12. Výjimky

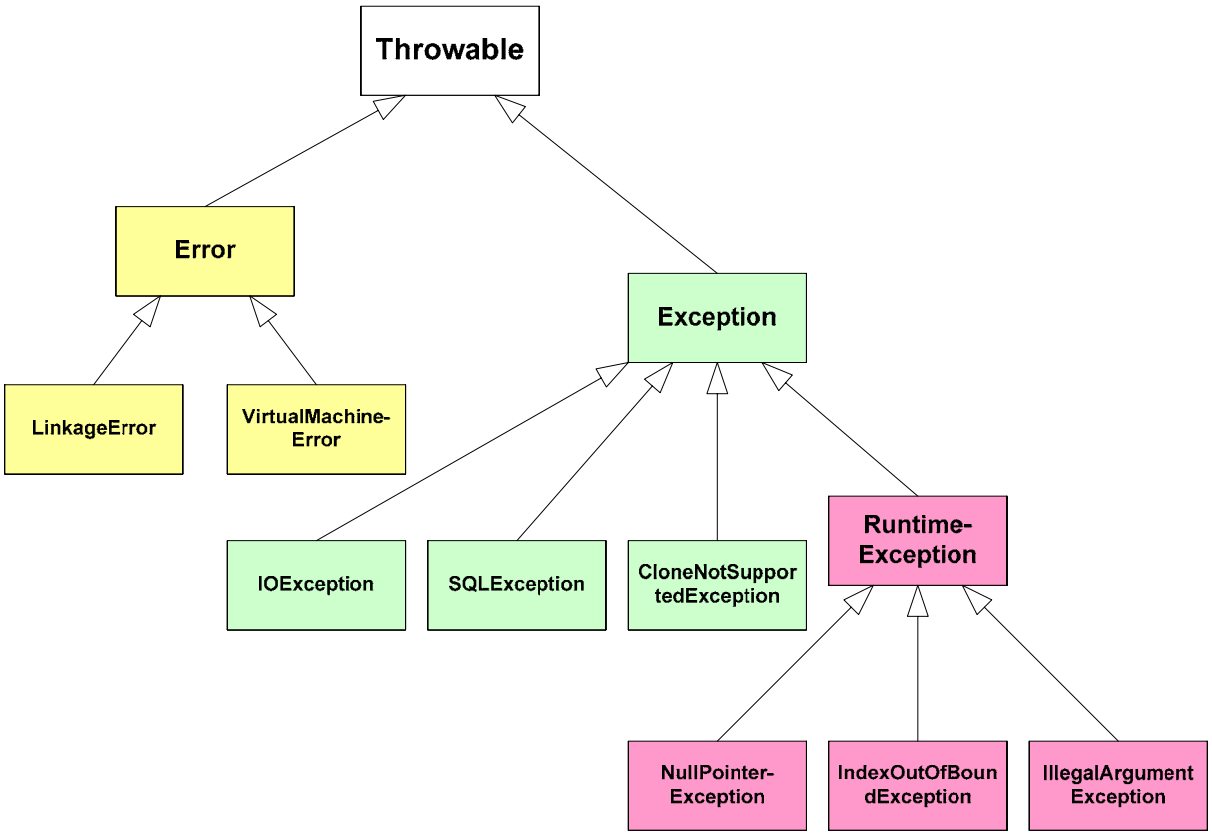
Při vytváření programu je třeba vždy zohlednit to, že za běhu programu se mohou objevit chyby, které je třeba ošetřit. Například uživatel se snaží otevřít neexistující soubor pro čtení a nebo místo čísla vloží na vstupu text.
S chybovými stavy se lze vypořádat dvěmi způsoby:

- Testováním splnění podmínek před vyvoláním kódu. Např. lze zjistit, zda soubor existuje před tím, než se bude otevírat pro čtení, lze zjistit, zda uživatel zadal pouze číslice. V některých situacích nelze tento postup použít – např. když v průběhu čtení souboru havaruje disk.
- Přes mechanismus výjimek – kód se napíše s vírou ve splnění podmínek, pokud v průběhu nastane chyba, tak se ošetří pomocí mechanismu výjimek.

Volba příslušného způsobu ošetření závisí na typu chyby, na pravděpodobnosti vzniku chybového stavu, na nákladech spojených s příslušnými typy ošetření výjimky. Použití mechanismu výjimek vede obvykle k přehlednějšímu kódu, někdy však za cenu snížení rychlosti algoritmu.

12.1. Druhy výjimek

Když v programu dojde k chybě, vznikne výjimka – přeruší se zpracování programu, vytvoří se objekt s informacemi o chybě a pokračuje se na místě, na které uživatel umístil kód s ošetřením chyby.
Předkem pro všechny výjimky je třída `Throwable` a jejími potomky třídy `Error` a `Exception`.
Java také umožňuje tvorbu vlastních výjimek. Hierarchie tříd pro výjimky je zachycena na obrázku 12.1.
Se třídou `Throwable` se přímo nepracuje, poskytuje totiž velmi obecnou informaci, že nastala nějaká chyba. Třída `Error` reprezentuje chyby systému Javy (např. nedostatek paměti – `OutOfMemoryError`, přetečení zásobníku – `StackOverflowError`, chyby v souboru class – `ClassFormatError`, `ClassCircularityError`, `VerifyError`, `NoClassDefFoundError`), v programu se obvykle neošetřují (některé ani nelze ošetřit).
Třída `Exception` a její potomci, mimo větev `RuntimeException`, jsou označováni jako kontrolované (synchronizované) výjimky, u kterých překladač kontroluje, zda je ošetřujeme. Mezi typické zástupce patří chyby vzniklé při práci se vstupy a výstupy. Na možnost výskytu výjimky musíme v kódu nějak reagovat (viz dále), jinak překladač vypíše upozornění a kód nepřeloží.
Třída `RuntimeException` a její potomci reprezentují chyby, na které lze také úspěšně reagovat, ale u kterých není překladačem vyžadováno jejich ošetření. Jsou to například `ArithmeticException`, `ArrayIndexOutOfBoundsException`, `NullPointerException` nebo `NumberFormatException`. V mnoha případech může programátor předejít těmto výjimkám vhodným testováním parametrů či pečlivostí při programování.



Obrázek 12.1 Hierarchie výjimek s příklady v jednotlivých skupinách

12.2. Vyvolání výjimky

Výjimky mohou vznikat pouze za následujících situací:

- chybný stav aplikace detekovaný virtuálním strojem, sem patří:
  - chybný výraz, např. celočíselné dělení nulou,
  - chyby při natažení tříd do paměti,
  - nedostatečné zdroje, např. nedostatek paměti,
  - vnitřní chyba virtuálního stroje,
- vyvoláním výjimky příkazem *`throw`* – nejčastější případ.

Při psaní svých metod můžeme vyvolat výjimku příkazem *`throw`*, což může vypadat následovně:

```
if (promenna == null) {
     throw new NullPointerException();
}
```

Vyvoláním výjimky se přeruší provádění metody. Při vyvolání výjimky můžeme předat jako parametr i podrobnější popis výjimky:

```
if (promenna == null) {
     throw new NullPointerException("promenna = null");
}
```

12.3. Ošetření výjimky

Java poskytuje dva způsoby práce se vzniklou výjimkou:

- zachycení a ošetření (jazyková konstrukce `try … catch`),
- předání výjimky výše (`throws`) – týká se kontrolovaných výjimek, pokud jejich ošetření chceme předat do volající metody.

Při vyvolání výjimky hledá JVM nejbližší úsek kódu ošetřující příslušnou výjimku – JVM jde v protisměru volání jednotlivých metod, v každé metodě hledá, zda volání bylo v bloku `try` a zda je u bloku ošetření tohoto typu výjimky nebo je uveden předek této výjimky. Pokud nikde ošetření tohoto typu výjimky nenajde, použije standardní ošetření, ve kterém se vypíší informace o výjimce a aplikace skončí. Následuje ukázka standardního chybového hlášení, na kterém je vidět typ výjimky a sekvence volání jednotlivých metod (ukázka je z projektu `Skola`).

```
java.lang.NullPointerException
        at Osoba.hashCode(Osoba.java:91)
        at java.util.HashMap.hash(HashMap.java:264)
        at java.util.HashMap.put(HashMap.java:382)
        at java.util.HashSet.add(HashSet.java:194)
        at Utvar.pridej(Utvar.java:49)
        at Skola.<init>(Skola.java:49)
        at Skola.main(Skola.java:64)
```

Vznikla výjimka `NullPointerException` a to v metodě `hashCode()` ve třídě `Osoba` (konkrétně na řádku 91 zdrojového kódu `Osoba.java`). Tato metoda byla volána z metody `hash()` ve třídě `HashMap`, která byla volána z metody `put()` ve třídě `HashMap`, atd. Na předposledním řádku se odkazuje výraz `<init>` na konstruktor. Tato chyba vznikla z toho důvodu, že místo jména pracovníka je napsána konstanta `null`.

U některých výjimek (většinou potomků třídy `Error`) může být popis chyby kratší. Následující příklad ukazuje chybové hlášení při pokusu o spuštění neexistující třídy.

```
C:\>java NeexistujiciTrida
Exception in thread "main" java.lang.NoClassDefFoundError: NeexistujiciTrida
```

12.3.1. Odchytávání výjimek (try catch)

Pokud chceme výjimku ošetřit v metodě sami, uzavřeme část kódu, ve kterém může vzniknout chyba, do bloku uvedeného slovem **try**. V rámci bloku `try` by měly být uvedeny příkazy, které odpovídají správnému průběhu programu. Po tomto bloku následuje alespoň jeden blok začínající klíčovým slovem **catch** v závorce následovaný typem (jménem třídy) výjimky a jménem proměnné, ve které budou uloženy informace o chybě (obvykle se používá jméno proměnné `e`)[27]. Těchto bloků `catch` může být více. Platí však, že jednotlivé bloky jsou procházeny postupně a první, který vyhovuje (tj. výjimka, která nastala, je instancí uvedené třídy nebo jejího potomka), tuto výjimku ošetří. Výjimky je tedy třeba při odchytávání uvádět v pořadí od konkrétních k obecným (od potomků k předkům). Jako poslední může být v `try catch` uveden blok **finally**, příkazy uvedené v tomto bloku jsou provedeny vždy, tj.:

- když k žádné výjimce nedojde,
- když je vyhozena výjimka a je zpracována v některém bloku `catch`,
- když vznikne výjimka, která není zpracována v blocích `catch`,
- když vznikne výjimka v některém bloku `catch`.

[27] Deklarace bloku `catch` částečně připomíná deklaraci metody. Formální parametr (obvykle se používá identifikátor `e`) má platnost v rámci následujícího bloku.

Následuje formální specifikace bloku *try catch*:

```
try {
     blok chráněných příkazů
}
catch (xxxException e) {
     reakce na výjimku daného typu
}
catch (......
.......
finally {
     ukončovací příkazy
}
```

Proměnná uvedená jako formální parametr (obvykle *e*) v bloku *catch* obsahuje odkaz na konkrétní výjimku. U všech výjimek lze zjistit popis chyby či vypsat popis včetně odkazu na příslušné řádky zdrojového kódu, kde k výjimce došlo. Požadované informace zjistíme pomocí metod deklarovaných ve společném předkovi všech chyb a výjimek třídě *Throwable* (některé výjimky mají další metody a informace). Jejich přehled je uveden v následující tabulce.

metoda	**popis**
String **getMessage**()	vrátí popis chyby (může být prázdný)
String **toString**()	vrátí jméno třídy chyby následované popisem (výsledek metody *getMessage()*)
StackTrace[] **getStackTrace()**	vrací zásobník s popisem průběhu vzniku a posílání výjimky, každá položka odpovídá jednomu řádku ve volací sekvenci volání metod
void **printStackTrace()**	vypíše výsledek metody *getStackTrace()* na *System.err* (standardně na konzolu)

Tabulka 12.1 Přehled informativních metod třídy Throwable

Použití odchycení a zpracování výjimky si ukážeme na následujícím příkladě. Jedná se o metodu *getInt()* ze třídy *CteniZKonzole* z projektu Trojúhelníky v kapitole 17. Tato metoda má přečíst z konzole celé číslo zadané uživatelem. Je zde použit nekonečný cyklus (řádek 4), který ukončí příkaz *return* v případě, že uživatelem zadaná hodnota je celé číslo. Uvnitř cyklu je v bloku *try* (řádky 5 až 11) uveden kód představující optimální průběh (uživatel zadá celé číslo). Řádek 6 kódu zapouzdří vstup z konzole – bude vysvětleno v kapitole 13 o vstupních a výstupních proudech. Řádek 7 obsahuje příkaz pro vypsání promptu na konzoli a na řádku 8 je do proměnné *radek* načtena uživatelem zapsaná hodnotu. Kód na řádcích 6 a 8 může vyvolat výjimku typu *IOException*, která patří mezi kontrolované výjimky (podrobnosti o *IOException* viz kapitola 13). Musí tedy být uveden blok *catch*, který tento typ výjimek odchytí a zpracuje (viz řádky 12 až 14). Pravděpodobnost výskytu chyby vstupu při čtení z konzole je velmi malá, použijeme tedy jen jednoduchý výpis pomocí *System.out.println(e)* – vypíše se popis výjimky vráceny metodou *toString()*. Na řádku 9 je načtený řetězec převáděn na celé číslo typu *int*. Pokud uživatel zadá řetězec, který nelze na celé číslo převést, bude metodou *parseInt()* vyhozena výjimka typu *NumberFormatException*. Výjimka tohoto typu je potomkem *RuntimeException*, takže blok *catch* s tímto typem výjimky není překladačem vyžadován. Na řádcích 15 až 17 je uvedeno ošetření tohoto typu výjimky.

```
 1  public int getInt(String prompt) {
 2    int cislo = 0;
 3    String radek;
 4    while (true) {
 5    try {
 6       BufferedReader vstup = new BufferedReader
                  (new InputStreamReader(System.in));
 7       System.out.print(prompt+": ");
 8       radek = vstup.readLine();
 9       cislo = Integer.parseInt(radek);
10       return cislo;
11    }
12    catch (IOException e) {
13       System.out.println(e);
14    }
15    catch (NumberFormatException e) {
16       System.out.println("chyba: nebylo vloženo celé číslo");
17    }
18    }
19  }
```

Doporučujeme nepoužívat prázdný blok `catch`, který výjimku zachytí a „zahodí“. Tím se ztratí potřebné informaci pro odhalení chyby. Také není vhodné odchytávat všechny výjimky jedním blokem `catch`, ve kterém budeme mít uvedenu třídu `Exception`. Lepší je uvést jednotlivé výjimky, neboť nám to umožňuje lépe ošetřit jednotlivé stavy.

12.3.2. Throws

Pokud tvoříme metodu, ve které může dojít ke kontrolované výjimce, a my ji v rámci této metody nechceme nebo neumíme ošetřit, musíme překladači explicitně sdělit, že ji předáváme k ošetření do nadřazené úrovně, tj. metodě, která naši metodu vyvolala. Toho dosáhneme tím, že v hlavičce metody použijeme klíčové slovo **throws**[28] a třídu výjimky (popřípadě více tříd). V následujícím příkladu metoda předává výše výjimku (a její potomky), která může vzniknout při čtení ze souboru:

```
public String ctiVetu ( ) throws IOException {
  ...
}
```

Je vhodné uvádět `throws` i pro nekontrolované výjimky, neboť poté při čtení kódu je hned vidět, že může vzniknout výjimka. V každém případě by měly být výjimky popsány v komentáři k metodě.

12.4. Často používané typy výjimek

Nejčastěji používané typy výjimek jsou uvedeny v tabulce 12.2. S výjimkou posledních dvou jsou všechny potomkem `RuntimeException`, tj. nemusí se povinně odchytávat.

[28] Slovo `throws` se do češtiny často překládá jako odmítání a z tohoto důvodu se často používá termín **odmítnutí výjimky** – nám připadá vhodnější termín předání výjimky výš.

výjimka	**popis**
IllegalArgumentException	Používá se v situaci, kdy byla metoda zavolána se "špatným" parametrem. Použijeme ji například při vytváření čtverce – při zadání záporné délky strany vyvoláme tuto výjimku.
NullPointerException	Vzniká v situaci, kdy identifikátor neobsahuje odkaz na instanci, ale konstantu `null`. Příklady situací: ♦ je volána metoda instance, ale identifikátor obsahuje hodnotu `null`, ♦ přistupuje se k datovému atributu instance, ale identifikátor má hodnotu `null`, ♦ přistupuje se k proměnné `length` pole, které ještě nebylo vytvořeno.
IllegalStateException	Používá se v situaci, kdy je zavolána přípustná metoda, ale instance je ve stavu, kdy metodu nelze provést. Např. pokud se po zavření souboru objeví požadavek na čtení věty, vznikne tato výjimka.
NumberFormatException	Vzniká při převodu řetězce na číslo a vstupní řetězec nelze na číslo převést. Např. při převodu následujících řetězců na int: "45gt" nebo "4.8".
ArrayIndexOutOfBoundException	Zadaný index je mimo rozsah pole, blíže je popsáno v kapitole 10.
ClassCastException	Chyba při přetypování instancí, tato výjimka je popsána v kapitole 11.
CloneNotSupported	Třída nepodporuje vytváření kopií instance, viz popis metody `clone()` v kapitole 6.
FileNotFoundException	Soubor neexistuje či nelze vytvořit, bližší popis je v kapitole 13.
IOException	Chyba vstupu/výstupu, předek více podrobných výjimek, např. `FileNotFoundException`, blíže viz kapitola 13.

Tabulka 12.2 Přehled nejčastěji používaných výjimek