Изпитна тема № 3: Обектно-ориентирано програмиране

# План-тезис:

Дефиниране на класове: клас, конструктор, полета, свойства, създаване на обекти от клас. Дефиниране на функции/методи в класовете, ключова дума this. Енкапсулация на данни в класовете, методи за достъп и промяна на полета (getters/setters). Статични полета и методи в класовете.

## Клас/ Class

**Д**ефинира се чрез състояние (полетата) и поведение (методите).

## Полетата/ Fields

Полетата описват състоянието на класа. Те имат тип и име.

## Свойства / Properties

Използват се за създаване на методи на четене и методи на промяна (getters – предоставят достъп до полето, setters – позволяват промяна на полето).

## Метод / Method

Именувано парче код, което може да се извиква. Има тип на връщаната стойност, име, параметри, които приема, и тяло. От своя страна всеки параметър си има име и тип. Методът може да има много варианти според различните сигнатури.

В ООП методите могат да променят състоянието на обектите или да го достъпват и анализират.

### Сигнатура на метод

Комбинацията от име и параметри на един метод.

## Конструктори/ Constructors

Специален вид методи, които се извикват при създаване на обекта. Може да имате множество конструктори за даден клас (конструктор без параметри, конструктори с параметри). Задават началното състояние на обекта. Могат да се извикват един друг.

## Капсулация/ Encapsulation

Процесът на обединяване на кода и данните в едно цяло (обект), което включва полетата на обекта да са private и да се използват getters и setters за достъп до данните.

Скрива имплементацията (реализацията на обекта - неговите компоненти – полета, свойства, методи) и намалява сложността. Осигурява структурните промени да останат локални.

### Модификатори за достъп/ Modifiers

Класовете и елементите на класа имат модификатори, които определят видимостта им. С помощта на модификаторите за достъп можем още да извършваме валидация на данните.

* **Private** - основен начин за капсулиране на обект и скриване на данни от външния свят. Класовете и интерфейсите/ interfaces не могат да са private. Идеята за интерфейс е да се даде възможност за връзка с „външния свят“ – т.е. – трябва да са достъпни (public)
* **Protected** - достъпни само от подкласове. Също не може да бъде приложен за класове и интерфейси. Предотвратява външни класове да се опитват да го използват
* **Internal** - модификатор по подразбиране в C#. Дава достъп на всеки друг клас в същия проект.
* **Public** - може да бъде достъпен от всеки клас, принадлежащ на .NET Света. Употребата се налага ако се опитваме да достъпим public клас в друг namespace. Интерфейсите са public. Тъй като смисълът им е да дават връзка с външния свят.

### Валидация/ Validation

Случва се в setter-ите. Конструкторите използват private setter с валидационна логика, което гарантира валидно състояние на обекта при неговото създаване.

### Ключова дума “this”

“this” е препратка към текущия обект. Mоже да сочи към променлива, която е инстанция (представител) на текущия клас, да се предава като аргумент в метод или като извикване на конструктор и да се връща като стойност на метод.

## Статични полета/ Static fields

Статичните полета в класа имат една и съща стойност за всеки обект, могат да бъдат достъпени и само чрез класа - без създаване на обект от този клас.

## Статични свойства/ Static properties

Статичните свойства в класа принадлежат на самия клас и могат да бъдат достъпвани само чрез класа - без създаване на обект от този клас.

Използването на статични свойства е удобно, когато имаме статични полета, но не искаме да позволим тяхната промяна в друг клас, който използва нашия. Ако трябва да използваме само поле, то за да го достъпим извън класа, трябва да е public, което пък би позволило неговото изменение. Именно тук идват статичните свойства (Удобни за достъпване на информация от статични полета, която не бива да се променя от ползвателя на класа ни).

## Статични методи/ Static methods

Статичните методи в класа принадлежат на самия клас и могат да бъдат достъпвани само чрез класа - без създаване на обект от този клас.

Удобни са за извършване на действия върху всички обекти от класа или за извършване на действия, които нямат пряко отношение към обектите.

## Статични класове/ Static classes

За да не се допуска създаване на обект от даден клас, който има само статични членове ние можем да поставим думата static пред class: static class.

Когато отбележим един клас като статичен това означава, че неговите членове също ще са статични и от този клас няма да може да се създават обекти, а ще може членовете му да се ползват само статично. Много класове от .NET са статични (например Math).

## Статични конструктори/ Static constructors

Конструкторите в един клас също могат да бъдат статични. Ако един конструктор е статичен той се изпълнява, когато едно от тези събития се случи за първи път:

* Създаде се обект от класа (ако той е нестатичен).
* Достъпва се статичен член от класа.

Най-често статични конструктори се използват за инициализация на статични полета

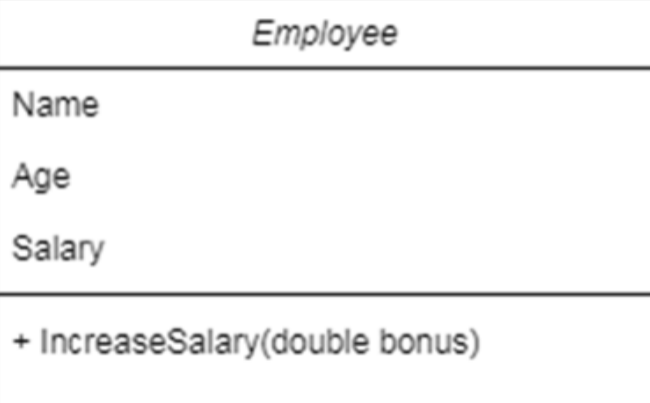
# Примерна приложна задача:

По време на теоретичния изпит се предоставя непълен/неработещ/некоректен програмен фрагмент на приложната задача. Предоставеният фрагмент да се приведе в работещ вид.

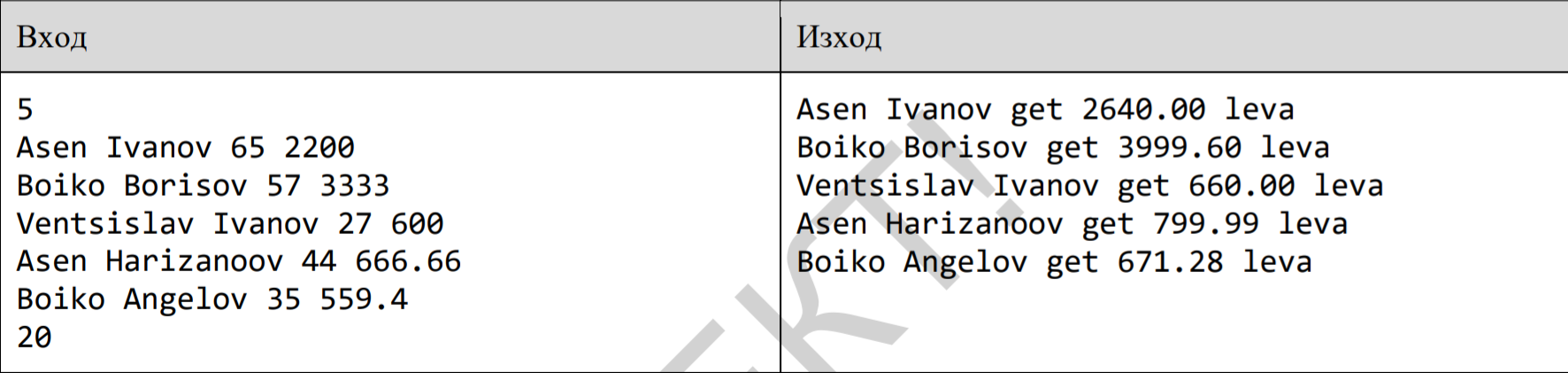
# Условие:

Да се напише програма, която да обработва информация за служители в предприятие. Дадени са служители (Employee) с техните имена, възраст и заплата. При въвеждане на процент бонус, той се добавя към заплатата на всеки служител. Служителите на възраст под 30 получават половината бонус спрямо останалите.

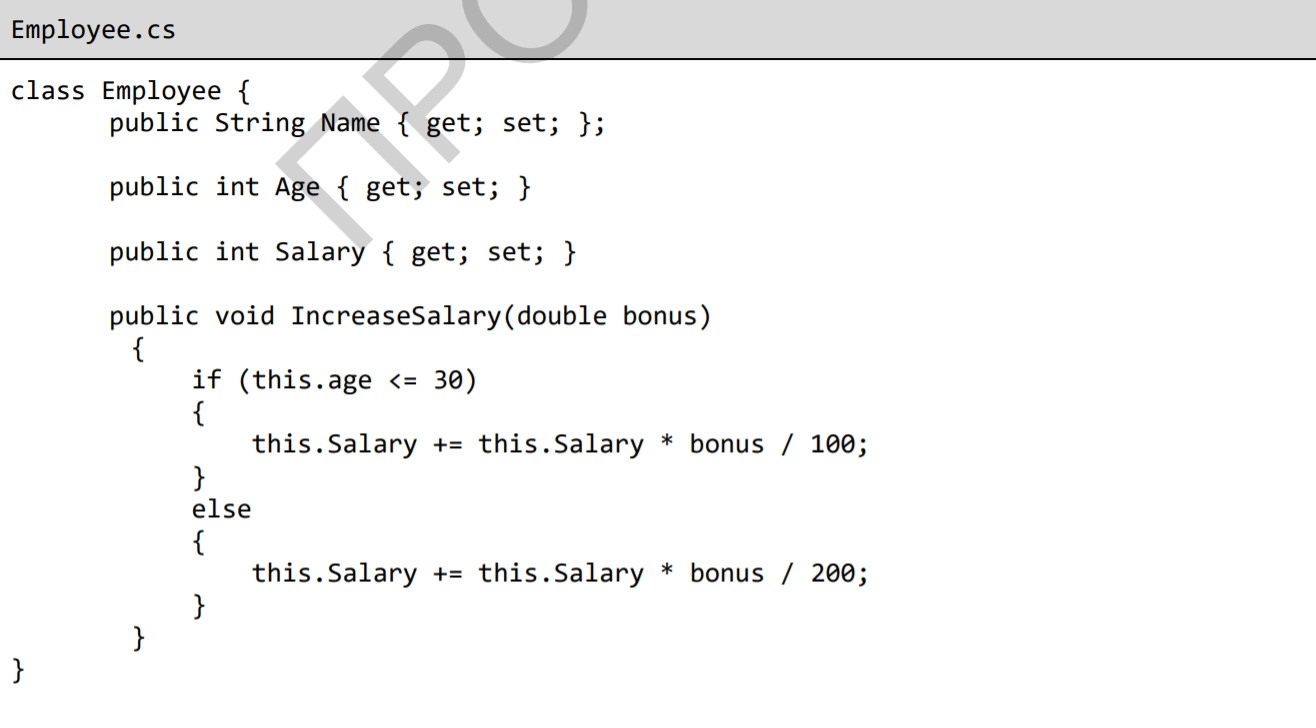
Реализирайте класа Employee по следната диаграма:



Примери:



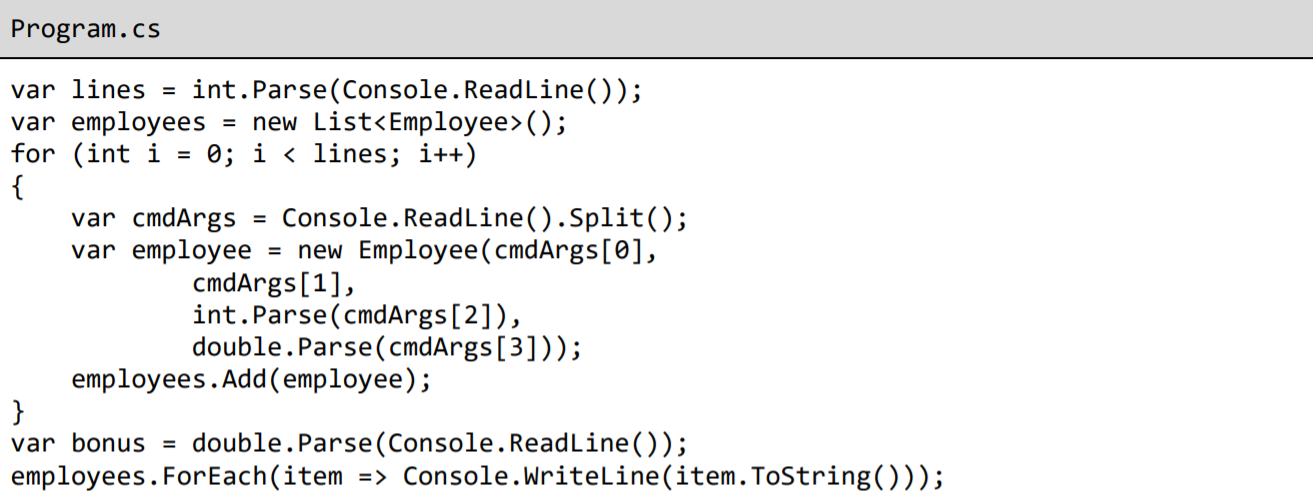
Фрагмент:



Примерен работещ фрагмент (Любомир Василев):

|  |
| --- |
| Employee.cs |
| public class Employee  {  private string name;  private int age;  private double salary;  public Employee(string firstName, string LastName, int age, double salary)  {  this.name = $"{firstName} {LastName}";  this.age = age;  this.salary = salary;  }  public string Name  {  get { return this.name; }  }  public int Age  {  get { return this.age; }  }  public double Salary  {  get { return this.salary; }  }  public void IncreaseSalary(double bonus)  {  if (this.age < 30) //Not "<=" but "<"  {  this.salary += this.salary \* bonus / 200; //Half of the bonus  }  else  {  this.salary += this.salary \* bonus / 100;  }  }  public override string ToString()  {  return $"{Name} get {Salary:f2} leva";  }  } |

Класът Employee трябва да работи със следния програмен фрагмент:



Примерен работещ фрагмент (Любомир Василев) (има грешки в горния):

|  |
| --- |
| Program.cs |
| var lines = int.Parse(Console.ReadLine());  var employees = new List<Employee>();  for (int i = 0; i < lines; i++)  {  var cmdArgs = Console.ReadLine().Split();  var employee = new Employee(cmdArgs[0],  cmdArgs[1],  int.Parse(cmdArgs[2]),  double.Parse(cmdArgs[3]));  employees.Add(employee);  }  var bonus = double.Parse(Console.ReadLine());  employees.ForEach(item => item.IncreaseSalary(bonus)); //Before printing - salary must be increased!  employees.ForEach(item => Console.WriteLine(item.ToString())); //Print result |

