Упражнения: Задачи за подготовка за входно ниво

Problem 1. Дефиниране на клас Person

Дефинирайте клас **Person** с **public** полета **name** (String) и **age** (int).

Бонус*

Опитайте се да създадете няколко обекта от Person:

Име	Възраст
Pesho	20
Gosho	18
Stamat	43

Problem 2. Семейство

Създайте клас **Person** с полета **name** (String) и **age** (int). Създайте клас **Family**. В този клас трябва да има списък от хора. Напишете програма, която въвежда информация за N човека от едно семейство, след което изведете семейството по азбучен ред.

Примери

Вход	Изход
3 Pesho 3 Gosho 4 Annie 5	Annie 5 Gosho 4 Pesho 3

Вход	Изход
5 Steve 10 Christopher 15 Annie 4 Ivan 35 Maria 34	Annie 4 Christopher 15 Ivan 35 Maria 34 Steve 10

Бонус*

Опитайте се да създадете метод Print за класа Family

Problem 3. Статистика

Използвайки класът Person, напишете програма, която въвежда от конзолата **N** реда информация за хора и изпечатва хората на възраст **по-голяма от 30** години, **сортирани по азбучен ред**.

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

Problem. 4. В най-тъмните подземия (Dungeonest Dark)

Като млад авантюрист търсиш злато и слава в най-тъмните подземия.

Имате първоначално **здраве 100** и първоначални **монети 0**. Ще ви бъде даден низ, представляващ стаите в подземието, където сте изпратен на мисия. Информацията за всяка стая е разделена от останалите с '|' (вертикална черта): "стая1 | стая2 | стая3 ..."

Всяка стая съдържа име на намерен предмет или чудовище и цяло число, разделени с интервал.

- Ако първата част е **"potion",** то сте попаднали на отвара, която ви лекува. Увеличете здравето на героя ви с числото във втората част. Но вашето здраве не може да надвишава първоначалното (100). Освен това, ако дадена отвара ви дава възможност да се излекувате над 100, то вие не може да се възползвате напълно от цялата ѝ сила, а само от тази нейна част, която довежда здравето ви до 100. Отпечатайте: "**You healed for {0} hp**.", където {0} е частта от отварата, от която сте се възползвали. След това, отпечатайте текущото си здраве: **"Current health: {0} hp."**.
- Ако първата част е **"chest",** то вие сте намерили сандък с монети, колкото е числото на втора позиция. Отпечатайте **"You found {0} coins."** и ги прибавете към вашите.
- **Във всеки друг случай** сте изправени пред чудовище, ще трябва да се биете. Втората част на информацията на стаята съдържа атаката на чудовището. Трябва да извадите силата на атаката на чудовището от вашето здраве. После:

- Ако още сте жив (здраве > 0), то вие сте убили чудовището и трябва да се изведе на конзолата "**You slayed {monster}.**"
- Ако сте умрели, изведете "**You died! Killed by {monster}.**" и най-далечната стая до която сте успели да достигнете: "**Best room: {room}**". С това вашата мисия приключва.

Ако сте успели да преминете през всички стаи в подземието, изведете на конзолата следващите три реда:

"You've made it!", **"Coins: {coins}"**, **"Health: {health}"**.

Вход

Получавате низ, представляващ стаите в подземието, разделение с **'|'**:

"room1|room2|room3…".

Изход

Отпечатайте съответните съобщения, описани по-горе.

Примери

Вход	Изход
rat 10|bat 20|potion 10|rat 10|chest 100|boss 70|chest 1000	You slayed rat. You slayed bat. You healed for 10 hp. Current health: 80 hp. You slayed rat. You found 100 coins. You died! Killed by boss. Best room: 6
Вход	**Изход**
cat 10|potion 30|orc 10|chest 10|snake 25|chest 110	You slayed cat. You healed for 10 hp. Current health: 100 hp. You slayed orc. You found 10 coins. You slayed snake.

	You found 110 coins. You've made it! Coins: 120 Health: 65

... игра, в която всеки герой печели деня с блестяща броня и усмивка ...

Problem 5. 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

Problem 6. Тестов Клиент

Създайте тестов клиент, който използва класа **BankAccount**

Класът трябва да има private полета за:

- id: int
- balance: double

Класът трябва да има и следните 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:

```
public static void Main()
{
    BankAccount acc = new BankAccount();

    acc.ID = 1;
    acc.Deposit(15);
    acc.Withdraw(5);

    Console.WriteLine(acc.ToString());
}
```

Решение

Създайте метод **Deposit(double amount)**

```
public void Deposit(double amount)
{
    this.balance += amount;
}
```

Създайте метод **Withdraw(double amount)**

```
public void Withdraw(double amount)
{
    this.balance -= amount;
}
```

Предефинирайте метода **toString()**

```
public override string ToString()
{
    return $"Account {this.id}, balance {this.balance}";
}
```

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

```
var cmdArgs = command.Split();

var cmdType = cmdArgs[0];
switch (cmdType)
{
    case "Create":
        Create(cmdArgs, accounts);
        break;
    case "Deposit":
        Deposit(cmdArgs, accounts);
        break;
    case "Withdraw":
        Withdraw(cmdArgs, accounts);
        break;
    case "Print":
        Print(cmdArgs, accounts);
        break;
}
```

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

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

```
private static void Create(string[] cmdArgs, Dictionary<int, BankAccount> accounts)
{
    var id = int.Parse(cmdArgs[1]);
    if (accounts.ContainsKey(id))
    {
        Console.WriteLine("Account already exists");
    }
    else
    {
        var acc = new BankAccount();
        acc.ID = id;
        accounts.Add(id, acc);
    }
}
```

Имплементирайте останалите команди работейки с подобна логика.

Problem 7. Проверка на данните

Разширяваме класа Person с подходяща валидация за всяко поле:

- Имената трябва да са поне 3 символа
- възрастта не трябва да е нула или отрицателно число
- заплатата не може да бъде по-малка от 460.0

Print proper message to end user (look at example for messages).

Use ArgumentExeption with messages from example.

Изведете подходящо съобщение за последен потребител (виж примера за съобщения).

Използвайте ArgumentExeption със съобщения от примера.

Примери

Вход	Изход
5 Asen Ivanov -6 2200 B Borisov 57 3333 Ventsislav Ivanov 27 600 Asen H 44 666.66 Boiko Angelov 35 300 20	Age cannot be zero or negative integer First name cannot be less than 3 symbols Last name cannot be less than 3 symbols Salary cannot be less than 460 leva Ventsislav Ivanov get 660.0 leva

Решение

Добавете проверка към всички setters на Person. Валидацията може да изглежда като това или нещо подобно:

```
public double Salary
{
    get { return this.salary; }
    set
    {
        if (value < 460)
        {
            throw new ArgumentException("Salary cannot be less than 460 leva");
        }

        this.salary = value;
    }
}
```

Problem 8. На пазар

Създайте два класа: клас Person и клас Product. Всеки човек трябва да има име, пари и една торба с продукти. Всеки продукт трябва да има име и стойност. Името не може да бъде празен низ. Парите не може да бъдат отрицателно число.

Създайте програма, в която всяка команда отговаря на закупуване на продукт от един обект Person (Човек). Ако човек може да си позволи продукт го добавя към чантата си. Ако човек не разполага с достатъчно пари, изведете подходящо съобщение (("[Person name] can't afford [Product name]").

На първите два реда са дадени всички хора и всички продукти. След всички покупки да се изведат за всеки човек по реда на въвеждане всички продукти, които той е купил, също в реда на въвеждане на покупките. Ако нищо не е купил, да се изведе името на човека, последвано от "**Nothing bought**".

При въвеждане на невалидни (отрицателна сума пари да се създаде изключение със съобщение: "**Money cannot be negative**") или празно име (празно име с изключение със съобщение : "**Name cannot be empty**") за край на програмата с подходящо съобщение. Вижте примерите по-долу:

Примери

Вход	Изход
Pesho=11;Gosho=4 Bread=10;Milk=2; Pesho Bread Gosho Milk Gosho Milk Pesho Milk END	Pesho bought Bread Gosho bought Milk Gosho bought Milk Pesho can't afford Milk Pesho - Bread Gosho - Milk, Milk
Mimi=0 Kafence=2 Mimi Kafence END	Mimi can't afford Kafence Mimi – Nothing bought
Jeko=-3 Chushki=1; Jeko Chushki END	Money cannot be negative

Problem 9. Пътувания с коли

Задачата ви е да напишете програма, която пази информация за автомобили, за това колко гориво имат и поддържа методи за движение на колите. Дефинирайте клас **Car** с информация за **модела, количеството гориво, разхода на гориво за 1 км.** и **пропътуваното разстояние**. Моделът на автомобилите е **уникален** - няма да има две коли с един и същи модел.

На първия ред на входните данни ще получите число **N** – броят на автомобилите, които ще следите. На всеки от следващите **N** реда ще има информация за по една кола в следния формат "<**Модел> <КоличествоГориво> <РазходНаГоривоЗа1км>**". Всички **коли започват с пропътувани 0 км**.

След тези **N** реда, до достигане на команда "**End**", ще получавате команди във следния формат "**Drive <МоделКола> <бройКм>**". Реализирайте в класа **Car** метод, изчисляващ дали колата може да измине това разстояние или не. Ако да, **горивото на колата** трябва да бъде **намалено** с количеството на горивото, използвано за пътуването, а **изминатите от нея километри** трябва да бъдат **увеличени** с пропътуваните километри. Ако няма да може да го пропътува, колата не трябва да се движи (т.е. количеството на горивото и пропътуваните от нея километри трябва да останат същите), а на конзолата да се отпечата "**Insufficient fuel for the drive**". След достигане на команда "**End**" трябва да се отпечата **всяка кола** и нейното **текущо количество гориво,** както и **пропътуваните километри във формата** ""**Модел> <КоличествоГориво> <пропътуваниКм>**", където количеството гориво трябва да е отпечатано с **две цифри** след десетичния знак.

Примери

Вход	Изход
2 AudiA4 23 0.3 BMW-M2 45 0.42 Drive BMW-M2 56 Drive AudiA4 5 Drive AudiA4 13 End	AudiA4 17.60 18 BMW-M2 21.48 56
3 AudiA4 18 0.34 BMW-M2 33 0.41 Ferrari-488Spider 50 0.47 Drive Ferrari-488Spider 97 Drive Ferrari-488Spider 35 Drive AudiA4 85 Drive AudiA4 50 End	Insufficient fuel for the drive Insufficient fuel for the drive AudiA4 1.00 50 BMW-M2 33.00 0 Ferrari-488Spider 4.41 97

Problem 10. Банкер

Създайте класа **BankAccount.**

Този клас трябва да има полета за:

- id: int
- balance: double

Класът трябва да има свойства за:

- **ID: int**
- **Balance: double**

Създайте методите:

- **Deposit(Double amount): void – който да вкарва пари в сметката**
- **Withdraw(Double amount): void – който да изтегля пари от сметката**

Заменете метода **ToString(),** като в този метод изпечатвайте информация за банковата сметка

Създайте статичния клас **Bank**.

В този клас трябва да създадете следните функционалности:

- **Теглене на средства** – на този метод трябва да подадете ID-то, списъка с всички създадени сметки и желаната сума за теглене. Ако сумата я няма в наличност или сметката не съществува, изведете подходящо съобщение – за тази функционалност може да се наложи да реализирате няколко метода.
- **Внасяне на средства** – на този метод трябва да подадете ID-то, списъка с всички създадени сметки и желаната сума за внасяне. Ако сметката не съществува, изведете

подходящо съобщение – за тази функционалност може да се наложи да реализирате няколко метода или да използвате вече реализирани такива от предходната точка

Изберете адекватна структура и логика за реализирането на желаните функционалности