Výjimky

Výjimky samotné jsou objekty, které dědí ze speciální hierarchie, jejímž kořenem je třída Throwable. Throwable dále rozšiřují třídy Error a Exception.

Error

Výjimky typu Error bychom neměli v žádném případě ošetřovat – značí totiž kritickou chybu (nedostatech zdrojů pro práci virtuálního stroje, přetečení zásobníku, nenalezení potřebné třídy při classloadingu atp.) – a většině případů je ani ošetřit nemůžeme.

Runtime exception

Mezi výjimky dědící z Exception patří také podtřída RuntimeException. Runtime exception jsou výjimky, které sice nejsou kritické z hlediska samotné možnosti pokračování aplikace (na rozdíl od Error), ale přesto se velmi často neošetřují. Značí totiž obvykle chyby, které způsobil sám programátor (neplatný index pole, volání nad nullovým ukazatelem...).

Zvláštní vlastností těchto výjimek je, že nemusíme deklarovat v hlavičce metody možnost jejich vyvolání (klíčové slovo throws). Tím je usnadněno vybublání chyby skrz program a jeho případné ukončení.

Exception

Zbylé výjimky dědící přímo z Exception značí ty situace, které by se sice neměly stávat, ale na které jsme schopni adekvátně zareagovat. Kód se nejdříve vyzkouší v bloku try a pokud se vyskytnou výjimky tak jsou podle zadaných výjimek v blocích catch následně zpracovány.

Důsledek nezachycené výjimky

Pokud výjimku nezachytíme, dojde k vypsání tzv “stack trace“. Jedná se o výpis volání metod na systémovém zásobníku v okamžiku, kdy došlo k výjimce. Tento výstup nám bude významně pomáhat při odstraňování chyb v aplikaci.

Ve vývojových prostředích můžeme obvykle na jednotlivé záznamy kliknout a dostat se tak přímo na jednotlivé řádky kódu, které byly volány v okamžiku vzniku výjimky.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at HelloWorld.main(HelloWorld.java:10)
```

Try, catch, finally

Try

Je blok, kam uvádíme kód, ve kterém dojde nebo může dojít k výjimce. Pokud kód v tomto bloku proběhne bez výjimek program pokračuje dál, pokud ne program přejde do bloků catch, kde by měli být všechny výjimky řádně ošetřeny.

(try with resources) = ulehčení při práci se soubory, stará se samo o vytvoření bloku finally a uzavření souboru.

Catch

Jsou bloky kódu, které by měly ošetřovat výjimky, které mohou nastat v kódu bloku try. Tyto bloky by výjimky měly ošetřovat výjimky od té nejspecifičtější po tu nejméně specifickou tedy obecnou Exception, která se postará o zachycení nečekané výjimky nebo výjimky, kterou nejsme schopni ošetřit.

Finally

Tento blok se nachází až na konci, po všech blocích catch. Jedná se o blok, který se provede bez ohledu na to, jestli program zachytí výjimku nebo proces proběhne bez výjimek. Finally blok proběhne vždy po tom co proběhne blok try a catch.

Throws

Klauzule, sloužící k označení metod, které mohou vyhazovat výjimky. Slouží k uspořádání kódu, protože při větším objemu metod bychom museli ošetřovat každou výjimku zvlášť. Kód uspořádáme poté při volání samotných metod v průběhu programu, třeba i ve větším množství a ušetříme si tak práci.

Vícevláknové programování (skládá se ze 2 a více vláken)

Vícevláknové programování umožňuje provádět několik činností v programu paralelně, pokud k tomu program bude mít dostatek prostředků. Průběh programu se tak může výrazně urychlit, protože program může při nenáročných činnostech dělat více činností, třeba ukládat data do databáze nebo provádět výpočty potřebné pro další chod programu.

Vlákno v Javě můžeme vytvořit dvěma způsoby. Děděním třídy Thread a nebo Implementací rozhraní Runnable. Tyto dva přístupy jsou si rovnocenné a záleží na každém programátorovi, který z nich si vybere. Kód poté běží v metodě run(), kterou naše třída s vytvořeným vláknem přepisuje.(polymorfismus)

Daemon

Je vlákno, které je schopno vykonávat potřebné procesy na pozadí a to i když program neběží. Konec programu tedy není závislý na ukončení procesu tohoto vlákna. Příkladem může být garbage collection(proces, který se stará o odstraňování objektů, které v programu již nejsou potřebné). Vlákno můžeme nastavit jako daemon pomocí Thread.setDaemon(boolean) a můžeme ho tak nastavit pouze před jeho spuštěním. Pokud toto provedeme po spuštění, je vyvolána výjimka IllegalThreadStateException.

Komunikace mezi vlákny

Vlánka mezi sebou můžou komunikovat pomocí různých metod například pro zjištění stavu vlánka(běží, neběží) atd.

IsAlive()

Metoda, která vrací boolean(true/false), odkazující na chod vlákna. Pokud vlákno běží vratí true, pokud ne vrátí false.

```
public static void main(String[] args) throws InterruptedException {
    System.out.println("Hlavní vlákno spuštěno");
    Vlakno mojeVlakno = new Vlakno("Druhe");
    mojeVlakno.start();
    while(mojeVlakno.isAlive()) {
        Thread.sleep(1);
    }
    System.out.println("Hlavní vlákno ukončeno");
}
```

Join()

Jedná se o metodu, která zajišťuje to, že vlákno, ve kterém je zavolána, počká, než doběhne vlákno, na které je metoda zavolána a až poté se ukončí.

```
System.out.println("Hlavní vlákno spuštěno");
Vlakno mojeVlakno = new Vlakno("Druhe");
mojeVlakno.start();
mojeVlakno.join();
System.out.println("Hlavní vlákno ukončeno");
```

Wait()

Metoda, která slouží k pozastavení vlákna. Činnost vlákna není zrušena, ale vlákno pouze čeká než je upozorněno na to, že má pokračovat v činnosti.

Notify()

Metoda, která zajišťuje upozornění vlákna. Pokud vlákno čeká na tento příkaz jeho činnost se obnoví.

Synchronizace

Deklaruje se pomocí klíčového slova synchronized. Slouží k omezení přístupu k metodě nebo bloku kódu pouze pro vlákno, které se k bloku kódu dostane jako první. Pokud tento blok kódu používá jedno vlákno, ostatní k němu nemůžou přistoupit dokud první vlákno svou práci nedokončí.