Упражнения: Енкапсулация

Можете да тествате решенията си в **Judge системата**: https://judge.softuni.bg/Contests/3163/Encapsulation

1. Сортиране на хора по име и възраст

Важно: Трябва да имате публичен клас **StartUp** в namespace **PersonsInfo**.

Създайте клас **Person**, който трябва да има **публични** свойства и **частни** setter-и за:

- **FirstName**: стринг (**string**)
- **LastName**: стринг (**string**)
- **Age**: цяло число (**int**)
- **ToString()**: стринг (**string**) - **override**

Трябва да можете да ползвате класа по следния начин:

```
static void Main(string[] args)
{
    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()));

}
```

Примери

Вход	Изход
5 Seth Nelson 65 Liam Scott 57 Brian Clark 27 Alisa Bell 44 Sophie Baker 35	Alisa Bell is 44 years old. Seth Nelson is 65 years old. Sophie Baker is 35 years old. Liam Scott is 57 years old. Brian Clark is 27 years old.

Решение

Създайте **нов class** и му задайте **коректно име**. Дефинирайте **публичните** свойства:

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

    public int Age
    {
        get { return age; }
        set { age = value; }
    }
    public string FirstName
    {
        get { return firstName; }
        set { firstName = value; }
    }
    public string LastName
    {
        get { return lastName; }
        set { lastName = value; }
    }
```

Създайте конструктор за **Person**, който приема 3 параметъра - **`firstName`**, **`lastName`** и **`age`:**

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

Презапишете метода **`ToString()`**:

```
public override string ToString()
{
    return $"{this.FirstName} {this.LastName} is {this.Age} years old.";
}
```

2. Повишение на заплатата

Важно: Трябва да имате клас **`StartUp`** в namespace **`PersonsInfo`**.

Създайте обекти от класа **Person**. Прочетете техните **name, age** и **salary** от конзолата. Прочетете процента на бонус заплатата на всеки човек. Тези, които са по-млади от 30 години, получават **половината от повишението**. Разширете класа **Person** от предишната задача.

Нови **свойства** и **методи:**

- `Salary: decimal`
- `IncreaseSalary(decimal percentage)`

Трябва да можете да използвате класа по следния начин:

```
static void Main(string[] args)
{
    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]),
                                decimal.Parse(cmdArgs[3]));

        persons.Add(person);
    }
    var parcentage = decimal.Parse(Console.ReadLine());
    persons.ForEach(p => p.IncreaseSalary(parcentage));
    persons.ForEach(p => Console.WriteLine(p.ToString()));

}
```

Примери

Вход	Изход
5 Nick Adams 65 2200 Lynda Fisher 57 3333 Paul Walker 27 600 Vera Nelson 44 666.66 Connor Perry 35 559.4 20	Nick Adams receives 2640.00 leva. Lynda Fisher receives 3999.60 leva. Paul Walker receives 660.00 leva. Vera Nelson receives 799.99 leva. Connor Perry receives 671.28 leva.

Решение

Добавете ново **публично** свойство за **заплата** и рефакторирайте конструктора. Добавете нов **метод**, който да **обновява заплатата** с даден бонус:

```
public void IncreaseSalary(decimal percentage)
{
    if (this.Age > 30)
    {
        this.Salary += this.Salary * percentage / 100;
    }
    else
    {
        this.Salary += this.Salary * percentage / 200;
    }
}
```

Рефакторирайте метода **ToString()**

3. Валидация на данни

Важно: Трябва да имате публичен клас **StartUp** в namespace **PersonsInfo**.

Разширете класа **Person** с **валидация** за всяко **поле**:

- **First** и **Last name** трябва да са с по **най-малко 3 символа**
- **Age не трябва** да бъде **нула** или **отрицателно число**
- **Заплатата не трябва** да бъде **по-малко от 460 (decimal)**

При **невалидни данни**, отпечатайте съответното съобщение:

- **"Age cannot be zero or a negative integer!"**
- **"First name cannot contain fewer than 3 symbols!"**
- **"Last name cannot contain fewer than 3 symbols!"**
- **"Salary cannot be less than 460 leva!"**

Използвайте **ArgumentExeption** за съобщенията.

Примери

Вход	Изход
5 Miles Parks -6 2200 B Potter 57 3333 Julie Brown 27 600 Alice H 44 666.66 Joey Hall 35 300 20	Age cannot be zero or a negative integer! First name cannot contain fewer than 3 symbols! Last name cannot contain fewer than 3 symbols! Salary cannot be less than 460 leva! Julie Brown gets 660.00 leva.

Решение

Добавете валидация към всички **setter**-и в класа **Person**. Валидацията може да изглежда по следния начин:

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

4. Отбор

Важно: Трябва да имате публичен клас **`StartUp`** в namespace **`PersonsInfo`**.

Създайте клас **Team**.

Класът трябва да има:

- **Частни полета** за:
  - **`name: string`**
  - **`firstTeam: List<Person>`**
  - **`reserveTeam: List<Person>`**
- **Конструктор**
  - **`Team(string name)`**
- **Публични свойства** за:
  - **`FirstTeam: List<Person> (read only!)`**
  - **`ReserveTeam: List<Person> (read only!)`**
- **Метод** за **добавяне** на **players:**
  - **`AddPlayer(Person person): void`**

Добавете към отбора всички хора, които получавате. Тези, които са **по-млади от 40 години** отиват в **първи екип (first team)**, а останалите – в **резервния екип (reserve team)**. Накрая отпечатайте **размера** на **първия** и на **резервния екип.**

Трябва да можете да ползвате класа по следния начин:

```
Team team = new Team("SoftUni");
foreach (var person in persons)
{
    team.AddPlayer(person);
}
```

Не трябва да можете да ползвате класа по следния начин:

StartUp.cs

```
Team team = new Team("SoftUni");

foreach (Person person in persons)
{
    if(person.Age < 40)
    {
        team.FirstTeam.Add(person);
    }
    else
    {
        team.ReserveTeam(person);
    }
}
```

Примери

Вход	Изход
5 Troy Jones 20 2200 Martin Francis 57 3333 Ted Adams 27 600 Alisa Gomez 25 666.66 Lucia Cox 35 555	First team has 4 players. Reserve team has 1 players.

Решение

Добавете нов **клас Team**. **Полетата** и **конструкторът** трябва да изглеждат така:

```
private string name;
private List<Person> firstTeam;
private List<Person> reserveTeam;

1 reference
public Team(string name)
{
    this.name = name;
    this.firstTeam = new List<Person>();
    this.reserveTeam = new List<Person>();
}
```

Свойствата за **`FirstTeam`** и **`ReserveTeam`** трябва да имат **само getter-и.**

```
public IReadOnlyCollection<Person> FirstTeam
{
    get { return this.firstTeam.AsReadOnly(); }
}
1 reference
public IReadOnlyCollection<Person> ReserveTeam
{
    get { return this.reserveTeam.AsReadOnly(); }
}
```

Трябва да има **само един метод**, който **добавя играчи** към екипите:

```
public void AddPlayer(Person person)
{
    if (person.Age < 40)
    {
        this.firstTeam.Add(person);
    }
    else
    {
        this.reserveTeam.Add(person);
    }
}
```

5. Клас Box

Дадена ви е кутия с параметри **`length`**, **`width`** и **`height`**. Създайте клас **Box**, който може да бъде инстанциран със същите **три параметъра**. Направете достъпни за външния свят **само методите** за **лице на околната и пълната повърхнина** и за **обем.** (формули: https://www.matematika.bg/geometry/volume.html).

Страната на кутията **не трябва** да бъде **нула** или **отрицателно число.** Добавете **валидация на данните** за **всеки параметър**, подаден на конструктора. Направете **частен setter**, който осъществява **вътрешна валидация на данните.**

На **първите три реда** ще получите **`length`** (дължина), **`width`** (ширина) и **`height`** (височина). На **следващите три реда** отпечатайте **пълната, околната повърхнина** и **обема** на кутията.

Примери

Вход	Изход
2 3 4	Surface Area - 52.00 Lateral Surface Area - 40.00 Volume - 24.00

1.3 1 6	Surface Area - 30.20 Lateral Surface Area - 27.60 Volume - 7.80
2 -3 4	Width cannot be zero or negative.

Насоки:

```
public Box(double length, double width, double height)
{
    this.Length = length;
    this.Width = width;
    this.Height = height;
}
```

```
public double Length
{
    get { return this.length; }
    private set
    {
        if (value <= 0)
        {
            throw new Exception("Length cannot be zero or negative. ");
        }
        else
        {
            this.length = value;
        }
    }
}
```

6. Животинска ферма

За тази задача трябва да **свалите** дадения в ресурсите **скелет**.

Вече сте запознати с **енкапсулацията**. За тази задача ще работите по проекта **`AnimalFarm`** (животинска ферма). Тя съдържа клас **`Chicken`**. **`Chicken`** съдържа няколко **полета**, **конструктор**, няколко **свойства** и **методи**. Вашата задача е да **енкапсулирате** или **скриете** всичко, което **не е предназначено** за **достъп** или **промяна извън** класа.

Стъпка 1. Енкапсулирайте полетата

Оставянето на полета за модификация извън класа може да бъде **опасно**. Направете **всички полета** в класа **`Chicken` частни** (private). В случай че се нуждаете от стойността на някое поле на друго място, можете да използвате **getter**-и, за да я вземете.

Стъпка 2. Уверете се, че класа има правилен state (състояние)

Създаването на **getter-и** и **setter-и** губи смисъла си, ако те не се използват. Конструктора на **`Chicken` модифицира полетата директно**, което е **грешно**, когато има подходящи **setter-и** за тази цел. **Променете** конструктора, за да коригирате тази грешка.

Стъпка 3. Валидирайте данните

Валидирайте **name (името)** на всеки chicken (не трябва да бъде **null** или **празен стринг**). В случай на **невалидно име**, отпечатайте Exception съобщение **`"Name cannot be empty."`** .

Валидирайте **age (възрастта)** - **минималната възраст е 0,** а **максималната – 15**. В случай на **невалидна възраст**, отпечатайте Exception съобщение: **`"Age should be between 0 and 15."`**.

Стъпка 4. Скрийте вътрешната логика

Ако **даден метод** е предназначен само за употреба от неговите класове-наследници (**descendants**), няма смисъл той да е **публичен**. Методът **`CalculateProductPerDay()`** се използва от публичния getter **`ProductPerDay`.** Това означава, че методът може спокойно да бъде **скрит** в класа **`Chicken`** и деклариран като **частен**.

Стъпка 5. Тествайте кода си в Judge

Предайте кода си като **zip файл** в Judge. Направете zip с всички файлове **без** папките **bin** и **obj** и качете създадения zip в Judge.

Примери

Вход	Изход
`Lucia` `10`	`Chicken Lucia (age 10) can produce 1 eggs per day.`
`Lucia` `17`	`Age should be between 0 and 15.`

7. Пазаруване

Създайте два класа: **class `Person`** и **class `Product`**. Всеки човек (person) трябва да има **`name (име)`**, **`money (пари)`** и **`bag of products (торба с продукти)`**. Всеки продукт има **`name (име)`** и **`cost (цена)`**. Името **не трябва** да бъде **празен стринг**. Парите **не трябва** да бъдат **отрицателно число**.

Създайте програма, в която **всяка команда** представлява **човек (person)**, който купува **продукт (product)**. Ако човекът **може да купи** продукта, **добавете** продукта към торбата. Ако човекът **няма достатъчно пари**, отпечатайте следното съобщение: ("**`{personName} can't afford {productName}`**").

На **първите два реда** ще получите данни за **всички хора** и **всички продукти**. След приключване на всички покупки отпечатайте **всеки човек (person)** и **всички продукти (product)**, които съответният човек е купил, в реда, в който сте ги получили. Ако даден човек **не е купил нищо**, отпечатайте името му, последвано от "**`Nothing bought`**".

В случай на **невалиден вход**, прекратете програмата и отпечатайте съответното съобщение ("**`Money cannot be negative`**" или "**`Name cannot be empty`**").

Примери

Вход	Изход
Mark=11;Lesley=4 Bread=10;Milk=2 Mark Bread Lesley Milk Lesley Milk Mark Milk END	Mark bought Bread Lesley bought Milk Lesley bought Milk Mark can't afford Milk Mark - Bread Lesley - Milk, Milk
Philip=0 Coffee=2 Philip Coffee END	Philip can't afford Coffee Philip - Nothing bought
Sandy=-3 Pepper=1 Sandy Pepper END	Money cannot be negative

8. Калории в пица

Една пица е направена от тесто и различен топинг. Създайте **клас `Pizza`**, който има **`name (име)`**, **`dough (тесто)`** и **`toppings (топинг)`** като полета. **Всяка съставка** има свой **собствен клас** с различни свойства: **тестото** може да е бяло (white) или пълнозърнесто (wholegrain), освен това може да бъде хрупкаво (crispy), гъвкаво (chewy) или домашно (homemade). **Топингът** може да бъде от тип месо (meat), зеленчуци (veggies), сирене (cheese) или сос (sauce).

Всяка съставка трябва да има **weight (тегло)** в грамове и метод да се **изчисляват** калориите според типа. Калориите за грам се изчисляват чрез **модификатори**. Всяка съставка има **2** калории за грам като **база** и **модификатор**, който **увеличава броя на калориите по даден множител.** Например бялото (white) тесто има модификатор от **1.5**, гъвкавото (chewy) тесто има модификатор от **1.1**, което означава, че **бяло гъвкаво тесто** с тегло **100** грама ще има **2 * 100 * 1.5 * 1.1 = 330 калории общо.**

Вашата задача е да създадете класовете по начин, по който са **енкапсулирани** и да добавите **публичен** метод за всяка пица, който **изчислява калориите според съставките, които има пицата.**

Стъпка 1. Създайте клас Dough

Основната съставка на една **`Pizza`** е тестото (dough). Първо, създайте съответния клас. Той има тип на тестото **(flour type)**, който може да е **white (бял)**, или **wholegrain (пълнозърнест)**. В допълнение, има и техника на изпичане **(baking technique)**, която може да бъде **crispy (хрупкава)**, **chewy (гъвкава)** или **homemade (домашна)**. Тестото трябва да има и **weight (тегло)** в грамове. Калориите за грам от тестото се изчисляват **в зависимост** от **типа** и **техниката**. Всяко тесто има база от **2** калории за грам и модификатор:

Модификатори:

- **`White - 1.5;`**
- **`Wholegrain - 1.0;`**
- **`Crispy - 0.9;`**
- **`Chewy - 1.1;`**
- **`Homemade - 1.0;`**

Единствената част от класа, която трябва да бъде публично достъпна, е **getter**-а за **калориите за грам**. Задачата ви е да създадете клас с **конструктор**, **полета**, **getter-и** и **setter-и.**

Стъпка 2. Валидирайте данните за клас Dough

Променете вътрешната логика на клас **Dough**, като добавите **валидация на данните** в **setter-ите.**

Ако **типът на брашното** или **техниката на изпичане** е **невалидна**, хвърлете **`Exception`** със съобщение **`"Invalid type of dough."`**.

Допустимото тегло на тестото е диапазон **[1..200] грама**. Ако е **извън** границите на този диапазон, хвърлете **`Exception`** със съобщение **`"Dough weight should be in the range [1..200]."`**.

Съобщения за грешки

- **`"Invalid type of dough."`**
- **`"Dough weight should be in the range [1..200]."`**

Създайте тест в главния метод (main), който чете Doughs (теста) и отпечатва калориите до получаване на команда"**END**".

Примери

Вход	Изход
Dough White Chewy 100 END	330.00
Dough Tip500 Chewy 100 END	Invalid type of dough.
Dough White Chewy 240 END	Dough weight should be in the range [1..200].

Стъпка 3. Създайте клас Topping

Създайте клас **Topping**. Има **четири** типа топинг – **meat (месо)**, **veggies (зеленчуци)**, **cheese (сирене)** и **sauce (сос)**. Топингът има **weight (тегло)** в грамове. **Калориите за грам** се определят **в зависимост от типа**. **Базовите** калории за грам са **2**. Всеки топинг има **modifier (модификатор)**:

Модификатори:

- **`Meat - 1.2;`**
- **`Veggies - 0.8;`**
- **`Cheese - 1.1;`**
- **`Sauce - 0.9;`**

Задачата ви е да създадете клас с **конструктор**, **полета**, **getter-и** и **setter-и**.

Стъпка 4. Валидиране на данните в клас Topping

Променете вътрешната логика на клас **Topping**, като добавите **валидация на данните** в **setter-ите.**

Ако **типът на топинга** е **невалиден**, хвърлете **`Exception`** със съобщение **`"Cannot place [name of invalid argument] on top of your pizza."`**.

Допустимото тегло на тестото е диапазон **[1..50] грама**. Ако е **извън** границите на този диапазон, хвърлете **Exception** със съобщение **"[Topping type name] weight should be in the range [1..50]."**.

Съобщения за грешка

- **"Cannot place [name of invalid argument] on top of your pizza."**
- **"[Topping type name] weight should be in the range [1..50]."**

Създайте тест в главния метод (main), който прочита данни за едно тесто и топинг, след което отпечатва калориите

Примери

Вход	Изход
Dough White Chewy 100 Topping meat 30 END	330.00 72.00
Dough White chewy 100 Topping Krenvirshi 500 END	330.00 Cannot place Krenvirshi on top of your pizza.
Dough White Chewy 100 Topping Meat 500 END	330.00 Meat weight should be in the range [1..50].

Стъпка 5. Създайте клас Pizza!

Всяка **Pizza** трябва да има **name (име)**, **toppings (топинг)** и **dough (тесто)**. Използвайте **двата класа**, които вече създадохте. Освен това всяка **Pizza** трябва да има **публични getter-и** за **името**, броя на **топингите** и **общия брой калории**. Той се изчислява, като сумирате **калориите на всички съставки на пицата**. Създайте **конструктор**, **метод** за **добавяне на топинг**, **публичен setter** за тестото и **getter** за **общия брой калории**.

Входът за всяка **Pizza** се състои от няколко реда. На **първия ред** получавате **името** на пицата, а на **втория** – **вход за тестото**. На следващите редове ще получите **всеки топинг**, който пицата има.

Ако пицата е създадена **успешно**, отпечатайте на един ред **името** на пицата и **общия брой калории**.

Стъпка 6. Валидирайте данните в клас Pizza

Името на пицата **не трябва** да бъде **празен стринг** и **трябва** да е **не по-дълго от 15 символа**. Ако не отговаря на тези изисквания, хвърлете **Exception** със съобщение **"Pizza name should be between 1 and 15 symbols."**.

Броят на топингите трябва да е в диапазона **[0..10]**. Ако е **извън** границите на този диапазон, хвърлете **Exception** със съобщение **"Number of toppings should be in range [0..10]."**.

Задачата ви е да отпечатате **името** на пицата и **общия брой калории**.

Примери

Вход	Изход
Pizza Meatless Dough Wholegrain Crispy 100 Topping Veggies 50 Topping Cheese 50 END	Meatless - 370.00 Calories.
Pizza Burgas Dough White Homemade 200 Topping Meat 123 END	Meat weight should be in the range [1..50].
Pizza Bulgarian Dough White Chewy 100 Topping Sauce 20 Topping Cheese 50 Topping Cheese 40 Topping Meat 10 Topping Sauce 10 Topping Cheese 30 Topping Cheese 40 Topping Meat 20 Topping Sauce 30 Topping Cheese 25 Topping Cheese 40 Topping Meat 40 END	Number of toppings should be in range [0..10].
Pizza Bulgarian Dough White Chewy 100 Topping Sirene 50 Topping Cheese 50 Topping Krenvirsh 20 Topping Meat 10 END	Cannot place Sirene on top of your pizza.

9. Генератор на футболен отбор

Футболен отбор (**Team**) има променлив **number of players (брой на играчите)**, **name (име)** и **rating (рейтинг)**. Всеки играч (**Player**) има **name (име)** и **stats (статистики)**, които са базата за неговия **skill level (ниво на умения)**. Статистиките на всеки играч са **endurance (издръжливост)**, **sprint (тичане)**, **passing (пас)** и **shooting (стрелба)**. Всяка статистика е **integer** (цяло число) в диапазона [0..100]. Цялостният **skill level (ниво на уменията)** представлява **средноаритметичната стойност** от статистиките. **Само името** на играча и неговите **статистики** трябва да бъдат видими за външния свят. **Всичко останало** трябва да бъде скрито.

Всеки **Team** трябва да има **name**, **rating** (средноаритметичното от **skill level-а** на всички играчи в отбора, **закръглено** до **цяло число**) и **методи** за **добавяне** и **премахване** на играчи.

Задачата ви е да създадете класове **Team** и **Player**, които да следват принципите на енкапсулацията.

Вход

Ще получавате команди до команда"**END**". Командите могат да бъдат следните:

- **"Team;{TeamName}"** – добавяне на нов отбор (**Team**);
- **"Add;{TeamName};{PlayerName};{Endurance};{Sprint};{Dribble};{Passing};{Shooting}"** – добавяне на нов играч (**Player**) към отбора (**Team**);
- **"Remove;{TeamName};{PlayerName}"** – премахване на играч (**Player**) от отбора (**Team**);
- **"Rating;{TeamName}"** – отпечатване на рейтинга на отбора (**Team**), закръглено към цяло число.

Валидация на данните

- Името не може да бъде null, празен стринг или white space. Ако е **невалидно**, отпечатайте **"A name should not be empty."**
- Статистиките трябва да бъдат в диапазона **[0...100]**. Ако са **невалидни**, отпечатайте **"[Stat name] should be between 0 and 100."**
- Ако получите команда да **премахнете** играч, който **не е** в отбора, отпечатайте print **"Player [Player name] is not in [Team name] team."**
- Ако получите команда да **добавите** играч към отбор, който **не съществува**, отпечатайте **"Team [team name] does not exist."**
- Ако получите команда да покажете **статистиките** на отбор, който **не съществува**, отпечатайте **"Team [team name] does not exist."**

Примери

Вход	Изход
Team;Arsenal Add;Arsenal;Kieran_Gibbs;75;85;84;92;67 Add;Arsenal;Aaron_Ramsey;95;82;82;89;68	Arsenal - 81

Remove;Arsenal;Aaron_Ramsey Rating;Arsenal END	
Team;Arsenal Add;Arsenal;Kieran_Gibbs;75;85;84;92;67 Add;Arsenal;Aaron_Ramsey;195;82;82;89;68 Remove;Arsenal;Aaron_Ramsey Rating;Arsenal END	Endurance should be between 0 and 100. Player Aaron_Ramsey is not in Arsenal team. Arsenal - 81
Team;Arsenal Rating;Arsenal END	Arsenal - 0