

Faculty of Mathematics and Physics
Charles University in Prague
17th March 2016



C# Made Easy!

Programming II

Workshop 4 – The Snake and OOP

Workshop 4

Outline

1. (No) Test
2. The Snake - Reloaded



(No) Test 04

Quick Warm up!

Find the test here (no-ads):

<https://goo.gl/zozqz7>

Permanent link:

https://docs.google.com/forms/d/1cl8d3LcijqX8DoIjX8wiHAh2RpRDPRq_KW-ND5-SKyc/viewform

Time for no-test:

5 min

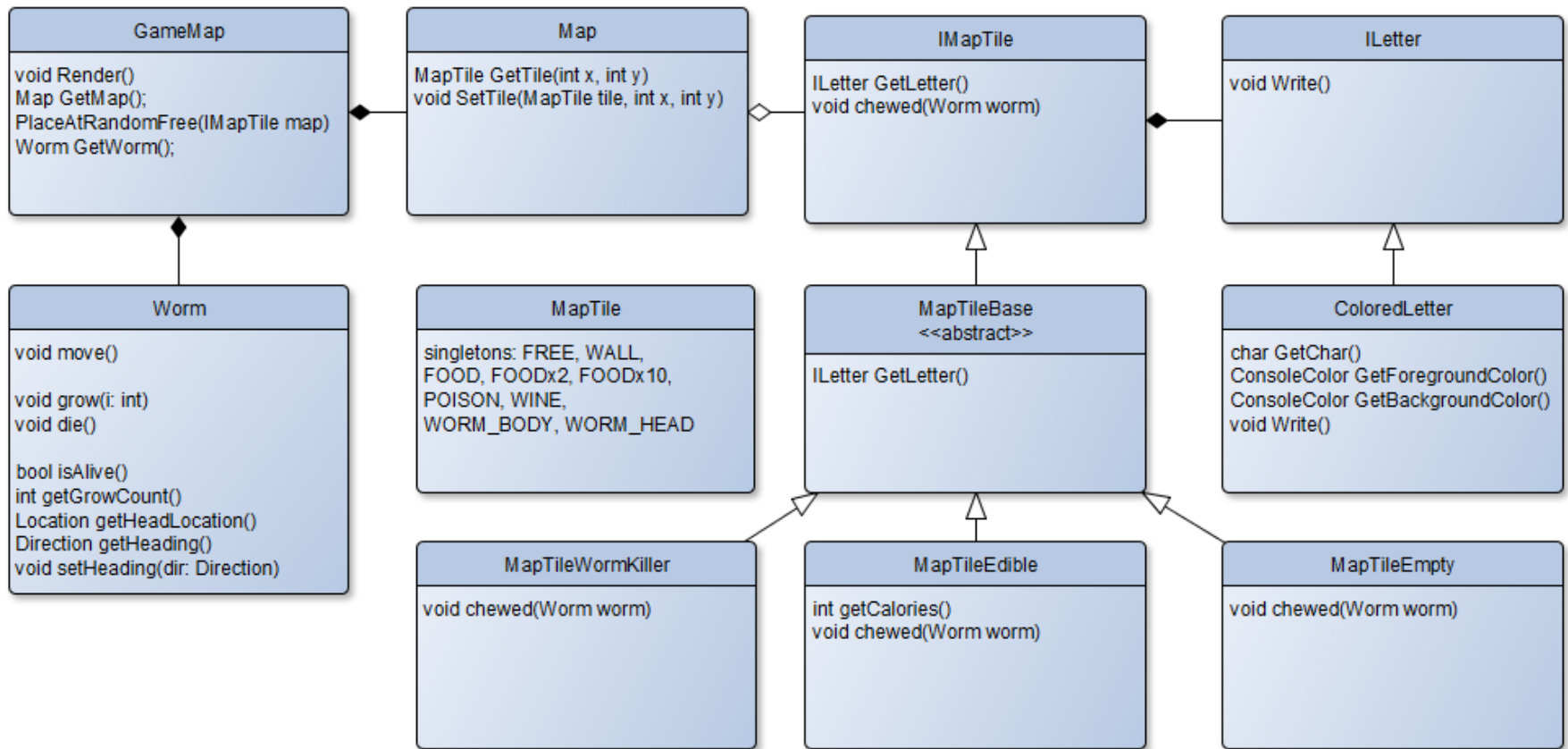
Assignment (3+)4

Create "The Snake" aka "The Worm" game

- Console application (text-based game)
 - Snake is controlled either by WSAD, or arrows, or 8246 numbers
 - You have to display "snake's length" and "time elapsed" somewhere at "bottom status line"
 - Escape key terminates the game
 - Implement tiles: Empty, Wall, Food, Posion, Wine, "Snake"
 - Snake movement speed should be increasing every N food eaten (regardless their calories)
- Provide solution that can easily change
 - Map dimensions
 - Ask me on how big map I want to play (max 80 x 40)
 - Food effects
 - Adding new food should be as easy as creating and „registering“ a new Food object
 - Both POISON and WALL kill the snake
 - Implement WINE so it reverse the controls (UP<->DOWN, LEFT<->RIGHT)
- Points:
 - 10, if finished till 24.3.2016, 23:59
 - 5, if finished later on
- Bonus:
 - Provide different visualization for the snake's body using
 - <, v, >, ^ as a head ... -, | as a straight body and /\ at "turning points"
 - 3 points

Assignment 4

Representing the Game Model + View

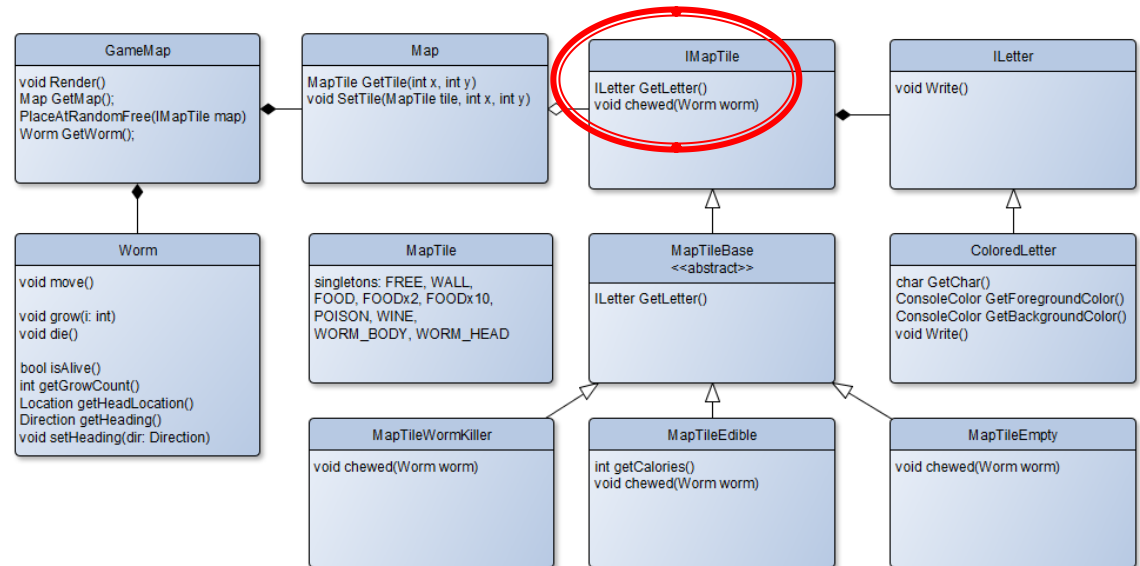


Assignment 4

Representing the Game Model + View

```
using System;

namespace Programko
{
    public interface IMapTile
    {
        ILetter GetLetter();
        void chewed(Worm worm);
    }
}
```



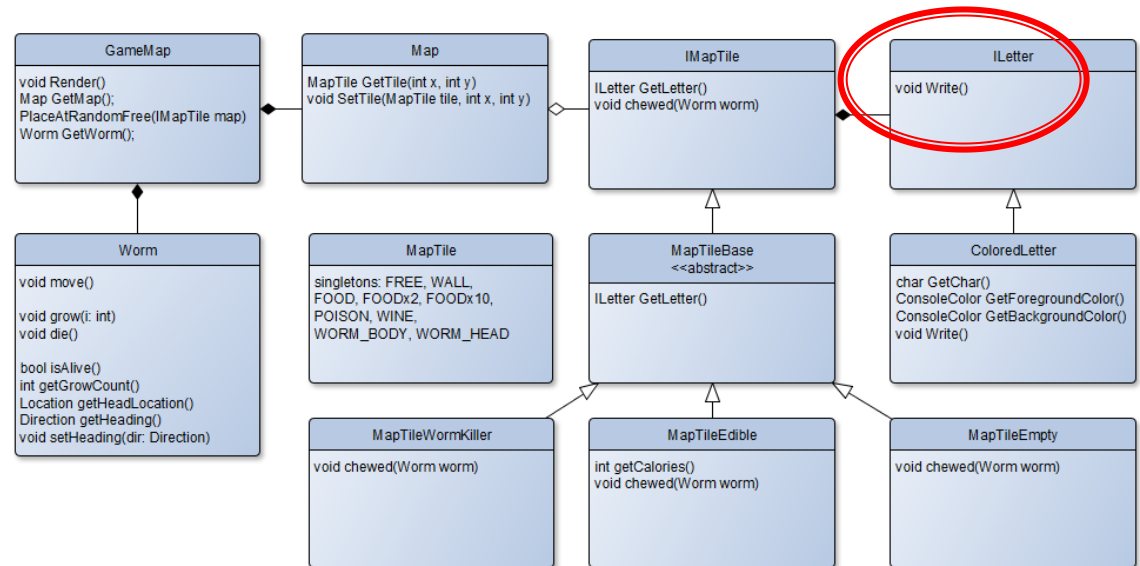
We're going to have "generic" tile described by "letter" and "what happens when the Snake chews the tile".

Assignment 4

Representing the Game Model + View

```
using System;

namespace Programko
{
    public interface ILetter
    {
        void Write();
    }
}
```



And the "letter" will in-fact be anything that can "output" some character to a screen. Again, we're trying to embrace "procedural knowledge" here rather than "symbolic" one.

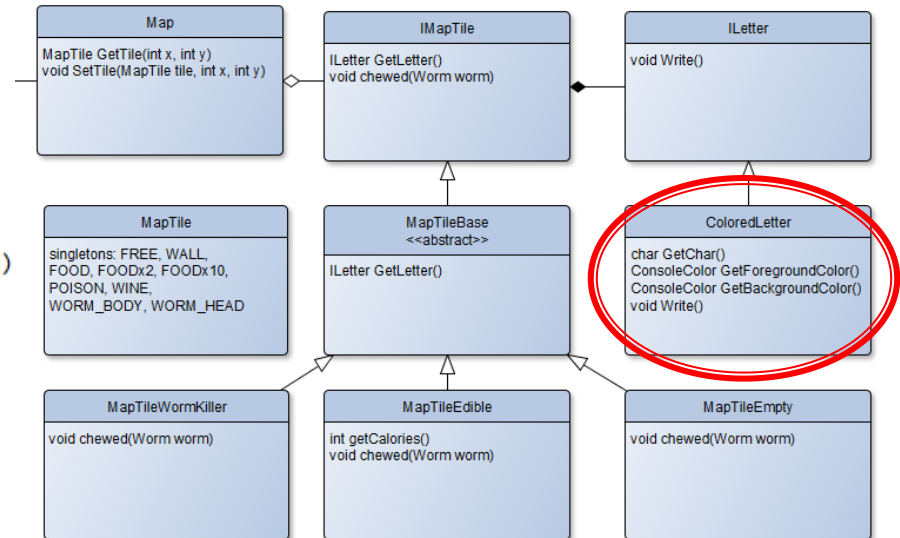
Assignment 4

Representing the Game Model + View

```
using System;

namespace Workshop03
{
    1 reference
    class ColoredLetter : ILetter
    {
        private ConsoleColor fore;
        private ConsoleColor back;
        private char c;
        0 references
        public ColoredLetter(ConsoleColor fore, ConsoleColor back, char c)
        {
            this.fore = fore;
            this.back = back;
            this.c = c;
        }

        1 reference
        public void Write()
        {
            Console.BackgroundColor = back;
            Console.ForegroundColor = fore;
            Console.Write(c);
        }
    }
}
```



And thus we can have an implementation that outputs some character in chosen colors. The trick is, that the "ILetter" is oblivious to what "IMapTile" does ... thus you can pair "visual style of the tile" with "its implementation of chewed()" as you see fit!

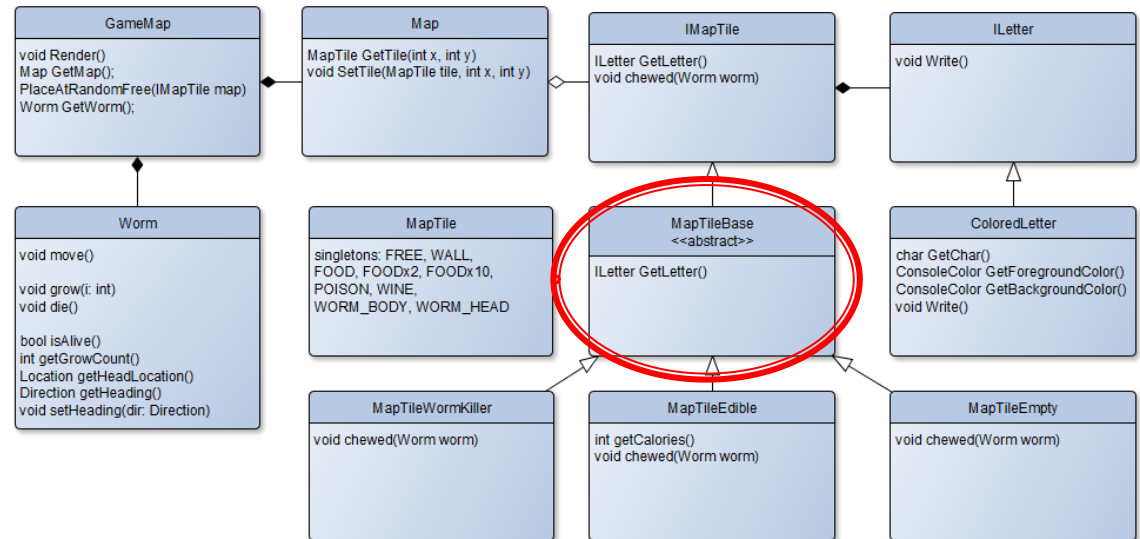
Assignment 4

Representing the Game Model + View

```
using System;
```

```
namespace Workshop03
```

```
{  
    7 references  
    abstract class MapTileBase : IMapTile  
    {  
        ILetter graphics;  
        3 references  
        public MapTileBase(ILetter graphics)  
        {  
            this.graphics = graphics;  
        }  
        1 reference  
        public ILetter getGraphics()  
        {  
            return graphics;  
        }  
    }  
}
```



So, let's define our "abstract" base for all tiles... you know, every tile will need to specify it's ILetter ... composition over inheritance here!

Assignment 4

Representing the Game Model + View

```
using System;
```

```
namespace Workshop03
```

```
{  
    6 references  
    class MapTileEdible : MapTileBase
```

```
{  
    private int calories;
```

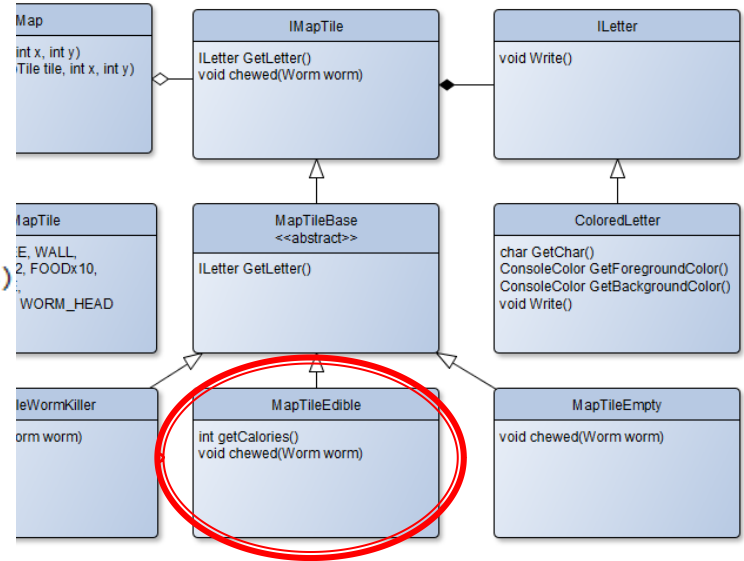
```
    5 references
```

```
    public MapTileEdible(ILetter graphics, int calories) : base(graphics)  
    {  
        this.calories = calories;  
    }
```

```
    0 references
```

```
    public void chewed(Worm worm)  
    {  
        worm.grow(calories);  
    }
```

```
}  
}
```



Now the real deal... MapTileEdible ... here we specify “what” should happen when “the Snake chews” the tile.

Note that apart from “grow” we should place a new food into the map as well!

Assignment 4

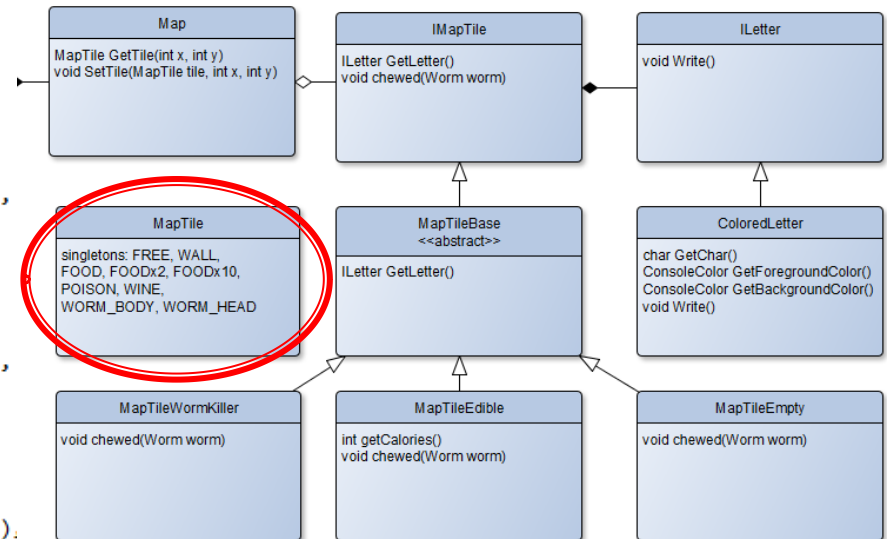
Representing the Game Model + View

```
using System;

namespace Workshop03
{
    4 references
    class MapTile
    {
        public const IMapTile FOOD =
            new MapTileEdible(
                new ColoredLetter(ConsoleColor.Blue, ConsoleColor.Green, '+'),
                1
            );

        public const IMapTile FOODx2 =
            new MapTileEdible(
                new ColoredLetter(ConsoleColor.Green, ConsoleColor.Blue, 'x'),
                2
            );

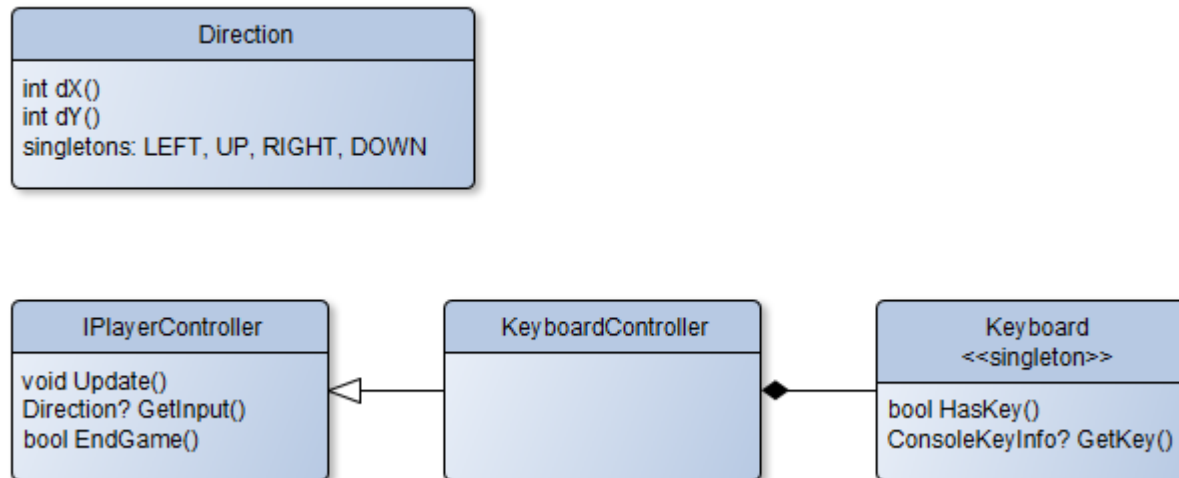
        public const IMapTile FOODx10 =
            new MapTileEdible(
                new ColoredLetter(ConsoleColor.Black, ConsoleColor.Green, 'X'),
                10
            );
    }
}
```



Thus, we can easily define any type of food we can ever dream of ... almost.

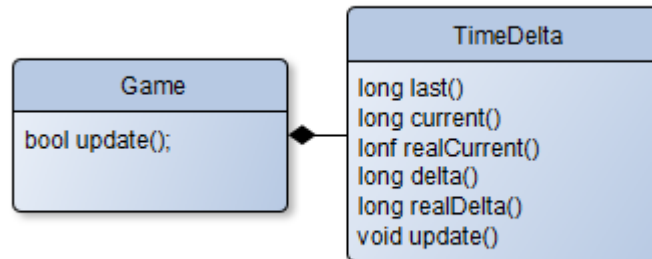
Assignment 4

Getting Player's Input



Assignment 3

Game & Time



CheatSheet

Reading Inputs from Keyboard

```
1 reference  
public bool hasKey()  
{  
    return Console.KeyAvailable;  
}
```

```
1 reference  
public ConsoleKey? getKey()  
{  
    if (Console.KeyAvailable)  
    {  
        return Console.ReadKey(true).Key;  
    }  
    return null;  
}
```

CheatSheet

Changing Console Output Color

- Google: C# Console Colors
 - Google – The Best Programmer’s Friend
 - Keep in mind the limit of “Googling” for “Code”
 - It’s always OK to Google for “API”
 - It’s okay to do the research “how to approach some problem”
 - It’s always bad to act as script kiddies who just copy-paste others’ code without understanding

CheatSheet

Timing your Snake

```
private static readonly DateTime Jan1st1970 = new DateTime(1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
```

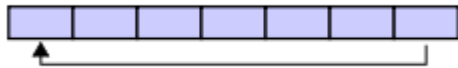
2 references

```
public static long CurrentTimeMillis()  
{  
    return (long)(DateTime.UtcNow - Jan1st1970).TotalMilliseconds;  
}
```


CheatSheet

Implementing the Snake's body

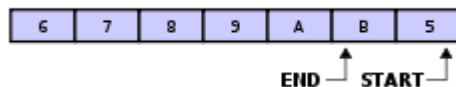
- Circular array / buffer



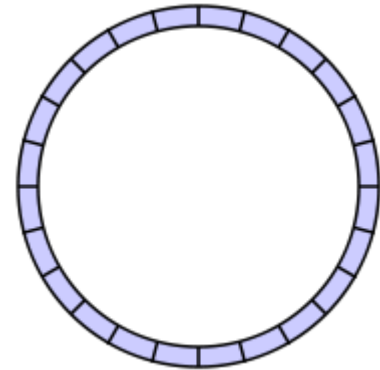
- Array



- Holding „start“ and „end“ index



- Which can overflow...



CheatSheet

Implementing the Snake's body

- Or just use „List“ – array that can change its size automatically

```
List<int> integers = new List<int>();  
integers.Add(1);  
integers.Add(4);  
integers.Add(7);  
  
int someElement = integers[1];
```

Assignment 4

Send me an email

- Email: jakub.gemrot@gmail.com
- Subject: **Programming II – 2016 – Assignment 04**
- Zip up the whole project and send it
- You WILL NOT find the assignment in CoDex!
- Deadline: **24.3.2016 23:59**

Questions?

I sense a soul in search of answers...

- In case of doubts about the assignment or some other problems don't hesitate to contact me!
 - Jakub Gemrot
 - jakub.gemrot@gmail.com