

Programovanie, algoritmy, zložitosť (UINF / PAZ1c)

Diel VI.

Róbert Novotný
robert.novotny@upjs.sk
29. 10. 2008



Pohadzujeme výnimky hore-dole

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

„systém padajúceho...“

- 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 vybubľe vyššie

táto metóda hádže
FileNotFoundException
Exception

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

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

ČitateľException

hlásenie = Čitateľ nemohol byť...

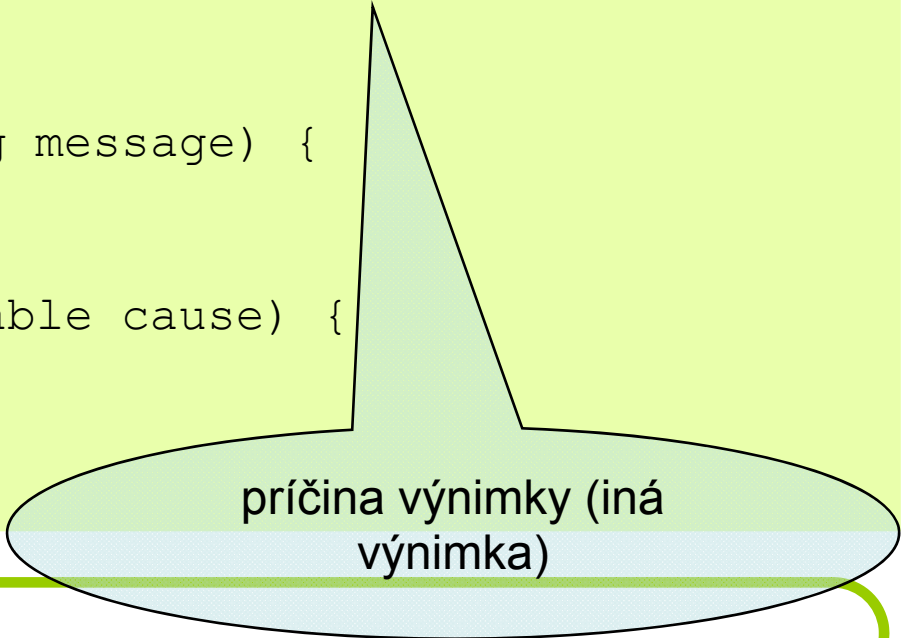
príčina = `FileNotFoundException`

hlásenie = File not found...

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);  
    }  
}
```



príčina výnimky (iná výnimka)

Prebaľovanie vÝnimiek

- načo je dobré prebaľovanie vÝnimiek?
- *PrÍklad*: sústruh-vÝrobnÁ linka-tovÁreň
- **Sústruh**: UrvaloŠeKoľečkoException
 - **VÝrobnÁ linka**: VýrobnÁLinkaException
 - **TovÁreň**: TovÁreňException
- Riaditeľ a továrne netreba zaťažovať s tým, že sa pokazil sústruh č. 244424/A. Jeho zaujíma hlavne to, či továrne 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)
```

- příklad 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 útrobov Javy, pretože ju nik neodchytil
- Dobublala až na vrchol do metódy `main()`, kde sme ju ošetrili

Výnimiek existuje viacero druhov

- zoznam možno vytvoriť s vopred danou kapacitou

```
ArrayList zoznam = new ArrayList(-25);
```

```
java.io.IllegalArgumentException: Illegal capacity: -25  
at java.util.ArrayList.<init>(ArrayList.java)  
at ZoznamTester.main(ČitateľTester.java)
```

- vyhodila sa výnimka `IllegalArgumentException`
- lenže v hlavičke konštruktora nie je žiaden `throws!`
- ako je to možné?
- môže metóda vyhodiť výnimku, čo nie je v `throws?`



Výnimiek existuje viacero druhov

- v Jave existujú ~~dva~~ tri druhy výnimiek:
 - **kontrolované** výnimky (*checked exceptions*)
 - ak metóda hádže výnimku, musí ju uviesť v `throws`
 - **nekontrolované** výnimky (*unchecked exceptions, runtime výnimky, behové výnimky*)
 - ak metóda hádže výnimku, nie je potrebné ju dať do `throws`
 - **chyby** (*errors*)
 - princíp ako v nekontrolovaných výnimkách

Výnimiek existuje viacero druhov

- `IllegalArgumentException` je nekontrolovaná výnimka
- indikuje nesprávnu hodnotu parametra metódy (zoznam zápornej dĺžky nemá zmysel)
- konštruktor `ArrayListu` ju nemusel uviesť do `throws`

Ako spoznám nekontrolovanú výnimku?

```
public class IllegalArgumentException  
    extends RuntimeException {
```

- nekontrolované výnimky dedia od `RuntimeException`



Kedy použiť akú výnimku?

- **chyby** (dedia od `Error`): abnormálny stav systému, aplikácia nemá možnosť sa brániť
 - `OutOfMemoryError`: došla pamäť, zachraňovať niečo nemá zmysel
 - `VirtualMachineError`: virtuálny stroj má problémy s chodom, programy nemôžu bežať ďalej



Kedy použiť akú výnimku?

- **kontrolované výnimky**: chyby, ktoré sa dajú predpokladať a z ktorých sa možno zotaviť
 - *chyba*: použitie súboru, ktorý neexistuje
 - *zotavenie*: získame existujúci súbor
 - príklady: nesprávny vstup od používateľa, problémy s databázou, ...
- **nekontrolované výnimky**: nedajú sa predpokladať, nedá sa z nich zotaviť
 - príklad: programátorské chyby, nesprávne použitie metódy, logické chyby, nedodržanie kontraktu metódy

Ta jak fasa s totymi vynimkami!

- (ne)kontrolované výnimky sú kontroverzné
*„Kontrolované výnimky sú experimentom,
ktorý zlyhal.“*



– Bruce Eckel, hrdina Javy

*„Kontrolované výnimky pre zotaviteľné chyby,
nekontrolované pre programátorské chyby.“*



– Joshua Bloch, iný hrdina Javy

- žiadny iný OOP jazyk nemá kontrolované výnimky
 - ni C# (poučili sa(?)), ni Python, ni C++...

Ta jak fasa s totymi vynimkami!

Wow, super, odteraz všetky moje výnimky budú dediť od `RuntimeException` a nemusím ich dávať do `throws`!

- filozofia smeruje k Eckelovi a kol.
 - všetky výnimky sú **nekontrolované**
 - ale:
 - uvedieme ich do *throws* (ak to má zmysel)
 - uvedieme ich do *dokumentácie*
- **dokumentácia**: nenastane náhla výnimka vynorená z hlbín kódu
- **throws**:
 - výnimka môže automaticky prebublať vyššie
 - pomáha IDE generovať kód

Časté chyby

- Všetko zatajte!

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

výnimka sa zhltnie, 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

- Však nech vidia, čo sa deje vo vnútri!

```
public String upečKoláč()  
  
    throws IOException, SQLException
```

lebo niekde v
útrobách sa načítava
recept zo súboru,
ktorý krachne...

- Načo prebaľovať výnimky?

```
try {  
    zaregistrujŠtudenta(študent)  
} catch (RegistráciaZlyhalaException) {  
    throw new Exception();  
}
```

je super, že
používateľ
dostane
všeobecnú
výnimku bez
akéhokoľvek
popisu

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)

Časté chyby

- Používajme štandardné výnimky, ak to ide

```
void setVek(int vek) {  
    if(vek < 0) {  
        throw new IllegalArgumentException(  
            "Vek nemôže byť záporný");  
    }  
}
```

IllegalArgumentExcepti on	Nesprávna alebo nepovolená hodnota parametra.
NullPointerExcepti on	Ktosi nám zadal <code>null</code> hodnotu na vstup.
IllegalStateExcepti on	Snažíme sa dostať inštanciu do stavu, ktorý nie je povolený.
UnsupportedOperati onExcepti on	Trieda nepodporuje danú metódu.
IOExcepti on	Vstupno-výstupná chyba