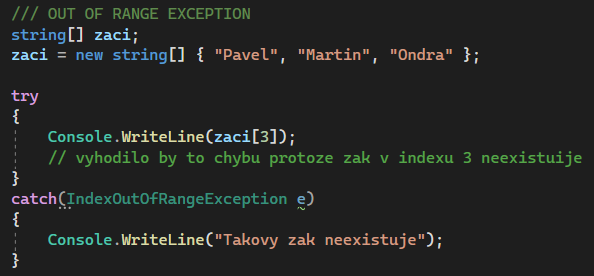
**Výjimky a aserce, debuggování a zpracování chyb**

**Co je to výjimka? (EXCEPTION)**

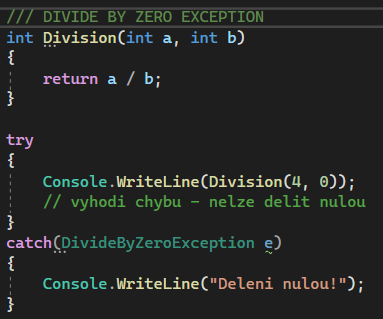
* Výjimka se „vyhodí“, když v průběhu běhu programu se objeví chyba nebo nečekaná událost
  + Pokud není chyba správně chycena a zpracována
    - Chyba znemožní další běh programu a program spadne – vyhodí výjimku v IDE
* Může nastat chyba v HW
  + NEMŮŽEME REAGOVAT
* Pro výjimky se dělají mechanismy pro ošetření chybových stavů – **Zpracování chyb**
* **TRY**
  + Oddělení místa OD až DO, kde bude chyba zachycena
* **CATCH**
  + Zpracování chyby
* Při vzniku chyby je automaticky vytvořen objekt, který nese informace o vzniklé výjimce
* Objevení chyby a její zachycení nemusí znamenat ukončení programu
  + Chybu lze zpracovat a navést na svůj předpokládaný běh programu
  + Různé chyby lze různě zpracovat
  + Příklady chyb v C#
    - Indexace mimo meze pole (OutOfRangeException)
    - Neexistující soubor (FileNotFoundException)
    - Dělení nulou (DivideByZeroException)
* Obecně chyba vzniká při porušení sémantických omezeni jazyka

**Postup chyby**

* Vznik chyby v bloku TRY
  + Hledání odpovídající řešení CATCH
  + Další příkazy v TRY po chybě se neprovedou



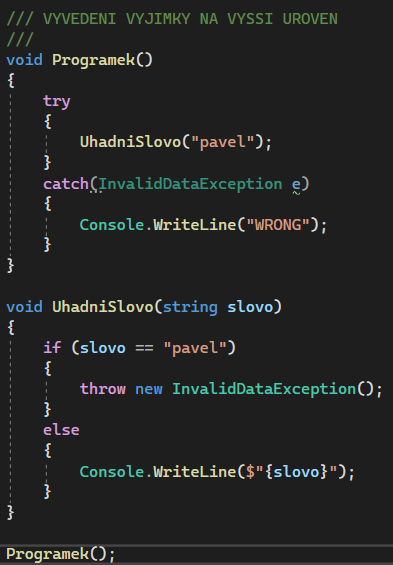
* Vznik chyby mimo blok TRY ale bylo zavoláno z TRY
  + Předá se do místa volání (do bloku try v tomto případě)
  + Pokračuje se jako při vzniku v bloku



* Vznik chyby bez bloku TRY
  + Výjimka se šíří na nadřazenou úroveň
  + Není-li ošetřena ve funkci MAIN
    - Program skončí a vypíše chybu

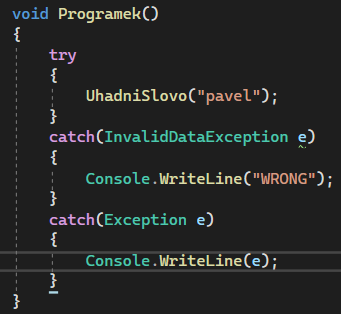
**Předání výjimky výše (vygenerování vlastní)**

* **Odchycení**
  + Odchycení výjimky a pomocí THROW předání výše
  + Výše se předpokládá také ošetření
* **Neodchycení**
  + Vyhození „vlastní“ výjimky pomocí THROW



**Obecné zachycení výjimky**

* I pokud máme dojem, že všechny možné chyby jsme ošetřili, není to nikdy jisté
* Je dobré mít vždy zachyceno globální chybu pomocí CATCH(EXCEPTION E)
  + Na tento catch se chyba zachytí, pokud na žádný jiný není určena
  + Zajistí to běh programu i po nastání nečekané chyby
  + V tento CATCH je dobré vypisovat chybové hlášení pro budoucí určení a popřípadě lepší ošetření



**Finally**

* Funkce, která umožňuje vykonání kódu za obou podmínek
  + Vykoná se při úspěšném průběhu bloku (TRY)
  + Vykoná se při chybě v bloku (po průběhu CATCH)
  + Vykoná se i při **RETURN** v bloku

**Aserce (Předpoklady)**

**TZV. Predikáty**

* O nich se domnívá, že budou na daném místě vždy pravdivé
* Pomocná funkce pro DEBUGGING a TESTING
* Jedná se o detekci stavů, který by neměli **NIKDY** nastat
  + Pokud se nesplní podmínka ASERCE – nastal nežádaný (nemožný v plánovaném průběhu) stav
* Často může pomáhat pochopit/číst kód
  + Ukazuje, jak správně vypadá průchod programu

**Debuggování**

Proces hledání a identifikace chyb

Za použití breakpointů si stopneme kód, kde chceme a dále můžeme pokračovat po řádku

* Je nám zároveň dostupná alokovaná paměť s objektama

Součástí procesu vývoje software

Umožňuje programátorům najít a opravit chyby dříve, než se software uvede do provozu