20. Výjimky

V Javě je vyjímka představována objektem popisujícím nějaký výjimečný (chybový) stav, který nastal v nějaké části kódu. Tento objekt je posléze vrácen metodě, která chybu způsobila. Daná metoda se pak může rozhodnout, zda výjimku zpracuje sama, anebo ji může předat dále. Bez ohledu na to ale platí, že v určité chvíli musí dojít k zachycení výjimky a jejímu zpracování.Výjimky můžou být generovány běhovým prostředím Javy anebo mohou být ručně generovány našim kódem. Výjimky vyvolané běhovým prostředím Javy obvykle představují základní výjimky popisující stav, kdy došlo k porušení pravidel jazyka Java anebo k narušení omezení běhového prostředí Javy. Ručně generované výjimky se pak typicky používají k předání informací o nějakém chybovém stavu tomu kódu, který danou metodu zavolal.

Pro zpracování výjimek se v Javě využívá celkem 5 klíčových slov: **try**, **catch**, **throw**, **throws** a **finally**. Ta část programu, v níž chceme hlídat výsky chyb či výjimek by měla být obsažena v bloku try. Pokud v tomto bloku vznikne nějaká výjimka, je tzv. vyvolána. Náš kód pak může takovou výjimku zachytit (pomocí klíčového catch) a nějakým rozumným způsobem jí zpracovat. Systémem generované výjimky jsou automaticky vyvolávány běhovým prostředím Javy. Chceme-li vyvolat výjimku ručně, použijeme klíčové slovo **throw**. Jakákoliv výjimka, která je metodou vyvolávána a předávána dále, musí být také patřičným způsobem specifikována klíčovým **throws**. Jakýkoliv úsek kódu, který musí být bezpodmínečně proveden po dokončení kódu v bloku try musí být uložen do bloku **finally**. Obecně blok kódu umožňující zpracování výjimek vypadá takto:

try {

//kód v němž chceme hlídat výskyt chyb

}

catch(TypVýjimky1 exOb){

//kód zpracovávající výjimku typu TypVýjimky1

}

catch(TypVýjimky2 exOb){

//kód zpracovávající výjimku typu TypVýjimky2

}

finally {

//blok kódu, který má být (a bude, nezávisle na výsledku) proveden po dokončení bloku try

}

## Typy výjimek

Typů výjimek je potom celá řada. Všechny tyto typy jsou podtřídami standardní třídy **Throwable**. Pod touto třídou se nacházejí dvě podtřídy, které výjimky rozdělují do dvou odlišných větví.

Na vrcholu jedné z nich se nachází třída **Exception** - výjimka. Tato třída se používá pro výjimečné stavy, které by uživatelské programy měly zachytávat. Podtřídy třídy Exception se navíc využívají při tvorbě vlastních typů podmínek. Nejvýznamnější podtřídou třídy Exception je pak **RuntimeException** - výjimky tohoto typu jsou automaticky definovány pro všechny Java programy a patří mezi ně i výjimky například pro dělení nulou či neplatné indexování pole.

Na vrcholu druhé větve se pak nachází třída **Error**. Ta definuje výjimky, u nichž není předpoklad, že budou za běžných okolností programem zachytávány. Výjimky tohoto typu jsou využívány běhovým prostředím Javy k indikaci chyb souvisejících přímo se samotným běhovým prostředím. Typickým příkladem takové chyby je pak například přetečení zásobníku.

## Nezachycené výjimky

Pokud v programu nedojde ke zpracování výjimky samotným uživatelem, postará se o toto zpracování samotné běhové prostředí jazyka Java. Na následujícím příkladě si ukažme, co se stane, pokud neošetříme dělení nulou:

class Deleni0 {

public static void main(String[]Args){

int d = 0;

int a = 88/d;

}

}

Když běhové prostředí Javy narazí na pokus o dělení nulou, vytvoří nový objekt výjimky a vzápětí tuto výjimku vyvolá. V důsledku toho se zastaví celý běh programu Deleni0, protože platí, že jakmile je nějaká výjimka vyvolána, musí být zachycena kódem pro řešení výjimek a okamžitě řešena. V tomto případě však neexistuje žádné uživatelské řešení, takže je tato výjimka zachycena standardním kodem pro obsluhu výjimek, který je součástí běhového prostředí Javy. Tento kód zobrazí řetězec popisující danou výjimku, dále zobrazí i stack trace od místa, v níž se chyba vyskytla a následně ukončí běh programu. Takže chyba vygenerovaná tímto kódem by pro náš příklad vypadala takto:

Exception in thread “main” java.lang.ArithmeticalException: / by zero

at Deleni0.main(Deleni0.java:4)

Tento stack trace obsahuje nízev třídy (Deleni0), název metody (main), jméno zdrojového souboru (Deleni0.java) a číslo chybového řádku (4). Vyvolaná výjimka je podtřídou třídy Exception zvanou ArithmeticalException. Ta vlastně blíže popisuje typ chyby, která se vyskytla. Platí, že stack trace bude vždy zobrazovat pořadí volání metod vedoucí k výskytu chyby.

## Try a catch

Stack trace z předchozího příkladu je sice užitečná pro programátora, ovšem standardní ošetření chyb přímo Javou má dvě obrovské nevýhody. Za prvé neumožňuje danou chybu okamžitě napravit a za druhé program automaticky ukončí. Většina běžných uživatelů by pak byla přinejmenším zmatena, kdyby při výskytu jakékoliv chyby došlo k ukončení běhu programu a k výpisu stack trace. Naštěstí se dá automatickému ukončení programu předejít celkem snadno - pomocí bloků try a catch, které umožní zpracování běhových chyb. Ošetření předchozí výjimky (dělení nulou) by tedy díky blokům try a catch vypadalo takto:   
  
class Deleni0 {

public static void main(String[]Args){

int d = 0;

try {

a = 88 / d;

System.out.println(“Tohle se vůbec nezobrazí”);

}

catch (ArithmeticException e){

System.out.println(“Nelze dělit nulou.”);

}

System.out.println(“Tohle se provede až po bloku catch - tedy po zpracování výjimky.”);

}

}

Výstup tohoto programu už nebude spočívat v “pouhém” zobrazení stack trace, ale bude vypadat takto:

Nelze dělit nulou.  
 Tohle se provede až po bloku catch - tedy po zpracování výjimky.

Příkaz println(), jenž je součástí bloku try se skutečně vůbec neprovede. Jakmile je totiž vyvolána výjimka, řízení programu je ihned předáno z bloku try do bloku catch.