## Datové typy

### Celočíselné

* Integer (int)
  + celočíselný datový typ
  + Bez znaménka
  + 32 bit (31 bit číslo, 1 bit znaménko)
* Short
  + 16 bit
  + celočíselné
  + -32 768 až 32 767
* Long
  + 64 bit
  + celočíselné
  + Hodnoty od hodně málo do hodně moc
* Byte
  + 8 bit
  + Od 0 do 255
  + Bez znamének (se znaménkami sbyte)
  + Celočíselné

### Desetinná čísla

* Float
  + S plovoucí čárkou
  + Nejčastěji používaný datový typ
  + 4 byt – 32 bitů
  + Přesnost 6-9 desetinných míst
  + Netradiční zápis se sufixem: **float number = 10.0f**
* Double
  + Přesnější float
  + 64 bit
  + Přesnost 15-17 desetinných míst
  + Zápis klasický nebo se sufixem **d**

### Text/charakter

* Char
  + 16 bit
  + Datový typ pro reprezentaci Unicode UTF-16
  + Znak
  + Char letter = ´a´;
* String
  + Prezentace 0 a více Unicode znaků
  + Nemá hodnotu, pouze reprezentuje umístění charů v paměti
  + String text = „Hello World“;
* Boolean
  + 1 bit
  + hodnotový
  + Return: True/False
  + bool decision = true;

## Cykly

* Instance třídy Random
  + Random rnd = new Random ();
  + Int a = rnd.Next(10);
  + Next je funkce ve třídě Random

float number = 0;

bool n = false; //n jako náhodná proměnná

while (n == false)

{

try

{

number = float.Parse(Console.ReadLine());

n = true;

}

catch (FormatException)

{

Write("Hele to asi nebude float zkus to znova");

}

}

* return number; // zkoušení datového typu

### If

* + Jednoduchá podmínka
  + Argument bez „;“

### While

* + Musí mít podmínku jinak se bude neustále opakovat
  + Podmínka true/false

### For

* + Pro každý člen udělej toto
  + Začínám od 0, potom nemusím řešit počet cyklů

## Funkce

* Rveme to nad Main
  + Funkce void – pracuje s danými hodnotami, nic nevrací
  + Static funkce – funkce pracující pouze pro daný soubor
    - to samé platí i pro proměnné
  + Funkce nevoidovské: static Integer(int a, int b)

{

Int result = a+b;

Return result;

}

* + Nevoidovské funkce mají vstupní argumenty a vrací právě daný integer, float atd.
  + Kontrola vstupu

int number;

While(!try.Parse(Convert.ToInt32(Console.ReadLine()),out number)

{

Console.WriteLine(„Ha lol špatný v stup, zkus to znova “)

}

### Rekurze

* + Volám funkci ze sebe samé
  + Do argumentu dám argument té samé funkce celé funkce
  + Musí mít podmínku jinak se zacyklí

## Pole/seznamy

* Uložení více hodnot (spíše adresy v paměti) v jedné proměnné
* Hodnoty se nazývají „prvky“
* Definice (integrované prvky) pole:
  + Int [] myArray = {1,9,5,7,6}
    - Hranaté závorky za datovým typem
    - Název daného pole
* Další varianty definování:
  + String [] terxtArray; využijeme, když budeme definovat později
  + Flaot [] numberArrray = new float [5] – vytvoří „prazdné“ pole velikosti 5
  + Bool [] arrayOfThruth = new bool[] {true,false,true,false} ekvivalentní s úplně první definicí
* K n-tému prvku pole se dstanu takto:
  + myArray[1]
  + pozor index začíná od nuly
* vícerozměrné pole
  + int[,] my2DArray = {{9,8,7}, {654}, {321}}
  + 2D pole představuje matici
  + 3D pole představuje pole matic

## Listy (seznamy)

* Nevýhody klasického pole (array):
  + Pevná délka
  + Složité vkládání nového prvku
  + Co když potřebujeme prvky označit jinak než jenom [0] nebo [1]
  + Obecně málo flexibilní
* Pole akorát chytřejší
* List <datový typ> myList = new List <datový typ>;
* Lze ho libovolně natahovat a zkracovat
* Vkládání prvků:
  + Na konce listu: myList.Add(variable);
  + Na konkrétní místo myList.insert(index, variable);
  + Více proměnných na konec listu mylist.AddRange(varibale);
* Odebírání prvků:
  + myList.Remove(variable);
  + myList.removeAt(index)
* iterujeme stejně jako array (for, foreach)
* čistě jednodimenzionální
* paměťově a výpočetně náročnější

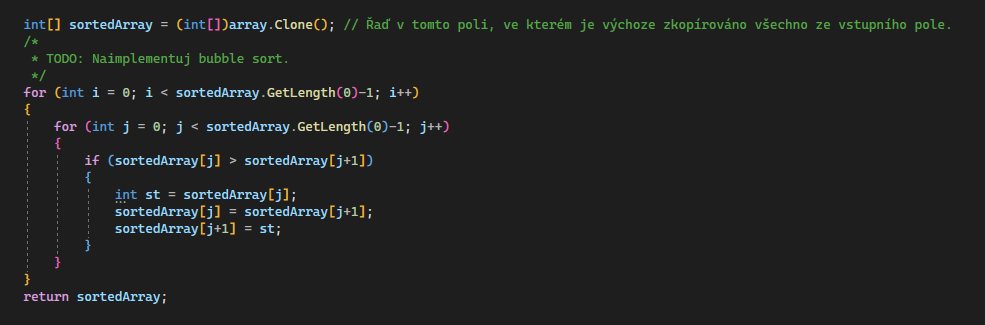
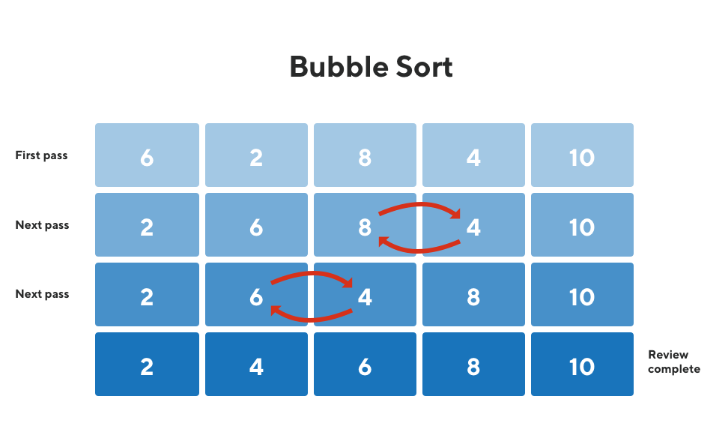
## Slovníky

* zápis Dictionary <datovýtypKlíče, datový typ hodnoty> myDictionary = new Dictionary <datový typ klíče, datový typ hodnot>
* zápis ve slovníku se skládá z klíče a hodnoty
  + klíč je „index“
  + hodnota je hodnota wau revoluční
  + obojí libovolnej datovej typ
* Př: známky pro žáka:
  + Dictionary <string, int> students = new Dictionary <string, int>();
  + Students [„novák“] = new Int [] {1,1,3,2};
* Iterace
  + Přes klíče (vlastnost dict.Keys nám vrátí pole klíčů)
    - Klasický for nebo foreach
  + Přes hodnoty (dict.Values nám vrátí pole hodnot)
    - Klasický for nebo foreach
  + Přes dvojice klíč - hodnota - foreach KeyValuePair nazevKVP in myDict) { }
  + Klasický foreach cyklus nebude fungovat
* Nepotřebujeme definovat žádnou velikost¨
* Zajímá nás jestli:
  + Je ve slovníku právě daná hodnota – myDict.ContainsValue(value);
  + Je ve slovníku daný klíč – myDict.ContainsKey(key);
* Pouze pokud nám list nebo pole nestáčí
* Výrazně náročnější
* Když potřebujeme spojit dvě „hodnoty“ k sobě

## Řadící algoritmy

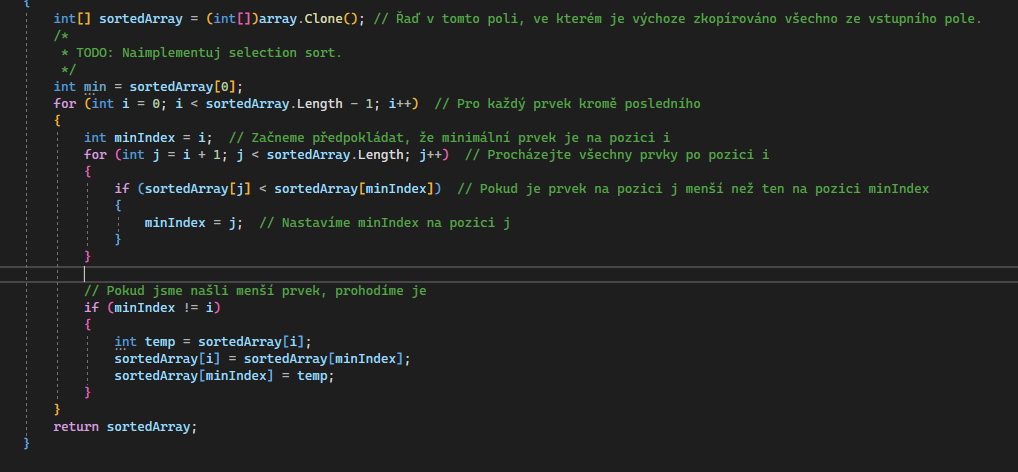
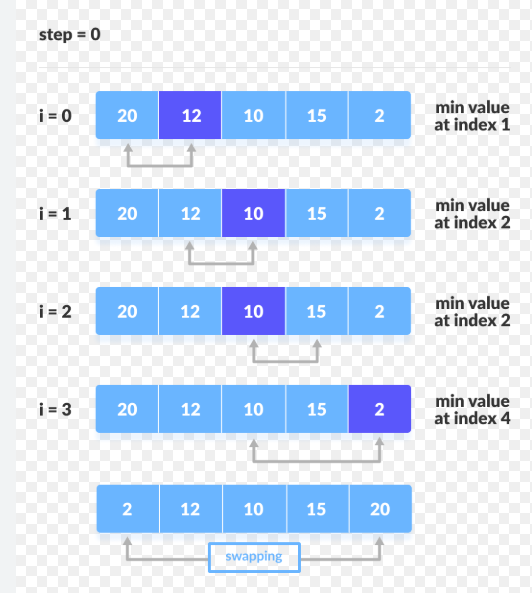
### Bubble sort

* Procházíme zleva doprava, tolikrát, kolik v něm máme prvků
* Prvky porovnáme a kdyžtak prohodíme, složití na n nadruhou



### Selection sort

* Vybírá nejmenší prvky a dá ho na začátek
* Najdu nejmenší dám ho na začátek, potom vždy hledám napravo od už hotového čísla
* Složité n na druhou



### Insertion sort

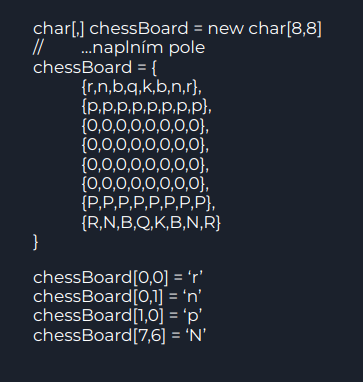
* Procházíme polem zleva doprava
* Aktuální prvek porovnáme s předešlým a případně to prohodíme

### Merger sort

* Rozdělíme na dvě části a ty potom řadíme samostatně a nakonec dohromady
* Obtížnost n.logn

### Quick sort

# 2 D pole

* čím víc čárek tím víc dimenzí
* Get lenght(0) – velikost 1. dimenze
* Úkol: dělat si pole Miss/Hit
* Soupis souřadnice – vystřelené

# Stringbuilder

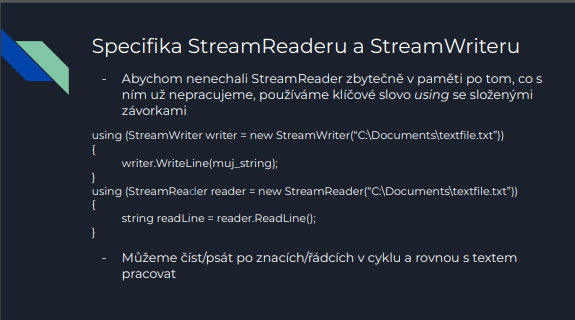
* String
  + Reprezentuje 0 a více unicode znaků
  + Pole
  + Referenční datový typ
  + Neuchovává hodnotu, ale jen odkaz na adresu v paměti
  + Do dvojitých uvozovek
  + Konverze do stringu var.ToString();
  + Konverze ze stringu: int.Parse()
    - * + Int.TryParse();
        + Convert.ToInt32()
* Stringbuilder
  + String nelze upravovat – vždycky se vytvoří nový string
  + StringBuilder nezabírá tolik paměti, netvoří se pokaždé nový
  + Pro používání potřebujeme přidat using.System.Text
  + StringBuilder myStringbuilder = new Stringbuilder(„Hello world“);
  + Funguje jako list
  + Pomocí funkce append můžeme do stringbuilderu přidat další string
  + Funkce insert – vloží string (index, „tvoje máma“)
  + Remove – dáme interval (5,7)
  + Replace – vymění specifický char s jiným)
  + String ze stringbuilderu dostaneme pomocí: .To(String)

# Kódování znaků

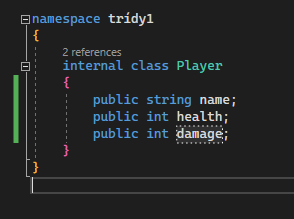
* Unicode – mezinárodní kódovací standart pro znaky
* Code point – písmeno/emoji
* Unicoe Transformation unit
* Reprezentuje codepointy z unicode 16 – bitovými code unity (UTF – 16: potřebujeme to celý narvat do 16 bytů)

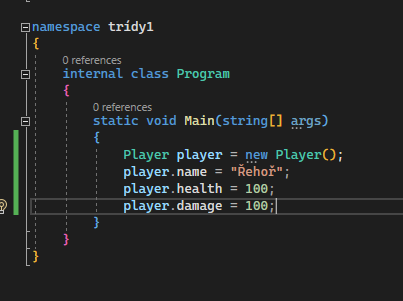
# Textové soubory

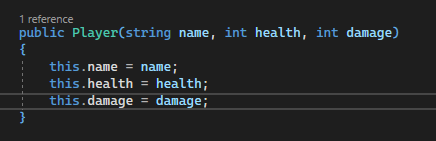
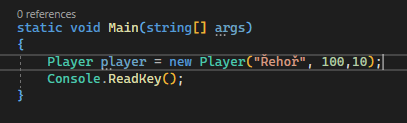
* Použijeme třídu File
* Potřebujeme přesnou adresu souboru
* Prvně si zkontroluji, že soubor existuje
* File.Existst(adresa) – true/false
* File.ReadAllText(adresa)
* FileReadAllLines(adresa)
* StreamReader
  + Objekt slouýící ke čtení textu z bytového streamu
  + Musíme ho instancovat
    - StreamBuilder reader = new StreamReader(adresa\_souboru);
    - Reader.Read() – přečte jeden znak, případně tolik, kolik mu řeknu v aprametru
    - Reader.ReadLine() – přečte jeden řádek
* StreamWriter
  + Musíme instancovat stejně jako streamreader
  + Write.Write() – zapíše textovou reprezentaci objektu, kterou mu dáme
  + Writer.WriteLine() – zapíše to co mu dáme a ukončí řádek
  + Používáme pomocí using, aby nám neběžel pořád



# Třídy

* Objektově orientované programování
* Uložení vlastností a činností pohromadě
* Třídy jeeeej skvělé úžasné nejlepší
* Funkční celek, předloha, šablona
* Je potřeba, aby mohly ty třídy nějaké vlastnosti sdílet
  + Dědičnost
  + Podtřídy
* Je potřeba inicializovat (vytvořit instanci) např. takto
* takto



* Konstruktor – funkce, která inicializuje celou funkci najednou
  + Prvně v samotné třídě:
  + Potom do toho musím ješte ty data narvat😊:

# Statická funkce

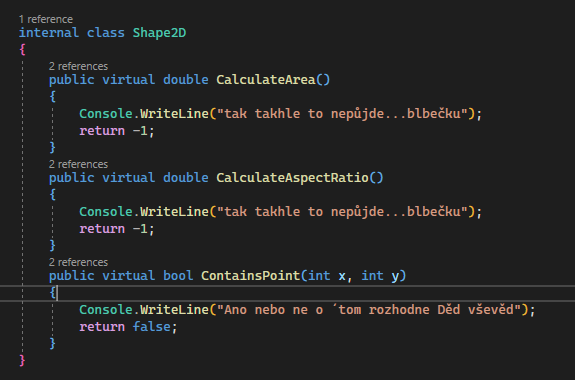
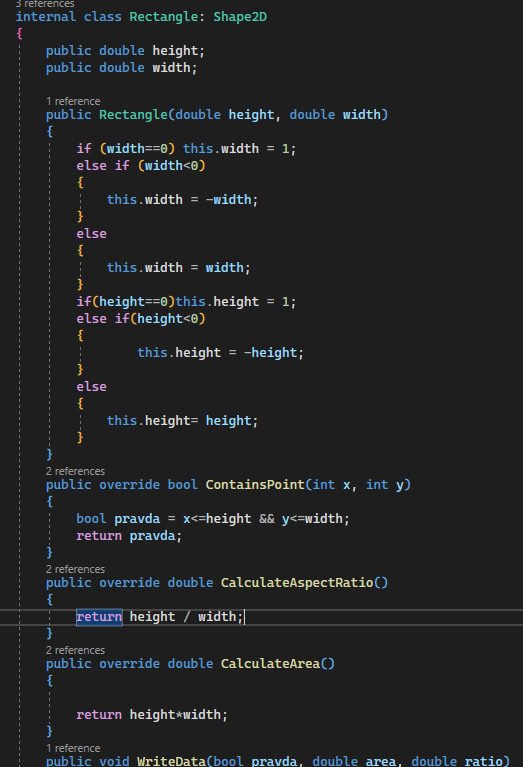
* Funkce, která se neváže na konkrétní instanci
* Mohu používat rovnou bez instancování
* Např. Console.Writeline(„“);

# Dědičnost

* Třídy od sebe mohou dědit proměnné a funkce
* Můžeme vytvářet hierarchii, jak od sebe ty třídy dědí
* Rodičovská třída – animal
* Potomek např cat
* Dědění značíme dvojtečkou za názvem

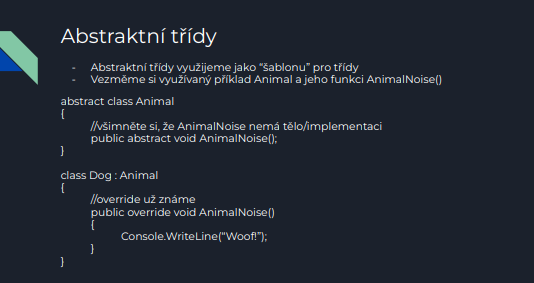
# Polymorfismus

* Polymorfismus umožňuje jedné funkci pracovat různě podle toho, v jaké třídě je
* V mateřské třídě, před funkci napíšeme virtual
* V dceřinné funkci napíšeme před funkci override a doděláme



# Abstraktní třída

* Šablona pro třídy
* Nelze instancovat
* Virtual vs abstract – ve virtual jsme mohli zapsat tu základní hodnotu
* Využití:
  + Neexistuje smysluplná výchozí implementace
  + Každá třída, která dědí, musí metodu implementovat jinak
  + Chceme zamezit možnosti implementovat instancování oné třídy



# Interface

* Co když potřebujeme sdílet určité chování mezi třídami, které si nejsou blízko v hierarchii (pták, letadlo?)
* Co když budeme dědit od více tříd
* V podstatě public abstract classa
* Je zvykem začínat velkým měkkým I
* Můžu si do toho zabalit nějaké chování

