublania výnimky

```
try {
   Čitateľ č = new Čitateľ();
   č.načítaj();
} catch (FileNotFoundException e) {
   e.printStackTrace(); //vypíše toto:
}
```

stack trace
zoznam vnorených volaní metód až k pôvodcovi výnimky

```
java.io.FileNotFoundException: C:/test.txt
  at java.io.FileReader.<init>(FileReader.java)
  at java.io.FileReader.<init>(FileReader.java)
  at Čitateľ.čítaj(Čitateľ.java)
  at ČitateľTester.main(ČitateľTester.java)
```

Pohadzujeme výnimky hore-dole

- príklad bublania výnimky

```
java.io.FileNotFoundException: C:/test.txt
  at java.io.FileReader.<init>(FileReader.java)
  at java.io.FileReader.<init>(FileReader.java)
  at Čitateľ.čítaj(Čitateľ.java)
  at ČitateľTester.main(ČitateľTester.java)
```

- Výnimka bublala z útrob Javy, pretože ju nik neodchytil
- Dobublala až na vrchol do metódy `main()`, kde sme ju ošetrili

Časté chyby

- Všetko zatajte!

```
try {
   Čitateľ č = new Čitateľ();
   č.načítaj();
} catch (FileNotFoundException e) {}
```

výnimka sa zhltne, nik sa o nej nedozvie.

Geniálne, ak program zdochne a nik nevie prečo.

- Keď máte v ruke výnimky, všetko vyzerá ***výnimočne***

```
try {
   int i = 0;
   while(true)
      a[i++] = 2 * i;
} catch(ArrayIndexOutOfBoundsException e) { }
```

načo máme cyklus a dĺžku poľa?

Časté chyby

- Výnimka si a vo výnimku sa obrátiš!

```
void načítaj() throws Exception {
   …
}
```

Milý programátor. V metóde môže nastať *nejaká* chyba. Hádaj aká!

- Výnimky by mali slúžiť na ošetrovanie ***výnimočných*** situácií
- Výnimočné situácie sú tie, s ktorými sa metóda nevie vysporiadať sama
- Ostatné situácie, pokiaľ sa s nimi vieme vysporiadať, riešme klasicky (*if*, *while* atď)