# Упражнения: Наследяване – продължение

Можете да тествате решенията си в **Judge системата**: <https://judge.softuni.org/Contests/Practice/Index/4065>

**Използвайте** качения **скелет** за последните **шест** задачи!

## Случаен списък

**Важно**: Трябва да имате клас StartUp в namespace **CustomRandomList**.

Създайте клас RandomList, който има всички функционалности на List<string>. Добавете допълнителен метод, който **връща** и **премахва** случаен елемент от списъка.

Трябва да можете да използвате класа по следния начин:

Text

Description automatically generated with medium confidence

## Поредица от стрингове

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

Създайте клас StackOfStrings, който разширява **Stack**, може да съхранява **само стрингове** и има следните функционалности:

* Публичен метод: IsEmpty(): bool
* Публичен метод: **AddRange(): Stack<string>**

Graphical user interface, text, application

Description automatically generated

## Следа от изключения

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

Прочетете всички редове от файл и сумирайте числата в него. Използвайте клас MyFileReader, който има поле и свойство path, **конструктор** и void метод ReadAndSum(). Ако пътят към файла (path) е **null** или **празен стринг**, хвърлете нов ArgumentException със съобщение "Invalid Path or File Name.".

Методът ReadAndSum() трябва да чете файла и да конвертира всяко число. Ако някоя от стойностите във файла **не може** да бъда конвертирана, хвърлете нов ArgumentException със съобщение"Error: On the line {line number} of the file the value was not in the correct format.".

Ако всичко е успешно, отпечатайте: "The sum of all correct numbers is: {numbers sum}".

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

## Човек

Създайте приложение, което съхранява данните за различни хора. Трябва да имате класовете **Person** и **Child**. **Person** трябва да е **базов** клас, а **Child** трябва да **произлиза** от **Person**.

### Важно

Имената на класовете **ТРЯБВА** да бъдат същите като тези, показани в следния пример:

A picture containing text, person, screenshot

Description automatically generated

Създайте нов празен клас с име **Person**. Задайте модификатор за достъп **public**, за да може да бъде инстанциран от всеки проект. Всеки човек има **name** (име) и **age** (възраст).

|  |
| --- |
| **Примерен код** |
| public class Person  {  // 1. Add Fields  // 2. Add Constructor  // 3. Add Properties  // 4. Add Methods  } |

### Стъпка 1 – Дефинирайте полета и свойства

Дефинирайте **полета** и **свойства** за **name** и **age**

### Стъпка 2 – Дефинирайте конструктор

Дефинирайте конструктор, който приема **name** и **age**.

Text

Description automatically generated with low confidence

### Стъпка 3 – Презапишете ToString()

Както вероятно знаете, всички класове в C# наследяват класа **Object** и следователно имат достъп до всички **публични** членове на този клас (**ToString()**, **Equals()** и **GetHashCode()**).

**ToString()** връща информация за инстанцията като стринг. **Презапишете** (променете) поевдението му за нашия клас **Person**.

Graphical user interface, text, application

Description automatically generated

Ако всичко е правилно, сега можете да създавате **обекти** **Person** и да показвате информация за тях.

### Стъпка 4 – Създайте клас Child

Създайте клас **Child**, който наследява **Person,** преизползва част от базовия конструктор и приема две инстанции на **Person** като **mother (майка)** и **father (баща)**. Дефинирайте свойства за **Mother** и **Father** от тип **Person** в класа **Child**.

**Важно**: не копирайте кода от класа **Person** – преизползвайте конструктора на **Person** и го допълнете.

**Няма нужда** да пренаписвате свойствата **Name** и **Age**, защото **Child** наследява **Person** и ги има по подразбиране.

Graphical user interface, text, application

Description automatically generated

Презапишете метода **ToString()** и го допълнете, като използвате същия метод на **базовия клас**. Добавете следния стринг: ", Mother: { Mother Name }, Father: { Father Name }".

Text

Description automatically generated

Ще получите следните данни (всяка част на нов ред): името на детето, неговата възраст, името на майката, нейната възраст, името на бащата и неговата възраст. Отпечатайте обекта от клас **Child**.



### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Peter  12  Maria  36  George  39 | Name: Pesho, Age: 12, Mother: Maria, Father: George |

## Зоологическа градина

Използвайте проекта **Zoo**. Следвайки схемата, създайте следната **йерархия** на **класовете**:

Diagram

Description automatically generated

**Всеки клас**, с изключение на **Animal**, трябва да наследява **друг клас**. Всеки клас трябва да има:

* Конструктор, който приема един параметър:
  + **name - string**
* Свойство**:**
  + **Name - string**.

Ще получавате на нов ред имена за: **Gorilla**, **Snake**, **Lizard** и **Bear**. Отпечатайте всяко животно на нов ред в следния формат:

* "Gorilla's name: {име на горилата}"
* "Snake's name: {име на змията}"
* "Lizard's name: {име на гущера}"
* "Bear's name: {име на мечката}"

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Isabel  Jorge  Miranda  Carlos | Gorilla's name: Isabel  Snake's name: Jorge  Lizard's name: Miranda  Bear's name: Carlos |

Направете **zip** с файловете **без** папките **bin** и **obj** и го качете в Judge.

## Играчи и чудовища

Вашата задача е да създадете следната **йерархия** от **класове**:

Diagram

Description automatically generated

Създайте клас **Hero**. Той трябва да има следните членове:

* Конструктор, който приема:
  + **username – string**
  + **level – int**
* Публични свойства за:
  + **Username - string**
  + **Level – int**
* Метод **ToString()**

### Вход

Ще получите следния **вход**:

1. Първи ред - **hero** **type** (тип на героя)
2. Втори ред – **name** (име на героя)
3. Трети ред – **level** (ниво на героя)

### Изход

Презапишете метода **ToString()** на базовия клас по следния начин:

|  |
| --- |
|  |
| public override string ToString()  {  return $"Type: {this.GetType().Name} Username: {this.Username} Level: {this.Level}";  } |

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| BladeKnight  Fenris  24 | Type: BladeKnight Username: Fenris Level: 24 |
| Wizard  Fredo  215 | Type: Wizard Username: Fredo Level: 215 |

## Need for Speed

Създайте **йерархия** със следните **класове**: Diagram

Description automatically generated

Създайте базов клас **Vehicle**. Той трябва да има следните членове:

* Конструктор, който приема следните параметри:
  + **horsePower - int**
  + **fuel - double**
* **DefaultFuelConsumption – double**
* **FuelConsumption – virtual double**
* **Fuel – double**
* **HorsePower – int**
* **virtual void Drive(double kilometers)**
  + Методът **Drive** трябва да намалява горивото (**Fuel**) на база на изминатите километри.

По подразбиране консумацията на гориво (**FuelConsumption**) за един **Vehicle** е **1.25**. Някои от класовете имат различна консумация на гориво по подразбиране:

* **SportCar – DefaultFuelConsumption = 10**
* **RaceMotorcycle – DefaultFuelConsumption = 8**
* **Car – DefaultFuelConsumption = 3**

### Вход и изход

Ще получите следния **вход**:

1. Първи ред – **тип** на превозното средство - Vehicle, Motorcycle, Car, RaceMotorcycle, CrossMotorcycle, FamilyCar или SportCar
2. Втори ред – **конски сили**
3. Трети ред – **гориво**
4. Четвърти ред – **изминати километри**

Отпечатайте оставащото гориво в следния формат: "Left fuel {remaining fuel}". **Форматирайте** оставащото гориво до **втората цифра** след десетичната запетая.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| FamilyCar  80  73.5  7 | Left fuel 52.50 |
| RaceMotorcycle  95  55.5  5 | Left fuel 15.50 |

Създайте **zip** на вашето решение **без** папките **bin** и **obj** и качете zip файла в Judge.

## Животни

**Важно**: в тази задача трябва да дефинирате **виртуален** метод в базовия клас и да го **презапишете** в производните класове. Научете повече за **virtual** методите тук: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual>.

Създайте йерархия от **Animals**. Програмата трябва да има три вида животни - **Dog**, **Frog** и **Cat**. На следващо ниво в йерархията трябва да имате два допълнителни класа - **Kitten** и **Tomcat**. **Kittens** са **женски (female)**, a **Tomcats** са **мъжки (male).** Всички типове животни могат да възпроизвеждат някакъв **звук** - **virtual** **ProduceSound()**. Звуците на различните животни са:

* **Dog: "Woof!"**
* **Cat: "Meow meow"**
* **Frog: "Ribbit"**
* **Kittens: "Meow"**
* **Tomcat: "MEOW"**

### Вход и изход

Всяка последователност от два реда от входа представлява животно:

* Първи ред – **тип** наживотното
* Втори ред – **име**, **възраст** и **пол** на животното

При получаване на команда "**Beast!**" прекратете приема на вход и отпечатайте животните в следния формат:

* Първи ред **"{AnimalType}"**
* Втори ред: **"{Name} {Age} {Gender}"**
* Трети ред – звукът, който животното възпроизвежда: **"{ProduceSound()}"**

### Забележки

* Всеки **Animal** трябва да има **name (име)**, **age (възраст)** и **gender (пол)**
* **Не трябва** да има **празни полета**
* Ако получите вход за **gender** на **Tomcat** или на **Kitten**, игнорирайте този пол, но **създайте** животното
* Ако входът за някое свойство е невалиден, хвърлете изключение със следното съобщение: **"Invalid input!"**
* Всяко животно трябва да има метод за възпроизвеждане на звук **ProduceSound()** като **override** метод.

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| Cat  Tom 12 Male  Dog  Sharo 132 Male  Beast! | Cat  Tom 12 Male  Meow meow  Dog  Sharo 132 Male  Woof! |
| Frog  Kermit 12 Male  Beast! | Frog  Kermit 12 Male  Ribbit |
| Frog  Sasha -2 Male | Invalid input! |