PV178 – Lab 02

Dominik Lašo


Kahoot!

Agenda

- HW01
- Quick Sneak Peak to naming convensic
- Class + Interface
- Debugging a Unit Testing
- Hodnotové a referenční typy
- ref vs out

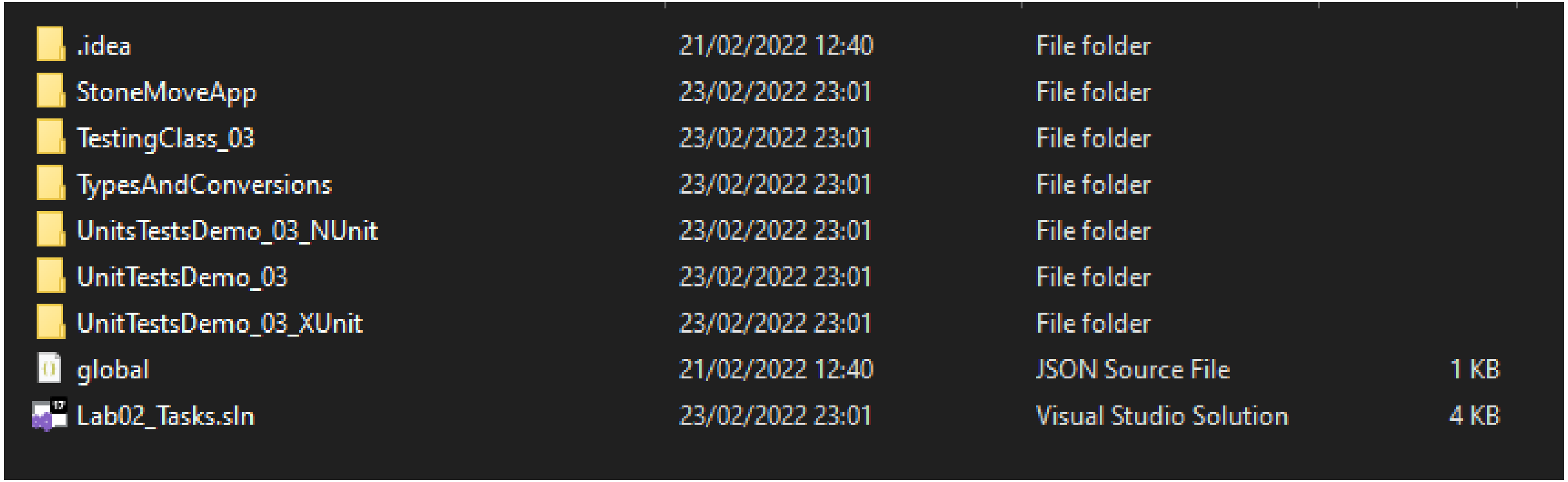
HW01

- 2 tyzdne (do 6.3.2022)
- Cielom ulohy: precvicit / zopakovat OOP
- Najdolezitejsie casti ulohy:
- 1. Cistota kodu
- 2. Funkcnost

- Odovzdavajte “.zip” file, kde bude:
- - “.sln”
- - cely subor s hw01

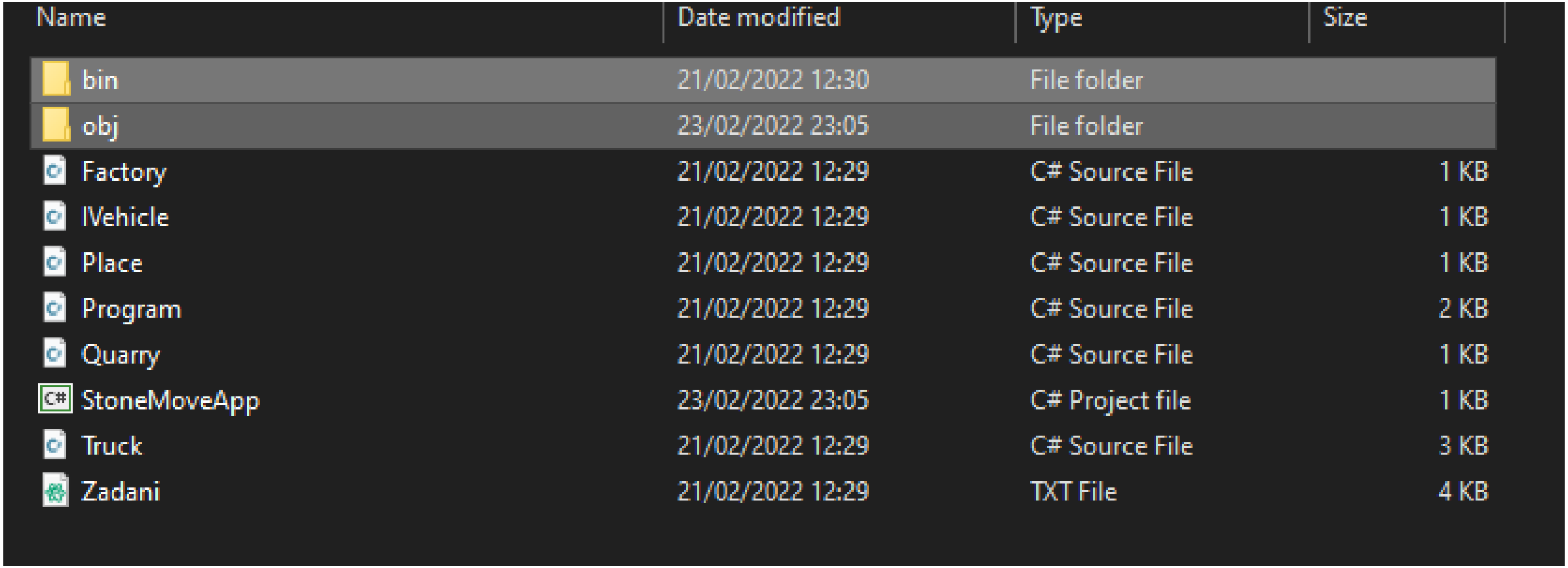
HW01

- Trosku inak …
- Urcite nemazte “.sln” subor !!!!



HW01

- Trosku inak …
- Mozte zmazat “bin” a “obj” … toto su vase subory a celkovo su nepotrebne pre nas ako opravujucich (generuju sa pri spusteni debug / program)



Name	Date modified	Type	Size
bin	21/02/2022 12:30	File folder	
obj	23/02/2022 23:05	File folder	
Factory	21/02/2022 12:29	C# Source File	1 KB
IVehicle	21/02/2022 12:29	C# Source File	1 KB
Place	21/02/2022 12:29	C# Source File	1 KB
Program	21/02/2022 12:29	C# Source File	2 KB
Quarry	21/02/2022 12:29	C# Source File	1 KB
StoneMoveApp	23/02/2022 23:05	C# Project file	1 KB
Truck	21/02/2022 12:29	C# Source File	3 KB
Zadani	21/02/2022 12:29	TXT File	4 KB

A quick peak to code-style

C# Coding Standards and Naming Conventions

Object Name	Notation	Length	Plural	Prefix	Suffix	Abbreviation	Char Mask	Underscores
Namespace name	PascalCase	128	Yes	Yes	No	No	[A-z][0-9]	No
Class name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Constructor name	PascalCase	128	No	No	Yes	No	[A-z][0-9]	No
Method name	PascalCase	128	Yes	No	No	No	[A-z][0-9]	No
Method arguments	camelCase	128	Yes	No	No	Yes	[A-z][0-9]	No
Local variables	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Constants name	PascalCase	50	No	No	No	No	[A-z][0-9]	No
Field name	camelCase	50	Yes	No	No	Yes	[A-z][0-9]	Yes
Properties name	PascalCase	50	Yes	No	No	Yes	[A-z][0-9]	No
Delegate name	PascalCase	128	No	No	Yes	Yes	[A-z]	No
Enum type name	PascalCase	128	Yes	No	No	No	[A-z]	No

https://github.com/ktaranov/naming-convention/blob/master/C%23%20Coding%20Standards%20and%20Naming%20Conventions.md
https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions

Class, Interface a pod.

- Lets check out some code
- Class example
- Interface example
- Field example
- Property example
- Constructor example

```
//  class           // interface
9 references
class Motorbike : IDriveable, IMarketable
{
    // field
    private readonly int _wheelCount = 1;

    // Property
    10 references
    public string Name { get; set; }

    // Constructor
    4 references
    public Motorbike() { }

    // Overloaded constructor
    0 references
    public Motorbike(int wheelsCount)
    {
        _wheelCount = wheelsCount;
    }
```

Class, Interface a pod.

```
Motorbike bike = new Motorbike();
bike.Name = "bike";

var bike2 = new Motorbike();
bike2.Name = "bike2";

var bike3 = new Motorbike { Name = "bike3" };

var bike4 = new Motorbike(10);

var bike5 = new Motorbike(wheelsCount: 10);

var bike6 = new Motorbike(wheelsCount: 10) { Name = "Bike6" };

// A little bit of Magic ;)
IDriveable bike7 = new Motorbike(wheelsCount: 10) { Name = "Bike7" };
// results in error -> bike7.Name; ... but why ? ved sme predsa Name setli :Hmmm:
```

Class, Interface a pod.

- ":" notacia
- Mozem dedit maximalne z ***1 triedy***, avsak mozem implementovat ***n interfacov***

```
2 references
interface IDriveable
{
    2 references
    void Drive();
    2 references
    void Run();
    3 references
    int NumberOfSeats();
}
```

```
2 references
internal class Car : IDriveable
{
    1 reference
    public void Drive()...

    2 references
    public int NumberOfSeats()
    {
        Console.WriteLine("Hello from car");
        return 4;
    }

    1 reference
    public void Run()...
}
```

Class, Interface a pod.

```
2 references
internal class Car : IDriveable
{
```

```
7 references
class Motorbike : IDriveable, IMarketable
{
```

```
1 reference
class Bus : Car, IMarketable
{
```

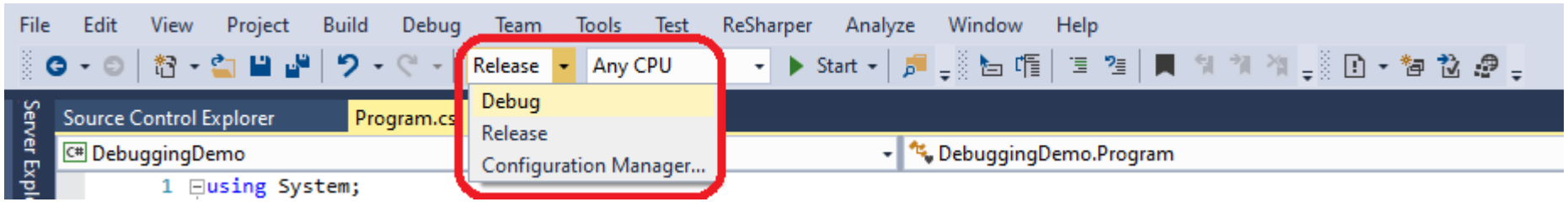
```
1 reference
class Bus : Car, IMarketable, Motorbike
{
```

Debugging - obecně

- „Proces hledání a odstraňování chyb“
- Obtížný a časově náročný proces
- **Anti-debugging** techniky („bránění se pokusům o ladění finálního programu“)
- https://github.com/bezzad/AntiDebugging
- Program sloužící k debugování = Debbuger

Debug vs Relase mód

- Typ sbuildování aplikace
- DEBUG - Pro vývoj, není optimalizována
- RELEASE - Jde k zákazníkovi, optimalizována (měla by běžet rychleji)



PDB soubor (.pdb)

- pdb = Program database
- Vytváří se při buildu (standardně)
- Obsahuje informace nepostradatelné pro debugging
- Slouží k obousměrnému mapování kódu (.cs) na instrukce výstupního programu (.bin)
- Většinou není dodáván zákazníkovi

Kód (.cs)

Program (.bin/.dll)

Debugging + Unit Testy

TDD + Pair Programming / samostatná práce ?

1. Rozdělte se do dvojic
2. Naprogramujte podle zadání https://github.com/xurxodev/FizzBuzz-Kata
3. Jeden píše testy, druhý kód – pracujte na 1 PC („pair programming“)

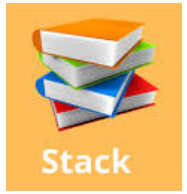
- Napište alespoň 3 testy

Honotové typy

- Proměnné přímo nesou svá data
- Každá proměnná má svoji oddělenou kopii dat
- Jsou ukládány na zásobníku (=stack)
- Operátor **==** porovnává obsah proměnné
- Příklady: int, double, struct, enum, tuple …

```
int intVar = 100;
float floatVar = 10.2f;
char charVar = 'A’;
bool boolVar = true
```

Hodnotové typy – příklad




```
int a = 100;
int b;
```

a: 100
b: 0

```
int b = a;
```

a: 100
b: 100

```
a = 0;
```

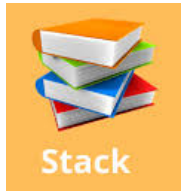
a: 0
b: 100

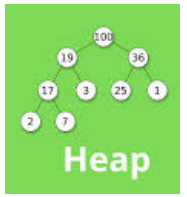
Referenční typy

- Proměnné nesou pouze odkaz na data (referenci)
- Všechny typy, které dědí od třídy *object*
- Vytvářejí se přes klíčové slovo *new*
- Operátor *==* porovnává jestli je odkazována stejná proměnná
- Proměnná se tvoří na haldě (heap)
- Příklady: List<T>, int[], object, string…

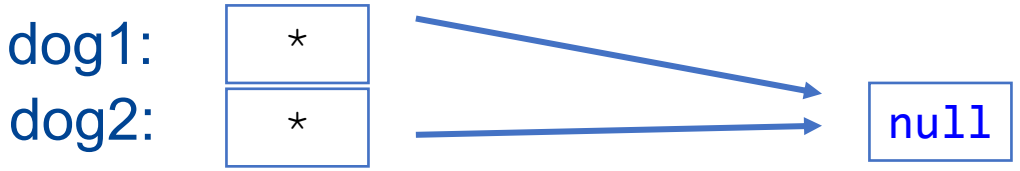
```
List<int> MyInts;
object MyObject;
string HelloString;
int[] ArrayOfInts;
```

Referenční typy – příklad

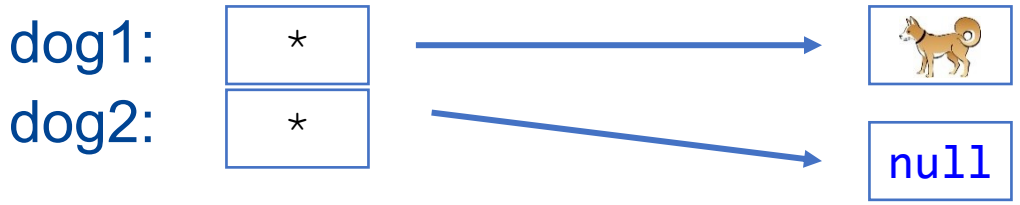




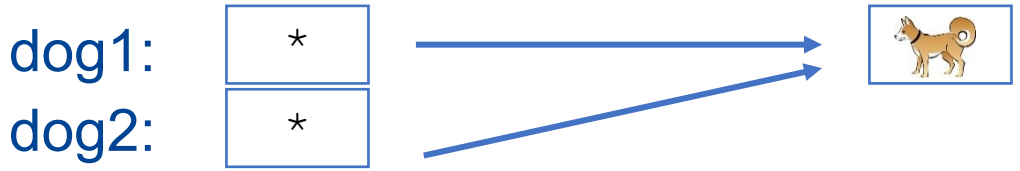

```
object dog1;
object dog2;
```



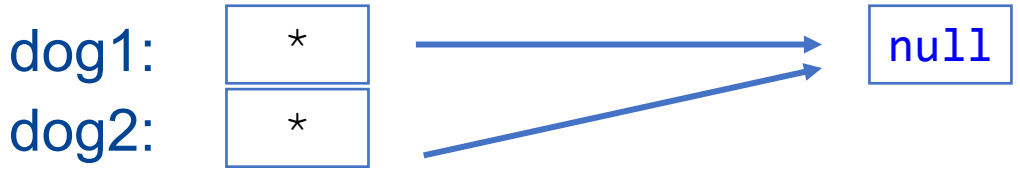

```
dog1 = new object();
```




```
dog2 = dog1
```

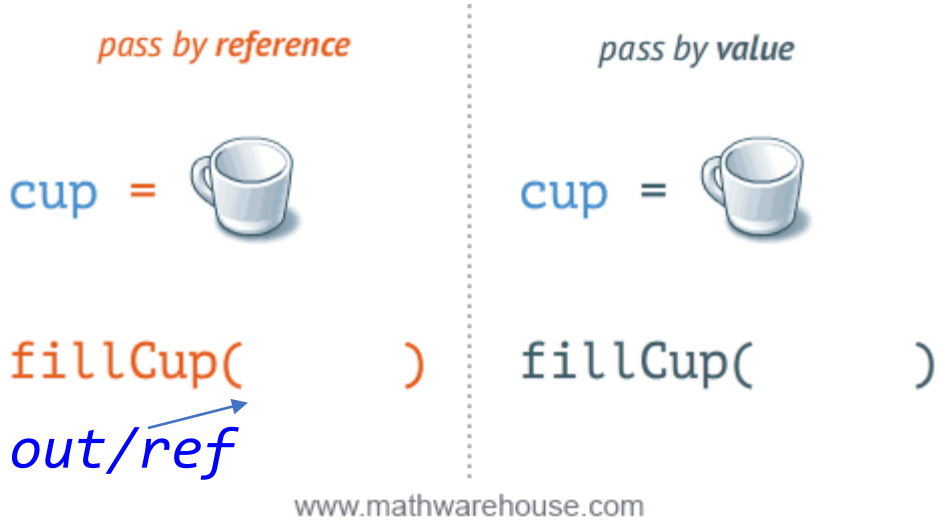



```
dog2 = null
```



ref vs *out*

- Klíčová slova jazyka C#
- Píší se před parametry funkce
- Řeší problém jak vracet více hodnot z metod
- Hodí se pro inicializaci proměnných
- Nejčastěji se používají ve spojení s hodnotovými typy



ref

- Proměnná **MUSÍ** být inicializována!

out

- Proměnná **NEMUSÍ** být inicializována!

ref a *out* – příklad

```
public static void SwapVariables(ref int a, ref int b)
{
        var temp = b;
        b = a;
        a = temp;
}

public static void InitVariables(out int a, out int b)
{
        a = 5;
        b = 3;
}

static void Main(string[] args)
{
        int a;
        int b;

        InitVariables(out a, out b);
        SwapVariables(ref a, ref b);
}
```

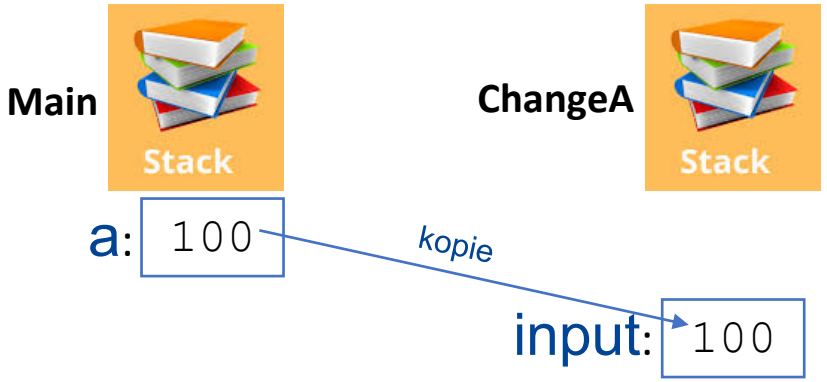
ref a *out* – příklad

Ref	Out
The parameter or argument must be initialized first before it is passed to ref.	It is not compulsory to initialize a parameter or argument before it is passed to an out.
It is not required to assign or initialize the value of a parameter (which is passed by ref) before returning to the calling method.	A called method is required to assign or initialize a value of a parameter (which is passed to an out) before returning to the calling method.
Passing a parameter value by Ref is useful when the called method is also needed to modify the pass parameter.	Declaring a parameter to an out method is useful when multiple values need to be returned from a function or method.
It is not compulsory to initialize a parameter value before using it in a calling method.	A parameter value must be initialized within the calling method before its use.
When we use REF, data can be passed bi-directionally.	When we use OUT data is passed only in a unidirectional way (from the called method to the caller method).
Both ref and out are treated differently at run time and they are treated the same at compile time.	
Properties are not variables, therefore it cannot be passed as an out or ref parameter.	

https://baldur.gitbook.io/net-c/.net-c-interview-questions/c-.net-interview-questions-and-answers-part-1

Hodnotové typy - předávání hodnotou

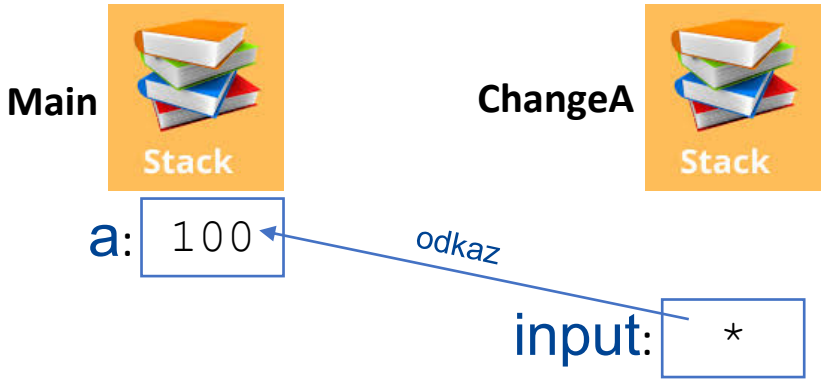
```
int a = 100;

ChanageA(a)
```




```
void ChanageA(int input)
{
    input = 5;
}
```

Hodnotové typy – předávání odkazem (*reflout*)

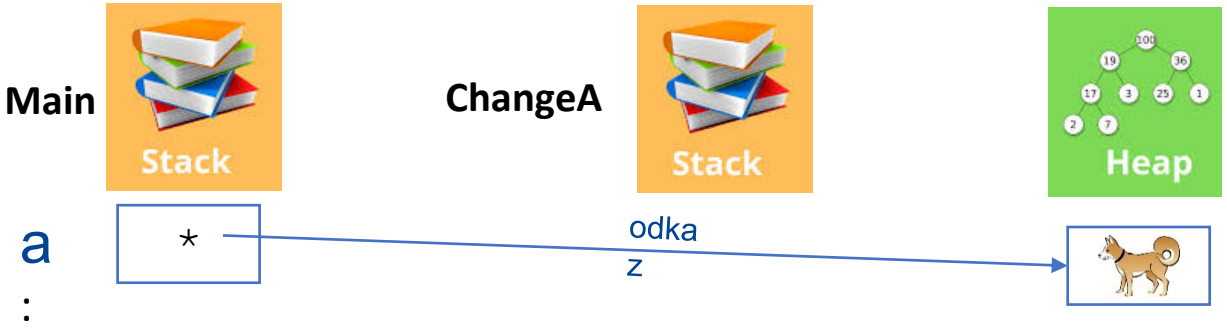
```
int a = 100;

ChanageA(out a)
```



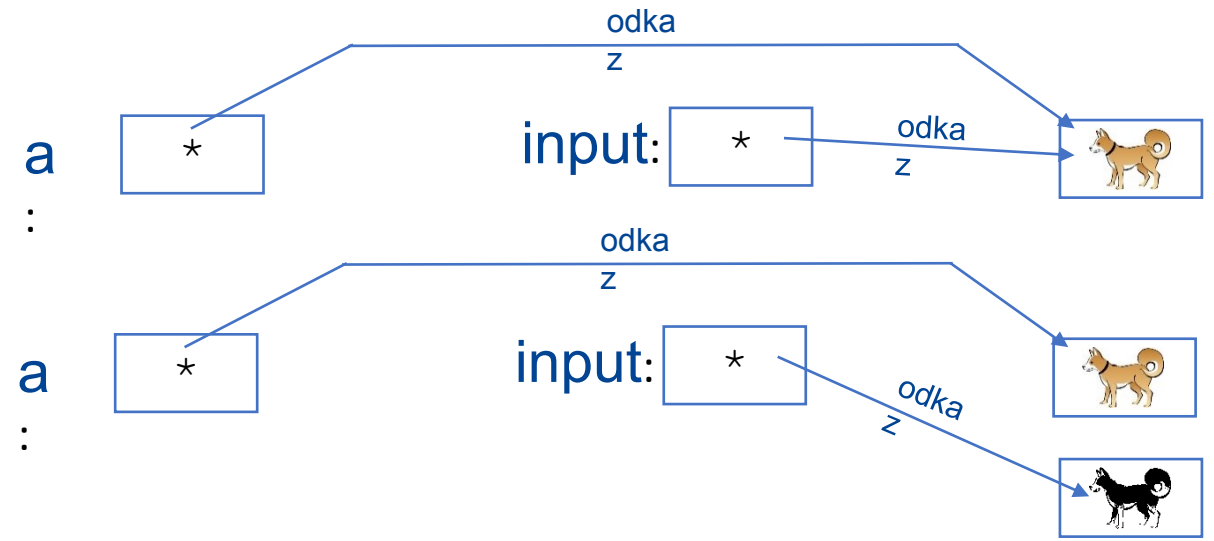

```
void ChanageA(out int input)
{
    input = 5;
}
```

Referenční typy – předávání hodnotou (hodnota = odkaz na heap)

```
object dog = new object();

ChangeDog(dog)
```

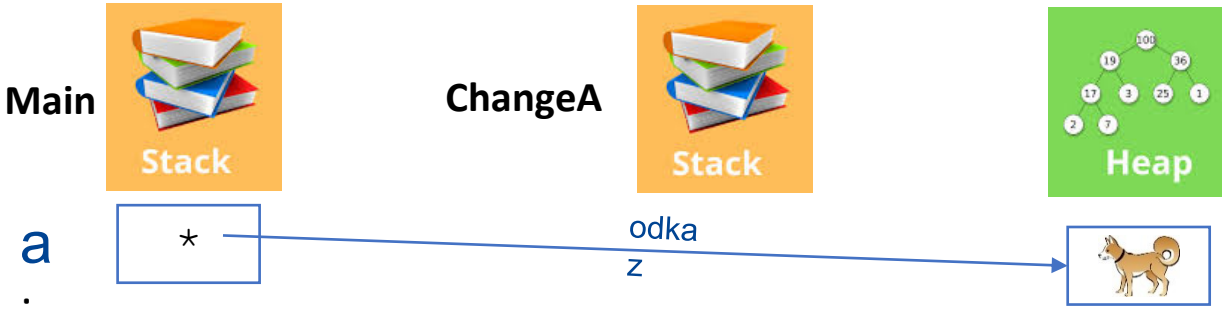



```
void ChangeDog(object intput)
{
    input = new object();
}
```

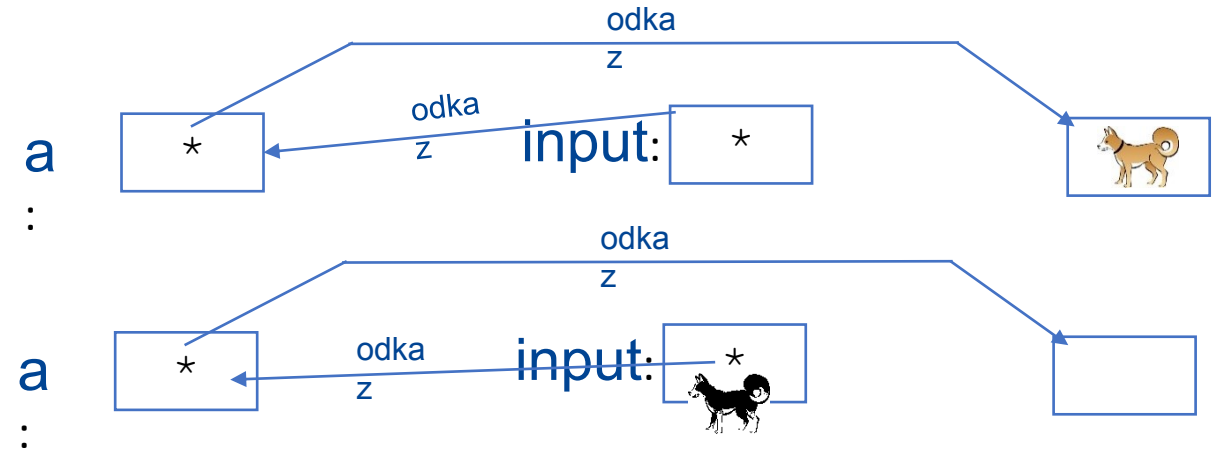


Referenční typy – předávání odkazem (*ref*/*out*)

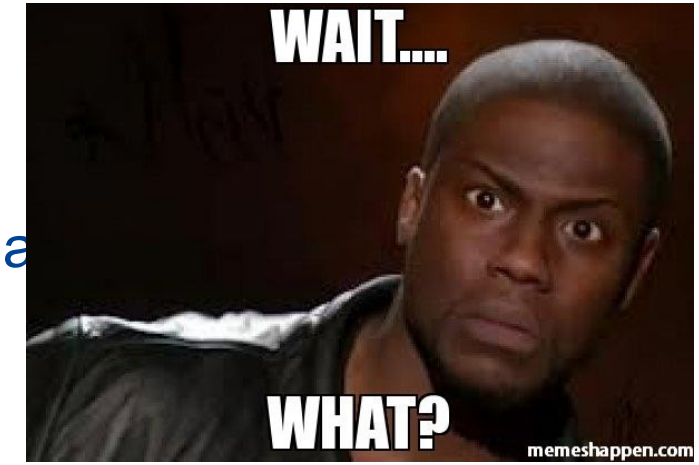
```
object dog = new object();

ChangeDog(ref dog)
```




```
void ChangeDog(ref object input)
{
      input = new object();
}
```



A co `string`?



- Tváří se jako hodnotový typ, ale je to referenční typ!
- Neměnný = po jeho vytvoření nelze měnit jednotlivá písmena
- Má spoustu vlastností jako hodnotový typ:
  - nevytváří se přes ***new***
  - ***==*** funguje jako u hodnotových typů
- Ale má i vlastnosti referenčních typů
  - vytvořen na heapu (haldě)
  - do metod se předává pouze jako odkaz, nevytváří se nový string

Pro zájemce – na doma ;-)

1. Do solution přidejte **existující** projekt `StoneMoveApp`
2. Otevřete projekt a vytvořte ve vlastním souboru abstraktní třídu `Place` obsahující vlastnost `string Name` inicializovanou přes konstruktor
3. Vytvořte ve vlastním souboru rozhraní `IVehicle` s metodami (všechny vrací void)
   `LoadStone()`
   `UnloadStone()`
   `Move(Place place)`
4. Změňte existují třídy `Factory` a `Quarry` tak, aby dědily ze třídy `Place`
5. Změňte třídu `Truck` tak, aby implementovala rozhraní `IVehicle`
6. Doimplementujte metodu `UnloadStone` ve třídě `Truck` (nákladní vozidlo vyloží svůj náklad do továrny)
   Pozor: nákladní auto musí být naložené a musí být v továrně!
   Note: Inspirujte se v metodě `LoadStone`
7. Dopište kód cyklu v metodě `main` (auto jede ke do lomu, nakládá kameny, jede do továrny, vyloží kameny)