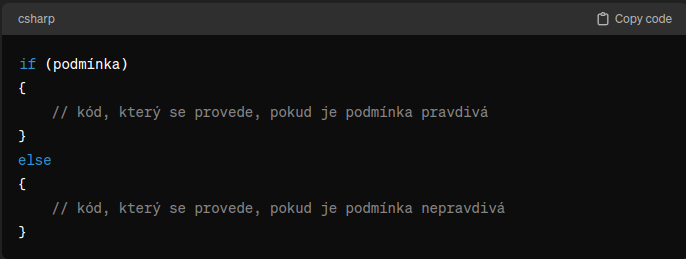
**Řízení toku programu (podmíněné výrazy, cykly, výjimky, řízení událostmi, funkcionální programování) (Kl)**

Dělané v jazyku C#

# Podmíněné výrazy

**IF-ELSE**

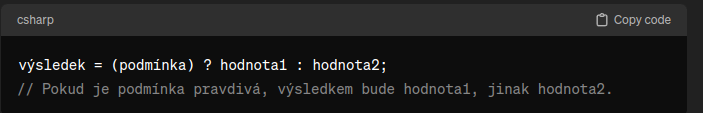


**IF-ELSE IF - ELSE**

# 

Lze zkratit do jednoho radku pomoci ternarnich operatoru:

Lze mit vice ternarnich operatoru v jednom radku

  
  
  
  
  
**SWITCH** **SWITCH - PROPADAVANI**

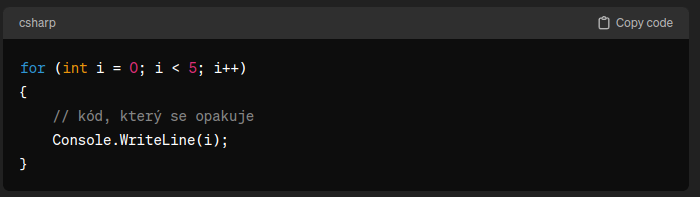
Pokud potřebujeme ve více blocích case provádět stejný kód, stačí tyto bloky vložit pod sebe a neukončovat každý blok pomocí break, ale celou skupinu bloku jedním break.

Neukončené bloky tak propadnou a vykoná se kód společný pro více bloků:

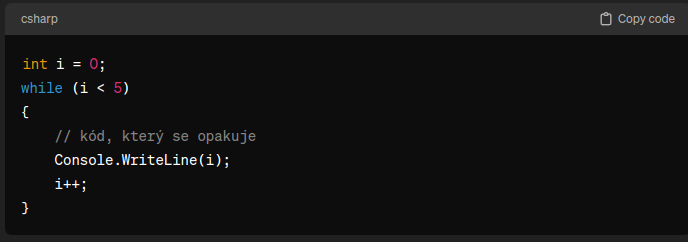


# Cykly (loops)

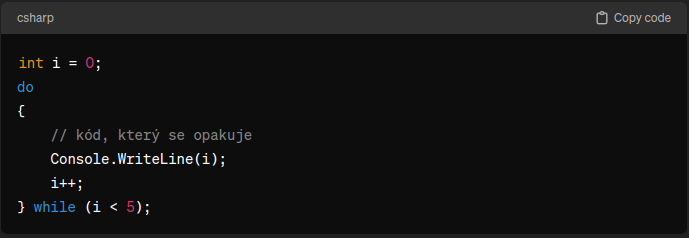
**FOR**

****

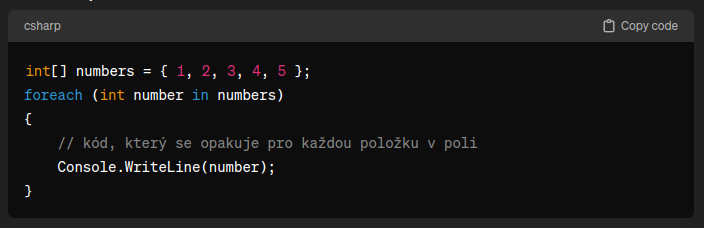
**WHILE**

****

**DO-WHILE**

****

**FOREACH**

****

**Do-while vs while**

While loop:

* Ve while loopu je podmínka zkontrolována před provedením bloku kódu.
* To znamená, že pokud je podmínka na začátku nepravdivá, blok kódu vůbec nevykoná a cyklus se vůbec neprovede.

Do-while loop:

* V do-while loopu je podmínka zkontrolována po provedení bloku kódu.
* To znamená, že blok kódu se vždy provede a poté se kontroluje podmínka. Pokud je podmínka pravdivá, cyklus se opakuje; pokud ne, cyklus končí.

**Kdy jaký loop použít?**

for loop:

* Přesně víme, kolik opakování chceme

while loop:

* Neznáme přesný počet opakování předem, tudíž checeme opakovat dokud je/není splněná určitá podmínka

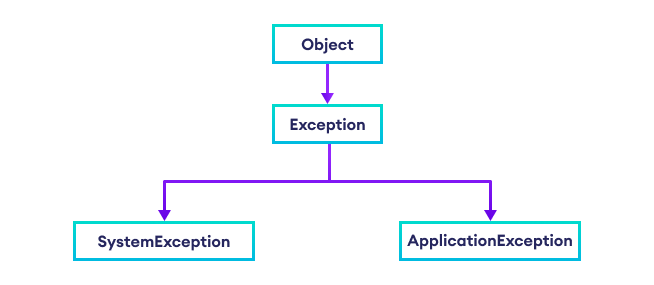
do-while loop:

* Smyčku chceme provést alespoň jednou a poté dokud je/není splněná určitá podmínka

foreach:

* Vhodný pro situace, kdy potřebujeme projít všechny prvky kolekce (například pole, seznam nebo jiný typ kolekce) a provést s nimi nějakou operaci.

# Výjimky (exceptions)



SystemException – jsou built-in vyjimky

ApplicationException – jsou user-defined vyjimky

## System Exceiption

Priklady vyjimek:

* StackOverflow (stack presahne svoji velikost, nekde bezi neco na loopu a nemelo by)
* Arithmetic
  + DivideByZero (delime 0)
  + OverFlow (vysledek je out of range)
* Argument Exception (neplatny argument passnuty do metody)

## User-defined Exception

A screen shot of a computer program

Description automatically generated

## Chyba (error) vs výjimka (exception)

* Chyby představují stavy, jako je chyba kompilace, syntaktická chyba, chyba v logické části kódu, nekompatibilita knihovny, nekonečná rekurze atd.
* Chyby jsou obvykle mimo nasi kontrolu a neměli bychom se je pokoušet řešit
* Výjimky můžeme zachytit a osetrit

## Exception Handling

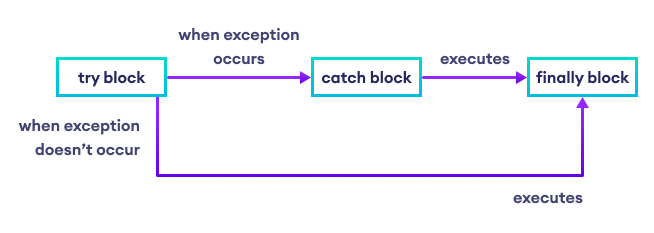
A black rectangular object with white text

Description automatically generated

A screenshot of a computer

Description automatically generated

Finally se spusti **pokazde**



# Řízení událostmi (events and delegates)

## [Co je událost (event)?](https://www.tutorialsteacher.com/csharp/csharp-event)

* Událost (event) je oznámení odeslané objektem, které signalizuje výskyt nějaké akce. Události v prostředí .NET se řídí observer design patternem.
* Třída, která události vyvolává, se nazývá Publisher (vydavatel) a třída, která oznámení přijímá, se nazývá Subscriber (účastník). Odběratelů jedné události může být více. Typicky vydavatel vyvolává událost, když došlo k nějaké akci. Odběratelé, kteří mají zájem dostávat oznámení, když k nějaké akci došlo, by se měli zaregistrovat k události a zpracovat ji.
* V jazyce C# je událost (event) zapouzdřený delegát => je závislá na delegátovi. Delegát definuje signaturu pro metodu obsluhy události třídy subscribe

A diagram of events

Description automatically generated

## [Co je delegate?](https://www.tutorialsteacher.com/csharp/csharp-delegates)

* Pouzijeme pokdu chceme predat funkci jako parametr
* Delegát je datový typ referenčního typu, který definuje signaturu metody. Stejně jako u jiných datových typů můžete definovat proměnné delegáta, které mohou odkazovat na libovolnou metodu se stejnou signaturou jako delegát
* Při práci s delegáty se postupuje ve třech krocích:
  + Deklarovat delegáta
  + Vytvoření instance a odkaz na metodu
  + Vyvolání delegáta
* Delegáta lze deklarovat pomocí klíčového slova delegate, za kterým následuje signatura funkce

A screenshot of a computer program

Description automatically generated

A diagram of a program

Description automatically generated

## [Func](https://www.tutorialsteacher.com/csharp/csharp-func-delegate)

* Built-in delegat, ktery vraci hodnotu/hodnoty

A diagram of a type of return

Description automatically generated

A triangle with text and a line

Description automatically generated with medium confidence

## [Action](https://www.tutorialsteacher.com/csharp/csharp-action-delegate)

* Built-in delegat, ktery ale nic nevraci
* Vyuziva se tedy s void metodami

A computer code with black and white text

Description automatically generated

## [Observer design pattern](https://refactoring.guru/design-patterns/observer) (navic, neni potreba se ucit, ale souvisi s eventy)

Observer je návrhový vzor chování, který umožňuje definovat mechanismus odběru pro upozornění více objektů na všechny události, které se stanou s objektem, který pozorují.

## Problem

Předstame si, že máme dva typy objektů: Zákazník a Obchod. Zákazník má velký zájem o výrobek určité znacky, který by měl být brzy dostupný v obchodě.

Zákazník by mohl obchod navštěvovat každý den a kontrolovat dostupnost produktu. Dokud je však výrobek stále na cestě, většina těchto cest by byla zbytečná.

A black and white image of a store

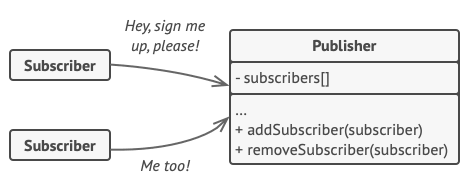
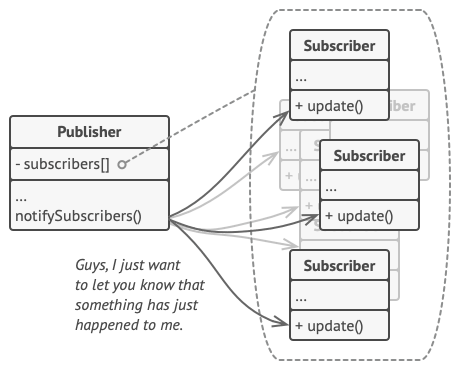
Description automatically generated

Na druhou stranu by obchod mohl posílat všem zákazníkům spoustu e-mailů (což by mohlo být považováno za spam) pokaždé, když je k dispozici nový produkt. To by některým zákazníkům ušetřilo nekonečné cesty do obchodu. Zároveň by to ale naštvalo jiné zákazníky, kteří o nové produkty nemají zájem.

A mame tu problem, panove. Buď zákazník ztrácí čas ověřováním dostupnosti výrobku, nebo obchod plýtvá zdroji na upozorňování nesprávných zákazníků.

## Reseni

Vzor Observer navrhuje přidat do třídy publisher mechanismus odběru, aby se jednotlivé objekty mohly přihlásit k odběru nebo odhlásit z proudu událostí přicházejících od tohoto publisheru. Tento mechanismus se skládá z 1) pole pro uložení seznamu odkazů na objekty odběratelů a 2) několika veřejných metod, které umožňují přidávat odběratele do tohoto seznamu a odebírat je z něj.



# [Funkcionální programování](https://hamidmosalla.com/2019/04/25/functional-programming-in-c-sharp-a-brief-guide/)

## Co je to funkcionální programování?

Existují tři "typy" programování:

* procedurální programování
* objektově orientované programování
* funkcionální programování.

V objektově orientovaném programování (OOP) vytváříme "objekty" (odtud název), což jsou struktury, které mají data a metody. Ve funkcionálním programování je vše funkcí. Funkcionální programování se snaží oddělit data a chování a OOP tyto pojmy spojuje.

## Jaká jsou pravidla funkcionálního programování?

Zasadne neprovadet

Pro pochopení tohoto konceptu je třeba znát dvě hlavní věci:

* **Data jsou neměnná:** Pokud chceme změnit data, například pole, vrátíme nové pole se změnami, nikoli původní.
* **Funkce jsou bezstavové:** Funkce se chovají, jako by to bylo poprvé, pokaždé! Jinými slovy, funkce dává vždy stejnou návratovou hodnotu pro stejné argumenty.
* Funkce by měly přijímat alespoň jeden argument
* Funkce by měly vracet data nebo jinou funkci
* Nepouzivaji se loopy

Vic tady: <https://github.com/readme/guides/functional-programming-basics>

A screen shot of a computer program

Description automatically generatedOOP Funkcionalni

A computer screen with text on it

Description automatically generated

Proc?

* Lehci debugovani
* Lepsi/ rychlejsi pochopeni kodu

## Ukazky

### Neměnitelné typy {Immutable Types}

* Upřednostňujeme vracení nových instancí, pokud změníme stávajicí typ

A screenshot of a computer program

Description automatically generated

### Vyjádření místo výroků (Expression Instead of Statements)

A screenshot of a computer program

Description automatically generated

### Method Chaining/Pipe-lining and Extension Methods

A white background with text

Description automatically generated with medium confidence

### LINQ

A screenshot of a computer program

Description automatically generated

Muzeme videt, ze pouziti LINQ (WHERE klauzule) vypada lepe a splnuje podminky funkcionalniho programovani

### Higher-Order Functions (HOF)

* Funkce vyššího řádu jsou funkce, které přijímají funkci jako argument nebo vracejí funkci (nebo obojí).
* Jejich opakem jsou funkce prvního řádu - to znamená, že nepřijímají funkci jako argument ani nevracejí funkci.
* HOF se vyskytuji v lambdach

Creation of higher-order function (MyWhere)

A screenshot of a computer code

Description automatically generated