Výjimky

November 12, 2014

1 Výjimky

1.1 K čemu jsou výjimky

- Výjimky jsou mechanizmem, umožňujícím reagovat na (řešit) nestandardní (=chybné) běhové chování programu, které může mít různé příčiny – chyba okolí (uživatele, systému) i vnitřní chyba programu (tedy programátora).
- Výjimky jsou mechanizmem, jak psát robustní, spolehlivé programy odolné proti chybám "okolí" i chybám v samotném programu.
- Výjimky v Javě fungují podobně jako v C++, C#, Ruby, Pythonu a dalších zejména objektových jazycích.
- v Javě jsou ještě lépe implementovány než v C++ (navíc klauzule `finally`)
- Podobně jako jinde, ani v Javě není dobré výjimkami potlačovat chyby *programu* samotného – to je hrubé zneužití.

1.2 Co technicky jsou výjimky

- *Výjimka* (*Exception*) je objekt třídy `java.lang.Exception`, příbuzným typem jsou rovněž *vážné běhové chyby* - ty jsou objekty třídy `java.lang.Error`.
- Každopádně v obou případech rozšiřuje třídu `java.lang.Throwable`, která je také často používaná jako deklarativní typ pro výjimky v širším slova smyslu.

1.3 Kdy výjimka vznikne

Objekty – *výjimky* – jsou vyhozeny (throw) buďto:

- automaticky běhovým systémem Javy, nastane-li nějaká běhová chyba – např. dělení nulou, nebo
- jsou vytvořeny a "vyhozeny" samotným programem, zdetekuje-li nějaký chybový stav, na nějž je třeba reagovat – např. do metody je předán špatný argument

1.4 Kdy výjimka vznikne – příklad

```
try {
// First, the exception can be raised in JVM when
// dereferencing using a bad index in an array:
   String name = args[i]; // ArrayIndexOutOfBoundsException

// or the exception(s) can be raised by programmer
// in the constructor of Person:
   p = new Person(name); // NullPointerException or IllegalArgumentException

} catch(ArrayIndexOutOfBoundsException e) {
   System.err.println("No valid person name specified");
   System.exit(1);
} catch(NullPointerException e) {
// reaction upon NullPointerException occurence
} catch(IllegalArgumentException e) {
// reaction upon IllegalArgumentException occurence
}
```

1.5 Co se s vyhozenou výjimkou stane

Vyhozený objekt výjimky je buďto:

- tzv. zachycen v rámci metody, kde výjimka vznikla – do bloku `catch` → výjimka je v bloku `catch` tzv. **zachycena**. Takto to fungovalo ve výše uvedeném příkladu – všechny výjimky byly zachyceny a řešeny svými bloky catch.
- výjimka "propadne" do nadřazené (volající) metody, kde je buďto v bloku `catch` zachycena nebo opět propadne atd.

Výjimka tedy "putuje programem" tak dlouho, než je zachycena. Pokud není nikde zachycena, běh JVM skončí s hlášením o výjimce.

1.6 Syntaxe kódu s ošetřením výjimek

Základní syntaxe:

```
try {
   //zde může vzniknout výjimka | an exception may occur here

} catch (ExceptionType exceptionVariable) {
   // zde je výjimka ošetřena | the exception is caught here and reacted upon
   // je možné zde přistupovat k "exceptionVariable"
   // the attributes from the exceptionVariable can be accessed here
}
```

Bloku `try` se říká *hlídaný blok*, protože výjimky příslušného typu uvedeného v hlavičce/-ách bloků catch zde vzniklé jsou zachyceny.

1.7 Reakce na výjimku – možnosti

Jak můžeme na vyhozenou výjimku reagovat? Technicky je to jasné – buďto výjimku zachytit nebo propustit výše. Co a jak ale kdy udělat?

1. Napravit příčiny vzniku chybového stavu – např. znovu nechat načíst vstup.
2. Poskytnout za chybný vstup náhradu – např. implicitní hodnotu.
3. Operaci neprovést (vzdát) a sdělit chybu výše tím, že výjimku propustíme z metody.

1.8 Reakce na výjimku – pravidlo č. 1

1. Vždy nějak reagujme. Výjimku neignorujme, nepotlačujme, tj.
2. blok `catch`*nenechávejme prázdný*, přinejmenším vypišme `e.printStackTrace()`
3. Nelze-li rozumně reagovat na místě, propusťme výjimku výše a popišme to v dokumentaci – Příklad komplexní reakce na výjimku (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java`)

1.9 Kaskády bloků catch

V některých blocích `try` mohou vzniknout *výjimky více typů*:

- pak můžeme bloky `catch` řetězit, viz přechozí příklad: Příklad komplexní reakce na výjimku (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyZachytZopakuj.java`)
- Pokud `catch` řetězíme, musíme respektovat, že výjimka je zachycena nejbližším příhodným `catch`
- Pozor na řetězení `catch` s výjimkami typů z jedné hierarchie tříd: pak musí být výjimka z podtřídy (tj. speciálnější) uvedena a tedy zachycována dříve než výjimka obecnější: Takto ne! (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/KaskadaVyjimekSpatne.java`)

1.10 Výjimky vs. jejich hlídání překladačem

- Java je staticky typovaný jazyk a jako takový se snaží hlídat místa potenciálního vyhození výjimky.
- V dosud uváděných příkladech se neprojevovalo, že by nás nějak hlídal.
- Bylo to proto, že jsme dosud nepoužívali tzv. hlídané výjimky, jejichž místa vzniku překladač sleduje a hlídá, jak na ně reagujeme.
- Nyní nastíníme úplnou kategorizaci výjimek vč. hlídaných.

1.11 Kategorizace výjimek a dalších chybových objektů

- Všechny objekty výjimek a chybových stavů implementují rozhraní java.lang.Throwable – "vyhoditelný"
- Nejčastěji se používají tzv. *hlídané výjimky* (checked exceptions), což jsou potomci/instance třídy java.lang.Exception
- Dosud zmiňované byly tzv. *běhové (runtime, nebo též nehlídané, unchecked) výjimky* jsou typu/typu potomka -→ java.lang.RuntimeException – takové výjimky nemusejí být zachytávány
- *Vážné chyby JVM* (potomci/instance java.lang.Error) – obvykle signalizují těžce napravitelné chyby v JVM - např. *Out Of Memory*, *Stack Overflow...*, ale též např. chybu programátora: AssertionError

1.12 Použít hlídanou nebo nehlídanou výjimku?

- Unchecked exceptions: represent defects in the program (bugs) – often invalid arguments passed to a non-private method.Gosling, Arnold, and Holmes: *"Unchecked runtime exceptions represent conditions that, generally speaking, reflect errors in your program's logic and cannot be reasonably recovered from at run time."*
- Checked exceptions: represent invalid conditions in areas outside the immediate control of the program (invalid user input, database problems, network outages, absent files)

1.13 Syntaxe metody propouštějící výjimku

Pokud výjimka nikde v těle nemůže vzniknout, překladač to zdetekuje a vypíše:`...Exception XXX is never thrown in YYY ...`Příklad s propouštěnou výjimkou – Otevření souboru s propouštěnou výjimkou (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/OtevreniSouboru2.java`)

```
modifikatory návratovýTyp nazevMetody(argumenty) throws TypPropouštěnéVýjimky {
   ... tělo metody, kde může výjimka vzniknout ...
}
```

1.14 Vlastní hierarchie výjimek

- Typy (=třídy) výjimek si můžeme definovat sami, např. viz - Výjimky ve světě chovatelství (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/svet.html`)
- bývá zvykem končit názvy tříd - výjimek - na *Exception*

1.15 Klauzule finally

Klauzule (blok) *finally*:

- Může následovat ihned po bloku `try` nebo až po blocích `catch`
- Slouží k "úklidu v každém případě", tj.

  - když je výjimka *zachycena* blokem `catch`
  - i když je výjimka *propuštěna* do volající metody
- Používá se typicky pro uvolnění systémových zdrojů - uzavření souborů, soketů...

Příklad: Úklid se musí provést v každém případě... (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky/VyjimkyFinally.java`)

1.16 Odkazy

- Oracle Java Tutorials – Lesson: Handling Errors with Exceptions (`https://docs.oracle.com/javase/tutorial/essential/exceptions/index.html`)
- demo programy z učebnice – Výjimky (`http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/vyjimky`)