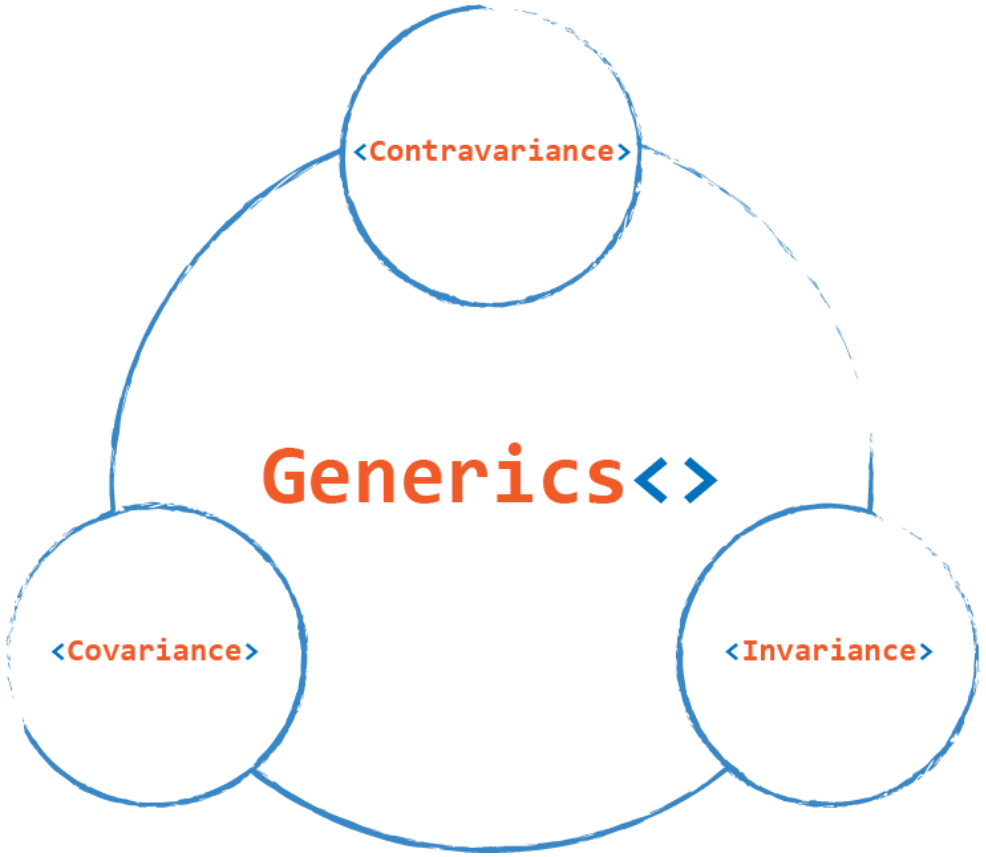
PV178 – Lab 04

Karel Jiránek


Kahoot!

Agenda

- Kolekce
- Extension methods
- Generika
- Kovariance a kontravariance
- Výjimky

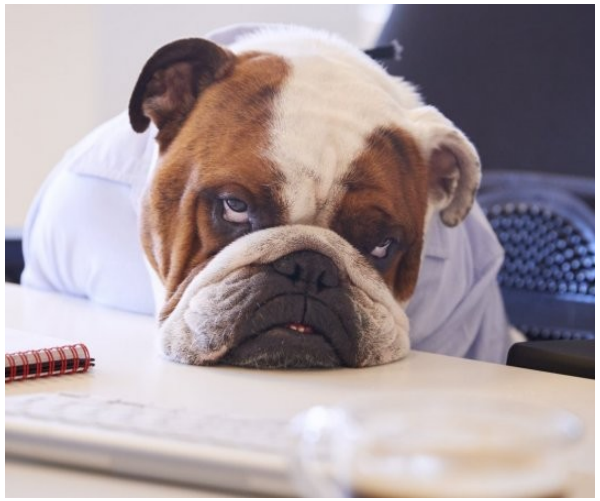


Samostatná práce

- Úkol Collections

Kdyby neexistovala generika (generické datové typy)

```
void Do(int i){ }

void Do(double i){ }

void Do(uint i){ }

void Do(double i){ }

void Do(object i){ }

void Do(float i){ }

...
```



Generické datové typy

- Zamezují opakování kódu – lepší znuvupoužitelnost
- Používají parametrizované typy – při překladu nahrazeny typy konkrétními
- Parametrizovaný typ T se zapisuje za název objektu mezi symboly <, > (<T>)

```
static void Swap<T>(ref T first, ref T second)
{
        var temp = first;
        first = second;
        second = temp;
}

public class MyList<T>
{
        public T[] items { get; set; }
}
```

Generické datové typy - příklady

1

```
static void Swap<T>(ref T first, ref T second)
{
        var temp = first;
        first = second;
        second = temp;
}

int a = 5;
int b = 10;
Swap(ref a, ref b);          // zápis A - přeferovaný
Swap<int>(ref a, ref b);     // zápis B
```

Nepovinné překladač si umí domyslet

2

```
var myList = List<int>();
```

Omezení generického typu

- Omezení se provádí pomocí klíčového slova `where`
- `where` omezuje typy, které mohou být použity pro parametrizovaný typ `T`
- Nejčastější omezení: „třída musí implementovat rozhraní“

```
public class MyList<T> where T: IList
{
        public T items { get; set; }
}
```

„`T` je datový typ, který musí implementovat `IList`, aby mohl být použitý jako parametrizovaný typ“

Samostatná práce

- Úkol GenericsAndMethodOverloading

Generické datové typy – a  a!

- Parametrizovaný typ T – konvence říká, že by se měl jmenovat T, ale může mít jakékoliv jméno

```
static void Swap<Whatever> (ref Whatever first, ref Whatever second)
{
        ...
}
```

- Metoda, třída (atd.) může mít i více parametrizovaných typů!

```
static void Do<T1, T2, T3, ...> (T1 a, T2 b, T3 c, ...)
{
        ...
}
```

Kovariance

- „Umožňuje předat derivovaný typ (poděděný typ) tam kde je očekáván bázový ty
  ==


```
// Třídy
public class Dog {}
public class Husky : Dog {}


// Použití
Dog husky = new Husky();
```

Je očekáván typ `Dog`, ale já dávám typ `Husky` == (kovariance)

```
// Obráceně to nejde !!!
Husky husky = new Dog();
```

Extension Methods

- Umožňují rozšířit existující datový typ dodatečnými **statickými** metodami
- Definují se ve **statické** třídě
- Typ kterého se metoda týká se uvádí jako první parametr metody s klíčovým slovem *this*

```
// Definice
public static class ExtensionMethods
{
        public static void DumpToConsole(this string str)
        {
             Console.WriteLine(str);
        }
}

// Volání
var myString = "my string";
myString.DumpToConsole();
```

Samostatná práce

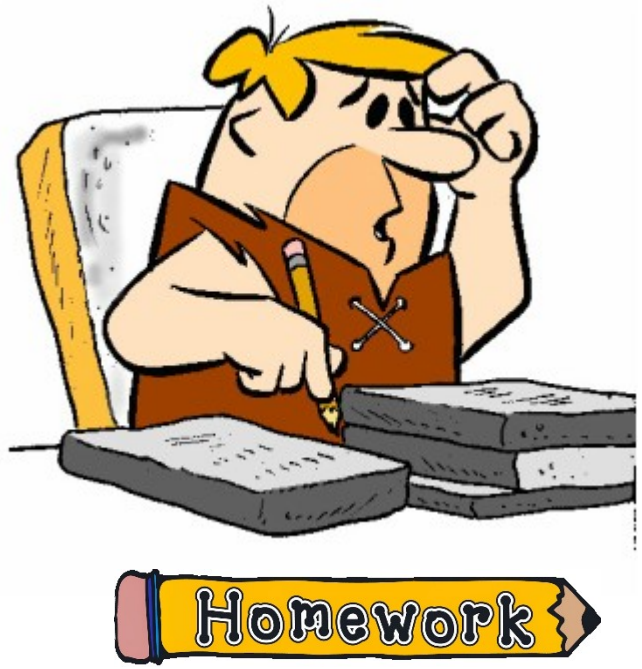
- Úkol ExtensionMethods

Kontravariance (na doma)

- Opak kovariance
- „Umožňuje předat bázový typ tam kde je očekáván derivovaný typ (poděděný typ)“
- https://stackoverflow.com/questions/2662369/covariance-and-contravariance-real-world-example
- Souvisí s použitím klíčového slova `in`

Samostatná práce

- Úkol Kontravariance – doma pro zájemce



Výjimky

- Umožňují reagovat na chyby za běhu programu
- Jsou nehlídané – musíme je hlídat my (try, catch, finally)
- Objektově orientované – výjimka == objekt
- Měly by být dokumentovány
- Povolené zápisy `try-catch`, `try-finally`, `try-catch-finally`
- `catch` bloků může být za sebou více (odchytávají jinou výjimku, provede se vždy jen jeden)
- `finally` blok slouží nejčastěji k uvolnění zdrojů (obrázků, souborů apod.)

Povolené zápisy zpracování výjimek

- Pomocí `try catch finally` bloku

```
// try - catch
try
{
        ...
}
catch (Exception exc)
{
        ...
}
```

```
// try – finally
// můžeme použít using
try
{
        ...
}
finnaly
{

}
```

```
// try – catch – finally
try
{
        ...
}
catch (Exception exc)
{
        ...
}
finnaly
{
        ...
}
```

Mechanismus výjimek

1. V `try` bloku operace způsobí výjimku
2. Provádění programu jde okamžitě do `catch` bloku, který odchytává daný typ výjimku (když existuje)
3. `catch` blok zpracuje výjimku
4. Zpracování jde do `finally` bloku (když existuje)
5. Provede se `finally` blok
6. Program dále pokračuje jako kdyby k výjimce nedošlo

Vyhození a opětovné vyhození výjimky

- K vyhození nové výjimky se používá klíčové slovo `throw`

```
throw new ArgumentException("new exception");
```

- V `catch` bloku můžeme výjimku znovu vyhodit (poslat dál)

```
catch (Exception exc)
{
        // resetuje callstack a „pošle výjimku dál“
        throw exc;

        // zachová callstack a „pošle výjimku dál“ – NEJČASTĚJŠÍ MOŽNOST
        throw;

        // zachová callstack, zaobalení starou výjimku do nové výjimka
        throw new Exception("new exception", exc);
}
```

Text výjimky - doporučení

- Zprávy typu `"something went wrong"` a `"Ups..."` jsou nepřijatelné
- V texu výjimky se snažte popsat CO se stalo, JAK k tomu došlo, popřípadě JAK to napravit

- Příklad (soubor nenalezen)
- `"Cannot open configuration file C:\User\config.txt because it does not exist. Please check the path"`

Samostatná práce

1. Vytvořte nový projekt *Exceptions* a přidejte ho do stávající Solutiony
2. Vytvořte třídu *DynamiteException*, která dědí ze třídy *Exception*
3. Ve třídě vytvořte vlastnost *Power*, která bude mít jen getter
4. Třída bude mít 2 konstuktory

    - konstruktor A – 2 parametry – `string` *message* a `int` *power* (power inicializuje vlastnost Power)

    - konstruktor B – 3 parametry – stejné jako konstruktor A + parametr `Exception` *ex*

5. Parametry konstruktoru *`message`* a *ex* předejte do konstruktoru bázové třídy
6. Ve třídě Program vytvořte metodu DynamiteMethod, která vyhodí výjimku *Dynamite exception* se zprávou „**My dynamit exception**“ a sílou (power) **20**
7. V metodě Main tuto metodu zavolejte v try bloku
8. V catch bloku odchyťte tuto výjimku a vypište zprávu výjimky do konzole
9. Výjimku znovu v catch bloku vyhoďte, tak aby byl zachován stacktrace

Poučení + linky

- **Když musím vyhodit výjimku ručně, tak vyberu výjimku, jejíž název nejlépe popisuje situaci, která nastala**

- Používání výjimek

  https://docs.microsoft.com/en-us/dotnet/standard/design-guidelines/using-standard-exception-types

- Seznam typů výjimek

  https://www.completecsharptutorial.com/basic/complete-system-exception.php