Otázka 30 - Y36PJV

Zadání

Výjimky, jejich vznik, vyhození, odchyt a zpracování. (Y36PJV)

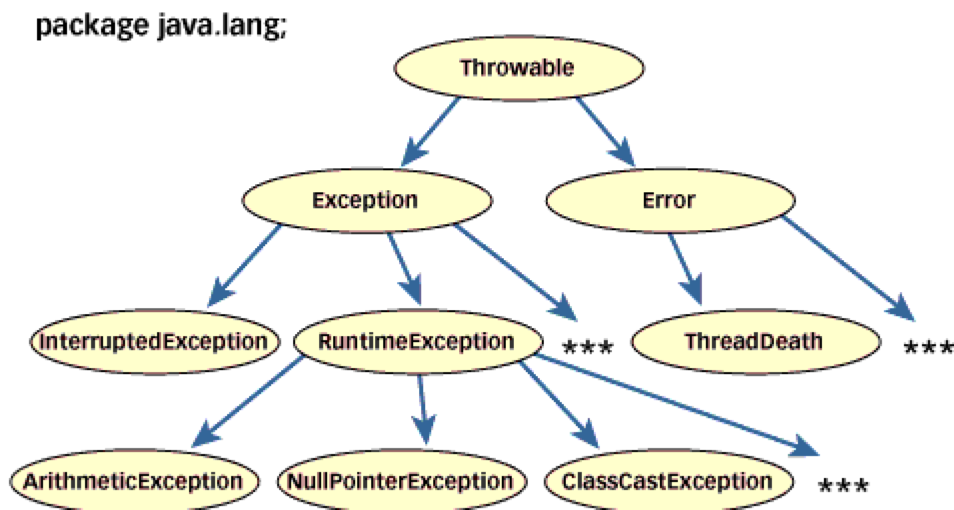
Výjimky, jejich vznik, vyhození, odchyt a zpracování. (Y36PJV)

Výjimka (exception) je definována jako událost, která nastane během provádění programu a která naruší normální běh instrukcí. Výjimka je vyvolána například při chybném otevření souboru, při překročení mezí pole, při aritmetické chybě apod. Výjimky způsobují nepředpokládatelné kombinace vstupů, poruchy HW, atd.

Když v Javě vznikne výjimka, vytvoří se v programu objekt, který ponese informace o vzniklém problému do místa, kde tento problém bude možné vyřešit. Nositelem informace o druhu problémů, je především typ tohoto objektu, navíc do tohoto objektu lze uložit další informace o daném problému. Výjimky jsou tedy objekty, které sloužící k indikaci a nápravě chyb v běhu programu.

Třídy pro přenos informací o výjimkách

Výjimky jsou instance tříd, které patří do stromu tříd *java.lang.Throwable*, odkud dědí všechny metody.



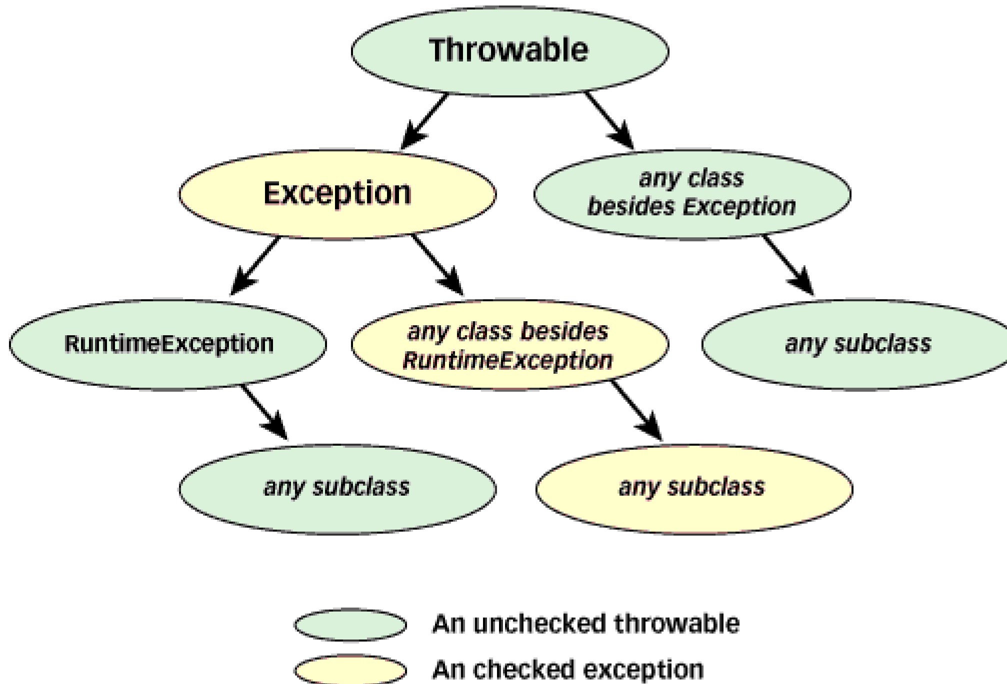
Pouze částečný strom rodiny Throwable.

Rozhraní *Throwable* má 2 potomky *Exception* (výjimka) a *Error* (chyba). Výjimky (*Exceptions*) jsou takové stavy, ze kterých se program může zotavit (například přistupujeme mimo velikost pole, nebo nastala chyba při čtení souboru). Z chyb (*Error*) se již korektní program nemá zotavit (například dojde paměť).

Existují dva druhy výjimek – **checked** (kontrolované) a **unchecked** (nekontrolované). Platí zásada, že pokud výjimka patří do stromu *java.lang.RuntimeException*, je nekontrolovaná, jinak je kontrolovaná.

PDF created with pdfFactory trial version www.pdffactory.com

- **checked** (kontrolované, hlídané, synchronní) - vyžadují ošetření klauzulí *try – catch* anebo vyznačení *throws* v hlavičkách metod.
- **unchecked** (nekontrolované, nehlídané, asynchronní) - nevyžaduje vyznačení ani ošetření. Jde o výjimky, které nemusí programátor kontrolovat, protože se nepředpokládá, že nastanou. Příkladem je procházení pole, nemá smysl každý přístup do pole uzavírat do try catch bloku.



Vznik výjimek

Výjimky buď vyhazuje je JVM (např. *NullPointerException*), nebo se o vznik výjimky musí postarat programátor. K vyvolání výjimky slouží klíčové slovo *throw*, za nímž uvedeme odkaz na objekt, který ponese informace o chybě. Můžeme použít odkaz na existující objekt, nebo můžeme vytvořit nový pomocí *new*. Po vzniku výjimky se přeruší přirozené pořadí provádění příkazů a program začne hledat *handler*, část kódu, která vzniklou situaci ošetří.

Pomocí *throws* nemusí být deklarovány:

- potomci *java.lang.Error*
- potomci *java.lang.RuntimeException*
  - je potomek *java.lang.Exception*
  - př. *NullPointerException*, *ArrayIndexOutOfBoundException*

Generování výjimek

příkaz throw:

- vyhodí výjimku
- „parametr" – reference na objekt typu Throwable

throw new MojeVyjimka();

Vyhazovat lze existující výjimky, častěji však vlastní výjimky.

PDF created with pdfFactory trial version www.pdffactory.com

java.lang.Throwable

Má atribut (private) typu String:

- obsahuje podrobnější popis výjimky
- metoda String getMessage()

Konstruktory:

- Throwable()
- Throwable(String mesg)
- Throwable(String mesg, Throwable cause) *od 1.4*
- Throwable(Throwable cause) *od 1.4*

Metody:

- void printStackTrace()

Vlastní výjimky

Vlastní výjimky se vytvoří jako potomek již existující výjimky. Pokud nová výjimka nepřidává žádné vlastní atributy či metody, pak je lepší použít již existující výjimku a pouze změnit text. Při vytvoření vlastní výjimky obvykle definujeme dva konstruktory (jeden bez parametrů, druhý s parametrem String pro uložení doprovodné zprávy). Vše ostatní se dědí z třídy Throwable. Hlavním indikátorem závady je (často velmi dlouhé) jméno výjimky – tím vzniká integrovaný „chybník“ pro run-time.

Příklad vlastní výjimky:

```
public class NegativeAgeException extends Exception {
   private int age;

   public NegativeAgeException(int age){
      this.age = age;
   }

   public String toString(){
      return "Věk nemůže nabývat záporných hodnot: " + age ;
   }
}
```

Použití vlastní výjimky:

```
public class CustomExceptionTest {

   public static void main(String[] args) throws Exception{

      int age = getAge();

      if (age < 0){
         throw new NegativeAgeException(age);
      }else{
         System.out.println("Age entered is " + age);
      }
   }

   static int getAge(){
      return -10;
   }
}
```

Generování výjimek

PDF created with pdfFactory trial version www.pdffactory.com

- Mechanizmus výjimek je časově náročný - má se využívat jen při závadách a nikoli pro testy podmínek.
- Deklarované výjimky se objeví ve standardní dokumentaci metod.
- Při jednoduchých pokusech lze obejít nepřehledné try klauzule připsáním *throws Exception* do hlaviček. Jakoukoli výjimku deklarovanou v signatuře není programátor nucen uvnitř metody ošetřovat.

Odchyt a Zpracování výjimek

Ke zpracování výjimek v Javě slouží klauzule try. Ta se skládá nejméně ze dvou bloků. Existují 3 typy bloků:

- *hlídač* - **try** (právě jeden)
- *řešitel* - **catch** (libovolný počet)
- *uklízeč* - **finally** (nanejvýš jeden)

Klauzule try může mít tři tvary:

- *try - catch - finally*
- *try - catch*
- *try - finally*

```
try {
   // ... zde je blok kódu, kde může nastat chyba a chceme ji ošetřit
} catch (Exception1 e) {
   // ošetření výjimky typu Exception1 a všech jejích podtypů
} catch (Exception2 e) {
   // ošetření výjimky typu Exception2 a všech jejích podtypů
} finally {
   // provede se vždy
}
```

Zjednodušeně řečeno:

- Pokud výjimku neodchytí blok, kde nastala, šíří se do následujícího vyššího bloku,
- Pokud není odchycena v metodě, šíří se do volající metody,
- Pokud se dostane až do main() a není odchycena, způsobí ukončení interpretu Javy - vypíšou se informace o výjimce, kde nastala a jak se šířila.

Hlídač - try

Hlídač dozírá, byla-li v bloku vyhozena výjimka (nezáměrně či záměrně):

- **Ne**: Před odchodem z hlídaného bloku předá řízení uklízeči.
- **Ano**: Další příkazy v try bloku se již neprovedou. Hlídač zjistí typ výjimky a procházením řešitelů shora dolů se snaží najít prvního kompetentního řešitele, tj. takového, jehož parametr je typem nebo nadtypem výjimky:
  - *Nalezne-li* ho, předá mu řízení.
  - *Nenalezne-li* řešitele, předá řízení uklízeči a pak nevyřešenou výjimku vyhodí do nadbloku. Je-li tím blokem obalová *try* klauzule postupuje se obdobně. Je-li tím blokem metoda, pak tato metoda vyhodí výjimku do příkazu metody, odkud byla zavolána a tam se postupuje obdobně. Nenalezne-li se žádný kompetentní řešitel, vyhodí se výjimka do obalové klauzule JVM - odtud byla zavolána metoda main(String[ ] args) - pak JVM vypíše hlášení a ukončí běh.

PDF created with pdfFactory trial version www.pdffactory.com

Řešitel - catch

Řešitel má prostřednictvím parametru referenci k aktuální výjimce i dalším proměnným a může situaci trochu napravit anebo alespoň řádně ohlásit. Nelze však nijak zařídit pokračování v nedokončeném hlídaném bloku. I když řešitel vůbec nic neudělá, je odchycená výjimka „sežrána“ - nic jsme neudělali tedy problém není odstraněn.

I řešitelé a uklízeč mohou vyhodit nějakou výjimku - tu však zpracuje (dynamicky) obalová try klauzule. Též lze sestrojit novou výjimku a jako její příčinu vložit tu původní.

Příkaz assert

Během ladění programu se snažíme obvykle na různá místa vkládat kontroly, zda platí určité předem dané podmínky. Například zda se nepokoušíme volat nějakou funkci s nesprávnou hodnotou argumentu. Tyto kontroly pak v okamžiku, kdy jsme zcela přesvědčeni o správnosti svého programu a zajímá nás již jen jeho rychlost, odstraníme nebo případě převedeme na komentáře. Brzy na to po objevení další chyby testy opět obnovujeme. Lepším řešením je použití příkazu **assert**, jenž umožňuje do přeloženého programu zařadit kontrolu, zda je splněna podmínka uvedená za klíčovým slovem assert. Pokud tato podmínka splněna není, vyvolá se výjimka **AssertionError**.

Testovanou podmínku můžeme doplnit po dvojtečce textovým řetězcem(případně libovolným objektem, pro jehož výpis se použije jeho metoda toString.), který se uloží jako popis výjimky. Příkaz **assert** byl do jazyka Java doplněn až ve verzi 1.4. Při spuštění programu se pak v případě, že zadáme parametr **-ea**, provádí kontrola splnění zadaných podmínek. Pokud tento parametr nezadáme, kontrola podmínek se neprovádí a výpočet se tak ničím nezdržuje.

Příklad:

Definujte funkci, která vrátí převrácenou hodnotu nenulového čísla. Ošetřete situaci, kdy je jako argument předána nulová hodnota. S použitím příkazu assert bude řešení včetně testovacího programu následující:

```
class TestAssert {
   static double prevracena_hodnota(double x)
   {
      assert x != 0.0 : "Argument nesmi byt nulovy";
      return 1.0 / x;
   }
   public static void main(String args[]) {
      System.out.println(prevracena_hodnota(1));
      System.out.println(prevracena_hodnota(0));
   }
}
```

Příkaz *assert* je vhodné uvádět na těch místech programu, kde potřebujeme zaručit platnost určité podmínky vyplývající přímo z logiky programu. Příkazem *assert* určitě nebudeme ošetřovat chyby vzniklé činností uživatele, neplatnost uvedené podmínky vždy indikuje chybu v našem programu (nebo chybně stanovenou podmínku, i na to si musíme dát pozor).

Příklad - Čtení ze souboru

Metoda *nactiBajt* otevře soubor zadaného jména (1) a načte z něj první byte, který vypíše na standardní výstup(2).

```
 void nactiBajt(String jmenoSouboru) {
     try {
       FileInputStream soubor = new FileInputStream(jmenoSouboru);   // (1)
       System.out.println(soubor.read());                            // (2)
    } catch(FileNotFoundException e) {                               // (3)
```


PDF created with pdfFactory trial version www.pdffactory.com

```
        System.out.println("Soubor " + jmenoSouboru + "nenalezen.");  // (4)
     } catch(IOException e) {                                         // (5)
        System.out.println("Chyba při čtení souboru " + jmenoSouboru);// (6)
     }
  }
```

Při pokusu otevření souboru se může stát, že tento nebude nalezen (nastane výjimka **FileNotFoundException**), např. pokud soubor neexistuje nebo je chybně zadané jméno. Pokud tento výjimečný stav nastane, je zachycen blokem **catch** na řádce (3) a vypíše se chybová zpráva (4).

Také při čtení může nastat chyba (výjimka **IOException**), např. pokud soubor není určen pro čtení nebo nastane chyba zařízení. Tento stav zachycuje blok **catch** na řádce (5) a vypíše se chybová zpráva (6).

Pokud vše proběhne bez problémů, pokračuje program za záchytnými bloky (metoda skončí).

Slovníček pojmů

Výjimka (obecně) - Událost, která naruší normální běh instrukcí

Výjimka (v Javě) - Objekt, který nese informace o vzniklém problému

Typy výjimek:

- checked (kontrolované) - vyžadují ošetření klauzulí try – catch anebo vyznačení throws
- unchecked (nekontrolované) - nevyžaduje vyznačení ani ošetření

try (hlídač) - Hlídá, jestli byla v bloku vyhozena výjimka

catch (řešitel) - Ošetřuje výjimku daného datového typu

finally (uklízeč) - Nepovinný, pokud je deklarován provede se vždy

throw - Používá se k vyhození výjimky

throws - Příkaz, který se zapisuje v signatuře metody. Za throws následuje výčet datových typů výjimek, které může metoda vyhazovat

assert - Umožňuje zařadit kontrolu, zda je splněna podmínka uvedená za klíčovým slovem assert

JVM - Java Virtual Machine

Zdroje

- Přednášky k předmětu Y36PJV - Programování v Jazyku Java
- Java bez předchozích znalostí
- Exceptions in Java
  - http://www.javaworld.com/javaworld/jw-07-1998/jw-07-exceptions.html [http://www.javaworld.com/javaworld/jw-07-1998/jw-07-exceptions.html]
- Úvod do JAVY, 4.část - exceptions
  - http://www.linuxzone.cz/index.phtml?ids=2&idc=136 [http://www.linuxzone.cz/index.phtml?ids=2&idc=136]

PDF created with pdfFactory trial version www.pdffactory.com

- 13. Výjimky
  - http://v1.dione.zcu.cz/java/sbornik/13.html [http://v1.dione.zcu.cz/java/sbornik/13.html]
- Creating user defined exceptions
  - http://www.javabeat.net/tips/29-creating-user-defined-exceptions.html [http://www.javabeat.net/tips/29-creating-user-defined-exceptions.html]

spolecne/spol30.txt · Poslední úprava: 2010/02/07 16:46 autor: Cimpr

PDF created with pdfFactory trial version www.pdffactory.com