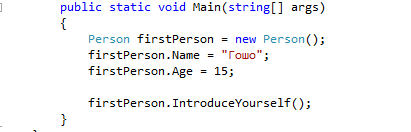
# Упражнения: Елементи на класа

## Kлас Човек

Дефинирайте клас **Person** с **public** свойства за **името** и **възрастта** и **private** полета, които да съхраняват информацията. Класът трябва да има:

* name: String - поле
* age: int - поле
* **Name:** **String - свойство**
* **Age:** **int - свойство**
* **IntroduceYourself() - метод**

Използвайте класа в Main по следния начин:  


### Бонус\*

Опитайте да създадете няколко обекта от тип Person и да отпечатате данните им:

|  |  |
| --- | --- |
| **Име** | **Възраст** |
| Pesho | 20 |
| Gosho | 18 |
| Stamat | 43 |

## Клас Банкова сметка

Създайте клас BankAccount (или използвайте вече създадения клас)

Класът трябва да има **private** полета за:

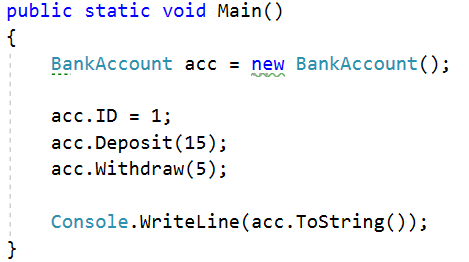
* id: int
* balance: double

Класът трябва да има и следните **public** свойства и методи:

* ID: int
* Balance: double
* Deposit(Double amount): void
* Withdraw(Double amount): void

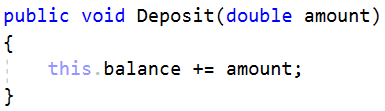
Предефинирайте и метода ToString().

Трябва да можете да използвате класа по този начин:

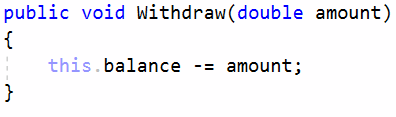


### Решение

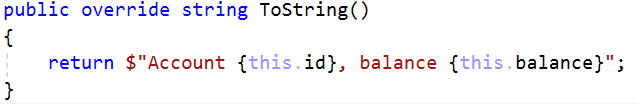
Създайте метод Deposit(double amount)



Създайте метод Withdraw(double amount)



Предефинирайте метода toString()



## Човекът и неговите пари

Създайте клас **Person**.

Той трябва да има полета за:

* Name: **string**
* Age: **int**
* Accounts: **List<BankAccount>**

Класът трябва да има метод, който изчислява всички пари, които притежава човека от сметките си:

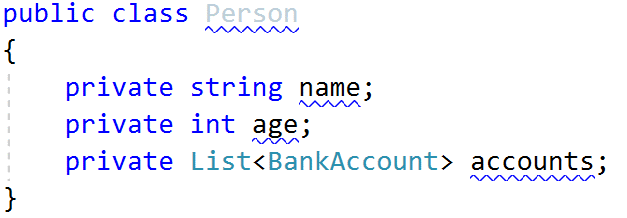
* **GetBalance():** **double**

Класът трябва да има и следните конструктори:

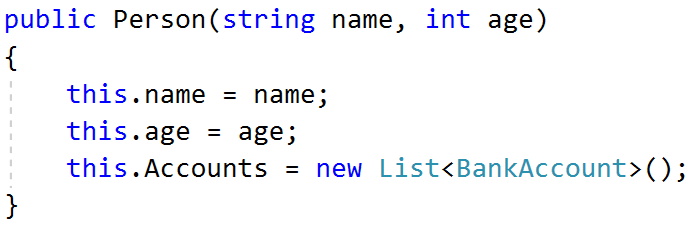
* **Person(string name, int age)**
* **Person(string name, int age, List<BankAccount> accounts)**

**Решение**

Създайте класа както обикновено:



Създайте конструктор с два параметъра:



Създайте конструктор с три параметъра:



Създайте метод **GetBalance()**



**По желание:** Можете да се възползвате от верижното извикване на конструктори:



1. **Конструктори за класа Човек**

Добавете 2 конструктора към класа **Person** от миналата задача и с помощта на верижно извикване на кода използвайте повторно съществуващ вече програмен код:

1. Първият конструктор трябва да е без параметри и да създава човек с име “**No name**” и възраст = **1**.
2. Вторият конструктор трябва да приема само един целочислен параметър за възрастта и да създава човек с име “**No name**” и възраст равна на подадения параметър.

В класа трябва да присъства и конструктор, който приема низ за името и цяло число за възрастта и да създава личност с указаното име и възраст

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Pesho  20 | No name 1  No name 20  Pesho 20 |
| Gosho  18 | No name 1  No name 18  Gosho 18 |
| Stamat  43 | No name 1  No name 43  Stamat 43 |

## Най-стария член на фамилията

Създайте клас **Person** с полета **name** и **age**. Създайте клас **Family**. Този клас трябва да има **списък от хора**, метод за добавяне на членове (**void AddMember(Person member)**) и метод, връщащ най-стария член на фамилията(**Person GetOldestMember())**. Напишете програма, която прочита името и възрастта на **N** души и **ги добавя към фамилията**. После **отпечатва името** и **възрастта** на най-стария ѝ член.

### Примери

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Вход** | **Изход** |  | **Вход** | **Изход** |
| 3  Pesho 3  Gosho 4  Annie 5 | Annie 5 |  | 5  Steve 10  Christopher 15  Annie 4  Ivan 35  Maria 34 | Ivan 35 |

## Статистическо проучване

С помощта на класа **Person** и класа **People** (съдържащ private списък от обекти от тип Person) напишете програма, която прочита от конзолата **N** реда с лична информация за хора и после извежда имената на всички, които са на **възраст над 30** години, **сортирани в азбучен ред**.

### Бележки

Добавете методи в класа People за добавянето, сортирането и извеждането на хората.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 3  Pesho 12  Stamat 31  Ivan 48 | Ivan - 48  Stamat - 31 |
| 5  Nikolai 33  Yordan 88  Tosho 22  Lyubo 44  Stanislav 11 | Lyubo - 44  Nikolai - 33  Yordan - 88 |

## \*Списък на служители

Дефинирайте клас **Employee**, съдържащ информация за **име, заплата, длъжност, отдел, ел.поща** и **възраст.** Полетата **име, заплата, длъжност** и **отдел** са **задължителни**, останалите са **опционални**.

Вашата задача е да напишете програма, която прочита **N** реда с информация за служители от конзолата, намира кой е отдела с най-висока средна заплата и за всеки служител от този отдел отпечатва неговото **име, заплата, ел.поща и възраст**. Служителите трябва да са **сортирани според заплатите им, в намаляващ ред**. Ако някой служител **няма ел.поща**, на нейно място трябва да се отпечати “**n/a**”, а ако няма указана **възраст**, да се изведе “**-1**” вместо това. **Заплатата** трябва да бъде отпечатана с **две цифри** след десетичния знак.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 4  Pesho 120.00 Dev Development pesho@abv.bg 28  Toncho 333.33 Manager Marketing 33  Ivan 840.20 ProjectLeader Development ivan@ivan.com  Gosho 0.20 Freeloader Nowhere 18 | Highest Average Salary: Development  Ivan 840.20 ivan@ivan.com -1  Pesho 120.00 pesho@abv.bg 28 |
| 6  Stanimir 496.37 Temp Coding stancho@yahoo.com  Yovcho 610.13 Manager Sales  Toshko 609.99 Manager Sales toshko@abv.bg 44  Venci 0.02 Director BeerDrinking beer@beer.br 23  Andrei 700.00 Director Coding  Popeye 13.3333 Sailor SpinachGroup popeye@pop.ey | Highest Average Salary: Sales  Yovcho 610.13 n/a -1  Toshko 609.99 toshko@abv.bg 44 |

## \* Разликата в дни между две дати

Създайте клас **DateModifier,** който пресмята разликата в дни между две дати. Той трябва да съдържа метод, приемащ **два низови параметъра, указващи дати** в текстов формат и **изчислява** разликата в дни между тях.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 1992 05 31  2016 06 17 | 8783 |
| 2016 05 31  2016 04 19 | 42 |

## \* Сурови данни

Вие сте собственик на куриерска компания и искате да направите система за проследяване на вашите коли и техния товар. Дефинирайте клас **Car** с информация за **модела, двигателя, товара** и **колекция от точно 4 гуми**. Моделът, товарът и гумите трябва да са **отделни класове**; създайте конструктор, който получава пълната информация за колата и създава и инициализира нейните вътрешни компоненти (двигател, товар и гуми).

На първия ред от входната информация ще получите число **N** - броя на колите, които имате, а на всеки от следващите **N** реда ще има информация за кола във формата “**<Модел> <СкоростНаДвигателя> <МощностнаДвигателя> <ТеглоНаТовара> <ТипНаТовара> <Гума1Налягане> <Гума1Възраст> <Гума2Налягане> <Гума2Възраст> <Гума3Налягане> <Гума3Възраст> <Гума4Налягане> <Гума4Възраст>”** където скорост, мощност, тегло на товара и възраст на гумите са **цели числа**, а налягането е дробно число, **с двойна точност.**

След тези **N** реда ще получите един-единствен ред с една от следните две команди: “**fragile**” или “**flamable**”. Ако командата е “**fragile**”, то отпечатайте всички коли с **тип на товара “fragile”** с гуми с **налягане** **< 1**; ако командата е “**flamable**”, отпечатайте всички коли с **тип на товара “flamable”** и **мощност на двигателя > 250**. Колите трябва да се изведат в реда, в който са подадени като входни данни.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 2  ChevroletAstro 200 180 1000 fragile 1.3 1 1.5 2 1.4 2 1.7 4  Citroen2CV 190 165 1200 fragile 0.9 3 0.85 2 0.95 2 1.1 1  fragile | Citroen2CV |
| 4  ChevroletExpress 215 255 1200 flamable 2.5 1 2.4 2 2.7 1 2.8 1  ChevroletAstro 210 230 1000 flamable 2 1 1.9 2 1.7 3 2.1 1  DaciaDokker 230 275 1400 flamable 2.2 1 2.3 1 2.4 1 2 1  Citroen2CV 190 165 1200 fragile 0.8 3 0.85 2 0.7 5 0.95 2  flamable | ChevroletExpress  DaciaDokker |

## \* Тестов Клиент

Създайте тестов клиент, който използва класа **BankAccount,** направен в задача 2.

Трябва да поддържате следните операции, подавани като входни данни от конзолата:

* **Create {Id}**
* **Deposit {Id} {Amount}**
* **Withdraw {Id} {Amount}**
* **Print {Id}**
* **End**

Създайте методи към Program.cs за всяка от командите. Имайте в предвид и следната допълнителна обработка на данните:

* Ако се опитате да създадете сметка със съществуващо Id, изведете **"Account already exists".**
* Ако се опитате да извършите операция върху несъществуваща сметка, изведете **"****Account does not exist"**.
* Ако се опитате да изтеглите сума, която е по-голяма от баланса, изведете **"****Insufficient balance"**.
* Print командата, трябва да изведе **"Account ID{id}, balance {balance}"**. Закръглете баланса до втория знак след запетаята.

**Примери**

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Create 1  Create 2  Deposit 1 20  Withdraw 1 30  Withdraw 1 10  Print 1  End | Account already exists  Insufficient balance  Account ID1, balance 10.00 |
| Create 1  Deposit 2 20  Withdraw 2 30  Print 2  End | Account does not exist  Account does not exist  Account does not exist |

**Решение**

Използвайте **Dictionary<int, BankAccount>** за да пазите сметките

Направете си цикъла за приемане на команда



Създайте методи към Program.cs, за всяка от командите.

* Create – проверявате дали в речника има ключ с такова id – ако няма, създавате сметката.

Имплементирайте останалите команди работейки с подобна логика.