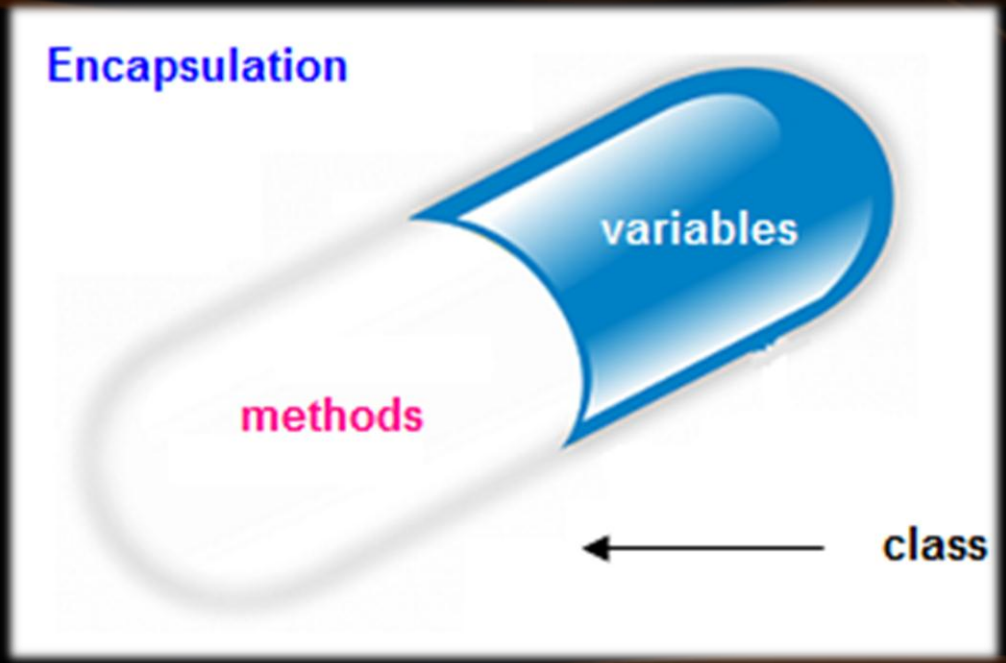
Капсулация и валидация
Модификатори за достъп

Какво е капсулацията и валидацията и какви са ползите от нея

Учителски екип

Обучение за ИТ кариера

https://it-kariera.mon.bg/e-learning/



Съдържание

1. Ключова дума `this`
2. Какво е капсулация?
3. Какво представляват модификаторите за достъп – Private, Public, Protected, Internal
4. Валидация
   - Променими и непроменими типове данни

Ключова дума this

- this е препратка към текущия обект. Той е едно цяло.
  - this сочи към променлива, която е инстанция на текущия клас

```
public Person(string name)
{
  this.name = name;
}
```

  - this може да се предава като аргумент в метод или при извикване на конструктор
  - this може да се върне като стойност на метод

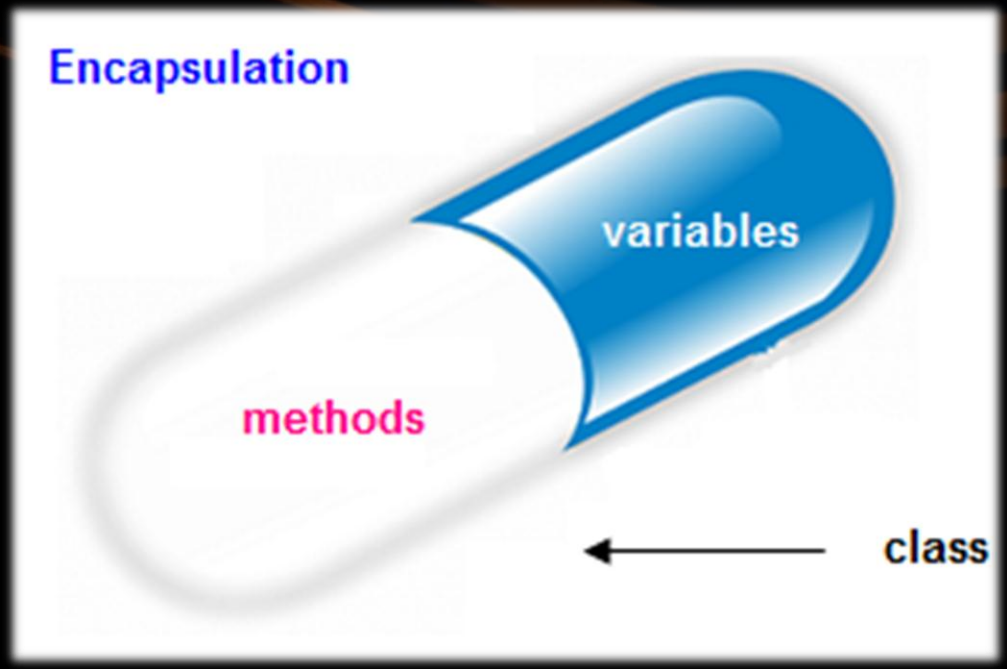
Ключова дума this (2)

- this може да извиква конструктор на текущия клас

```
public Person(string firstName, string lastName)
{
  this.firstName = firstName;
  this.lastName = lastName;
}
public Person(string fname, string lName, int age)
  : this(fName, lName);
{
  this.age = age;
}
```

Капсулация

Какво е капсулацията и ползите от нея



Капсулация

- Процесът на обединяване на кода и данните в едно цяло (обект)
- Полетата трябва да са private

```
private int age;
```

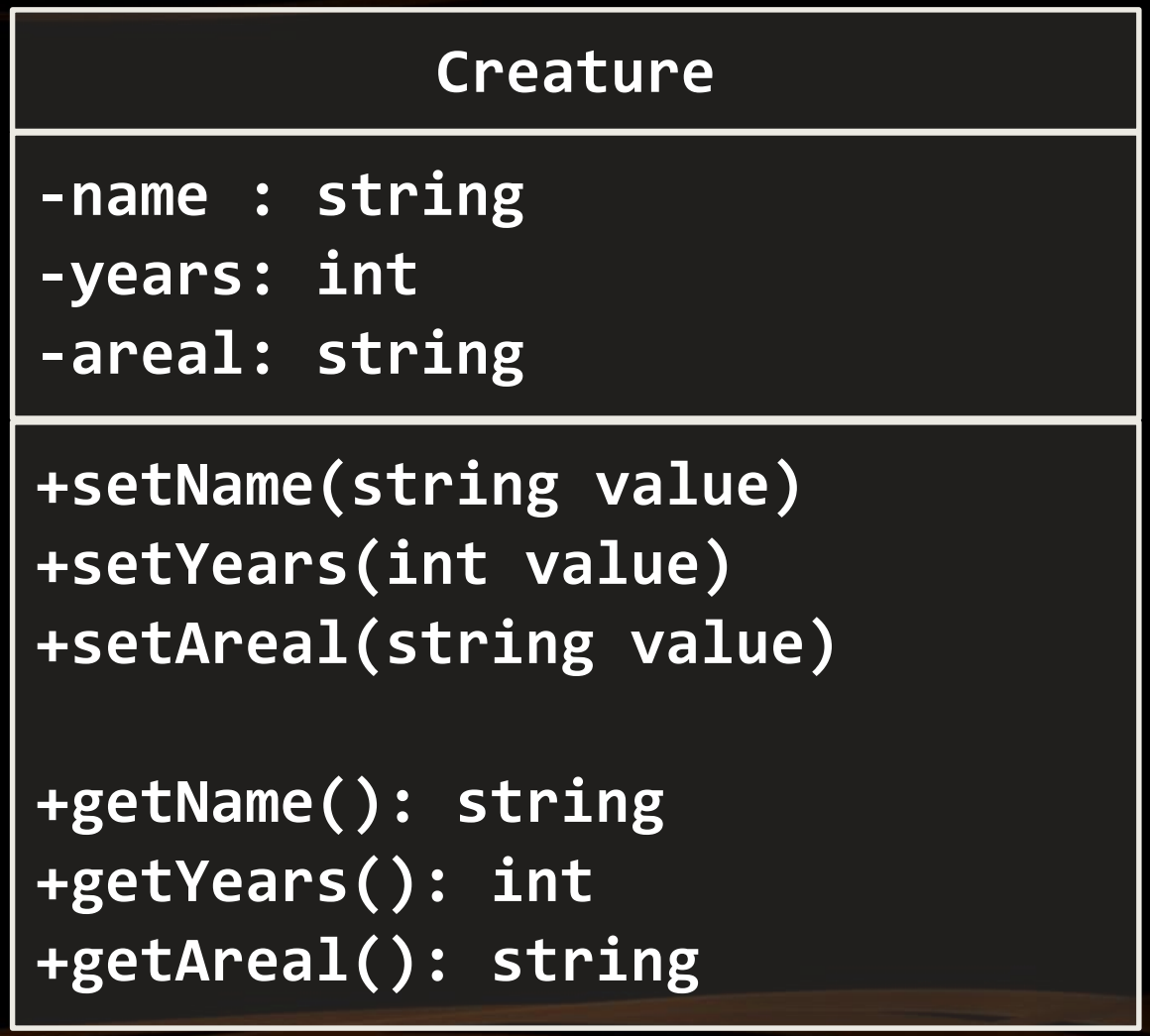
- Използване на getters и setters за достъп до данните

```
class Person
{ ...
  public int Age => return this.age;
}
```

Person
-name: string -age: int
+Person(string name, int age) +Name: string +Age: int

Задача: Клас Creature

- Създайте клас Creature
- Creature трябва да има полета name, years, areal, съответно за име, възраст и местообитание
- Създайте методи за достъп до обектите getters и setters за полетата



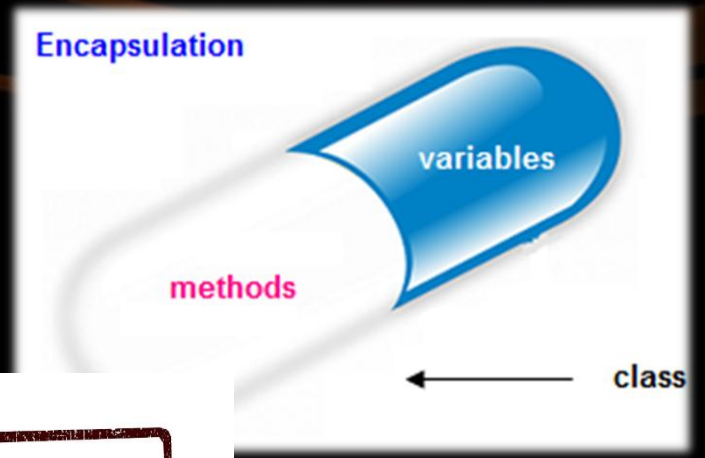
Решение: Клас Creature

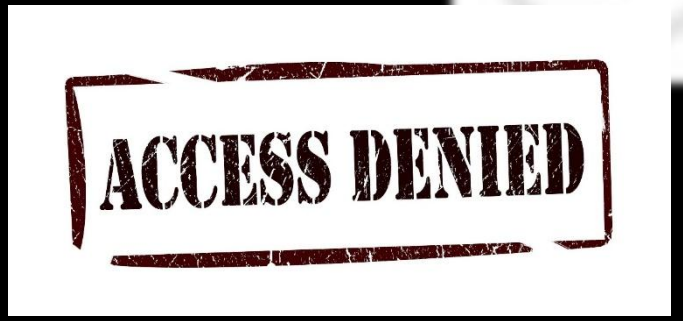
```
private string name;
private int years;
private string areal;

public string Name
{
  get { return this.name; }
  set { this.name = value; }
}
public int Years
{
  get => this.years;
  set => this.years = value;
}
```

```
public string Areal
{
  get
  {
     return this.areal;
  }

  set
  {
     this.areal = value;
  }
}
```

Модификатори за достъп

Видимост на членовете на клас





Private Модификатор за достъп

- Основен начин за капсулиране на обект и скриване на данни от външния свят

```
private string name;
Person (string name)
{
  this.name = name;
}
```

- private членовете са достъпни само в декларацията на класа

Protected Модификатор за достъп

- Спира достъпа на външни класове

```
class Person
{
  protected string FullName { get; set; }
}
```

Автоматично се създава поле за съхраняване на информацията

- protected елементите са достъпни само от подкласовете
- Не може да бъде използван за класове и интерфейси

Internal модификатор за достъп

- Това е модификатора по подразбиране за класове в C#

```
class Person
{
  string Name { get; set; }
  internal int Age { get; set; }
}
```

класът ще бъде internal

полето обаче ще е private

- internal дава достъп на всеки друг клас в същия проект

```
Person p = new Person();
p.Name = "Ivan";
```

Public модификатор за достъп

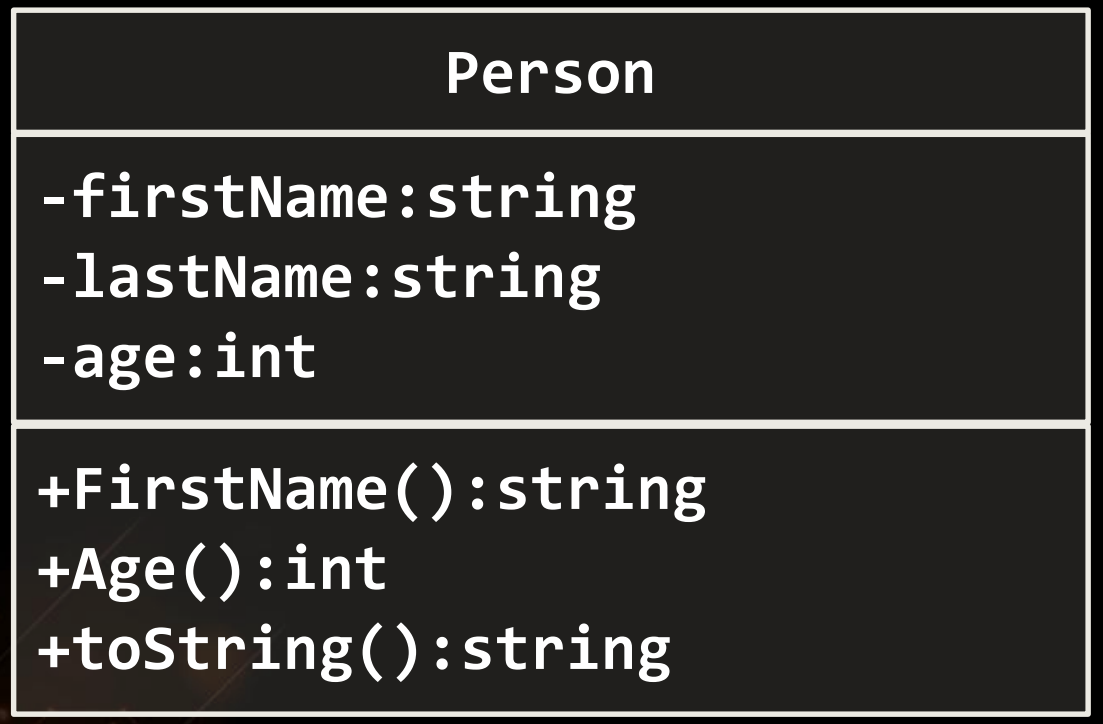
- Ако искаме да достъпим public клас в друг namespace

```
public class Person
{
  public string Name { get; set; }
  public int Age { get; set; }
}
```

- public елемент, деклариран в public клас може да бъде достъпен от всеки клас, принадлежащ на .NET Света
- Методът `main()` в приложението трябва да е public

Задача: Подредете Persons по Name и Age

- Създайте клас Person



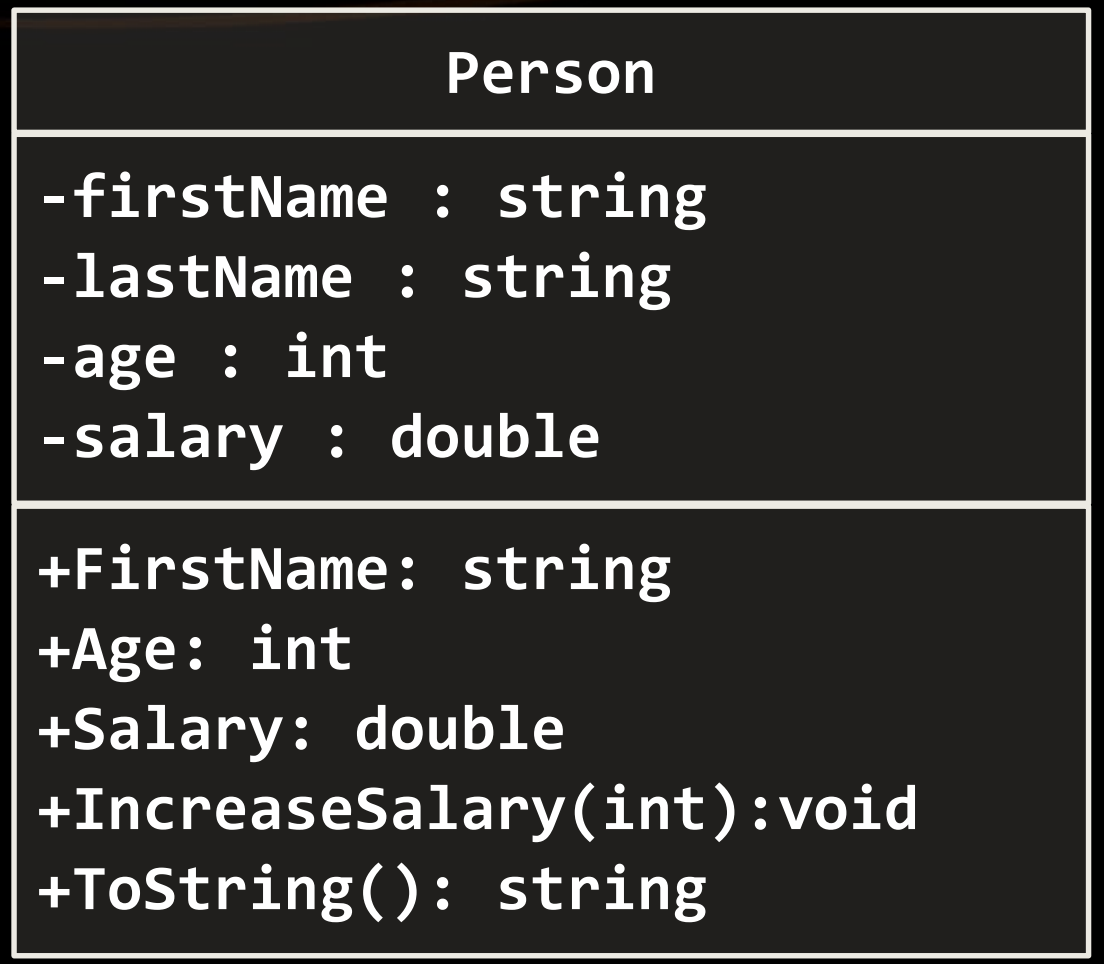

```
public static void Main()
{
    var lines = int.Parse(Console.ReadLine());
    var persons = new List<Person>();
    for (int i = 0; i < lines; i++)
    {
        var cmdArgs = Console.ReadLine().Split();
        var person = new Person(cmdArgs[0],
                                cmdArgs[1],
                                int.Parse(cmdArgs[2]));
        persons.Add(person);
    }

    persons.OrderBy(p => p.FirstName)
           .ThenBy(p => p.Age)
           .ToList()
           .ForEach(p => Console.WriteLine(p.ToString()));
}
```

Задача: Подредете Persons по Name и Age

```
public class Person {
  private string firstName;
  private string lastName;
  private int age;

  public string FirstName => return this.firstName;
  public string LastName => return this.lastName;
  public int Age => return this.age;
  public override string ToString() {
    // TODO
  }
}
```

Задача: Увеличение на заплатата

- Добавете към Person salary
- Добавете getter за заплата
- Добавете метод, който променя заплатата с даден процент
- Хора, по-млади от 30 години вземат половината от увеличението



Решение: Getters and Setters

- Разширяваме Person от предишната задача

```
private double salary;
public void IncreaseSalary(double percent)
{
  if (this.age > 30)
    this.salary += this.salary * percent / 100;
  else
    this.salary += this.salary * percent / 200;
}
```

Валидация

Проверка при въвеждане
на данните на обекта

Валидация

- Валидацията ни гарантира, че винаги работим с валидни данни
- Тя е възможна заради капсулирането и се се случва в setters

```
public double Salary
{
  set
  {
    if (value < 460)
      throw new ArgumentException("...");
    this.salary = value;
  }
}
```

По-добре е да се „хвърли“ изключение, отколкото да се извежда на екрана

- Някой трябва да се грижи за обработка на изключенията

Валидация (2)

- Конструкторите използват private setter с валидационна логика

```
public Person(string firstName, string lastName,
              int age, double salary)
 {
  this.FirstName = firstName;
  this.LastName = lastName;
  this.Age = age;
  this.Salary = salary;
}
```

Валидацията се случва в setter-а

- Гарантират валидно състояние на обекта при неговото създаване

Задача: Валидация на данни

- Разширете Person с валидация за всяко поле
- Names трябва да са с не по-малко от 3 символа
- Age не може да е нула или отрицателно
- Salary да не е по-малко от 460

Person
-firstName : String -lastName : String -age : Integer -salary : Double
+Person() +FirstName(string fname) +LastName(string lname) +Age(int age) +Salary(double salary)

Задача: Валидация на данни

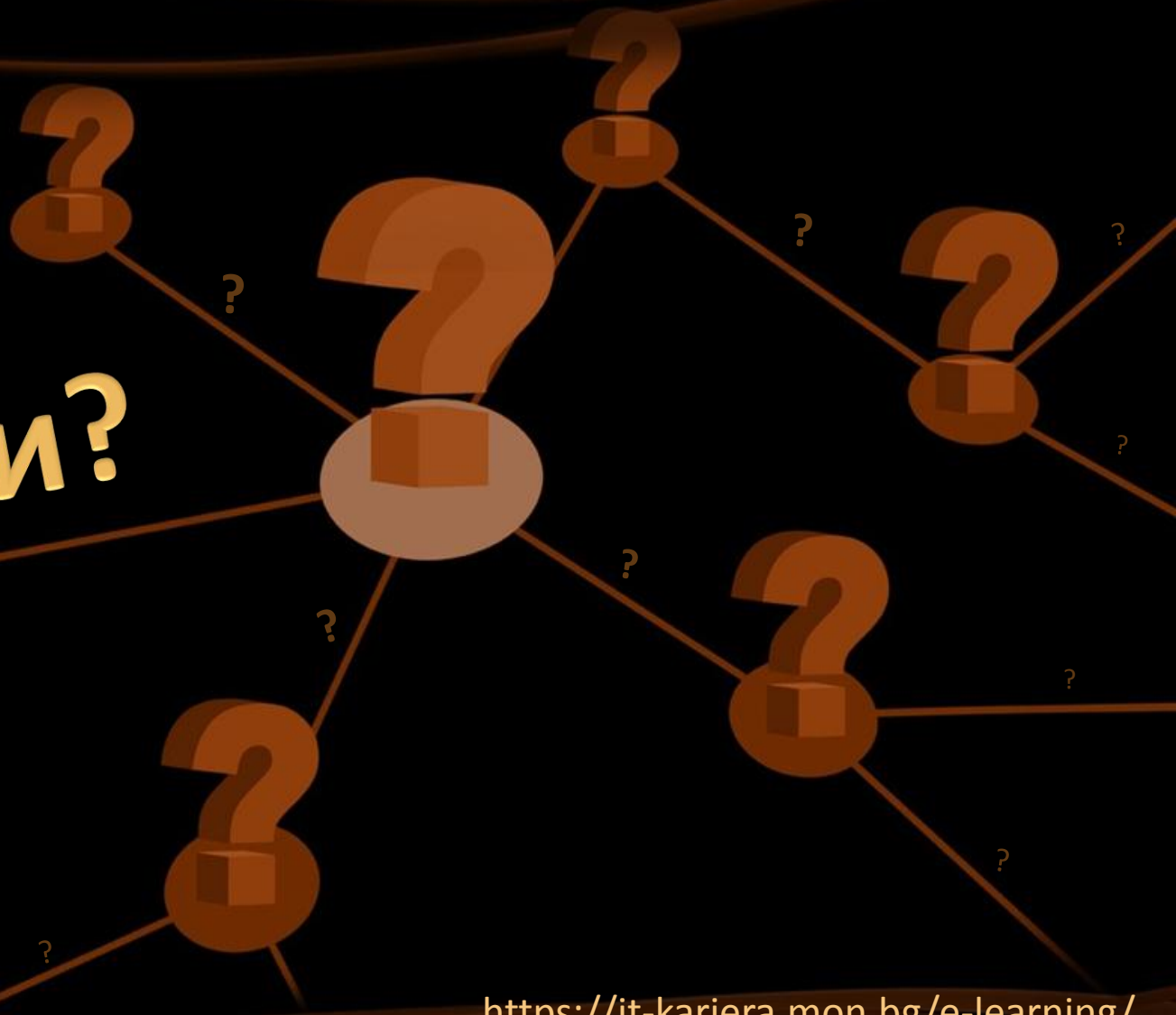
```
// TODO: добавете валидация за firstName
// TODO: добавете валидация за lastName
public int Age
{
  set
  {
    if (value < 1)
      throw new ArgumentException("...");
    this.age = value;
  }
}
// TODO: добавете валидация за salary
```

Обобщение – капсулация и валидация

- Капсулацията скрива програмната реализация
- Структурните промени остават локални
- С this си осигуряваме достъп до инстанциите
- Модификаторите определят степента на капсулация:
  - Private - за полета и методи, вътрешни за класа
  - Protected – за наследниците (подкласовете)
  - Internal – за класовете от същия проект (namespace)
  - Public – за класове и интерфейси в целия .Net свят
- Валидацията ни гарантира, че обектът съдържа валидни данни

Капсулация

Въпроси?



https://it-kariera.mon.bg/e-learning/

Лиценз

- Настоящият курс (слайдове, примери, видео, задачи и др.) се разпространяват под свободен лиценз "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International"

- Благодарности: настоящият материал може да съдържа части от следните източници
  - Книга "Основи на програмирането със C#" от Светлин Наков и колектив с лиценз CC-BY-SA