Упражнения: Наследяване

Можете да тествате решенията си в **Judge системата**: https://judge.softuni.bg/Contests/3164/Inheritance

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

1. Куче наследява животно

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

Създайте два класа - **Animal** и **Dog**:

- **Animal** трябва да има един публичен метод **Eat()**, който отпечатва: **"eating..."**
- **Dog** трябва да има един публичен метод **Bark()**, който отпечатва: **"barking..."**
- **Dog** трябва да наследява **Animal**

Трябва да можете да използвате класа по следния начин:

```
static void Main(string[] args)
{
    Dog dog = new Dog();
    dog.Eat();
    dog.Bark();
}
```

2. Верижно наследяване

Важно: трябва да имате публичен клас **StartUp** в namespace **Farm**.

Създайте три класа - **Animal**, **Dog** и **Puppy**:

- **Animal** трябва да има един публичен метод **Eat()**, който отпечатва **"eating..."**
- **Dog** трябва да има един публичен метод **Bark()**, който отпечатва: **"barking..."**
- **Puppy** трябва да има един публичен метод **Weep()**, който отпечатва: **"weeping..."**
- **Dog** трябва да наследява **Animal**
- **Puppy** трябва да наследява **Dog**

Трябва да можете да ползвате класа по следния начин:

```
static void Main(string[] args)
{
    Puppy puppy = new Puppy();
    puppy.Eat();
    puppy.Bark();
    puppy.Weep();
}
```

3. Наследствена йерархия

Важно: трябва да имате публичен клас **`StartUp`** в namespace **Farm**.

Създайте три класа - **`Animal`**, **`Dog`** и **`Cat`**:

- **`Animal`** трябва да има един публичен метод **`Eat()`**, който отпечатва: **"eating..."**
- **`Dog`** трябва да има един публичен метод **`Bark()`**, който отпечатва: **"barking..."**
- **`Cat`** трябва да има един публичен метод **`Meow()`**, който отпечатва: **"meowing..."**
- **`Dog`** и **`Cat`** трябва да наследяват **`Animal`**

Трябва да можете да използвате класа по следния начин:

```
static void Main(string[] args)
{
    Dog dog = new Dog();
    dog.Eat();
    dog.Bark();

    Cat cat = new Cat();
    cat.Eat();
    cat.Meow();
}
```

4. 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:

```
static void Main(string[] args)
{
    RandomList list = new RandomList();
    list.Add("Bond");
    list.Add("Lind");
    list.Add("Nyle");
    list.Add("Parker");

    Console.WriteLine(list.Count);
    Console.WriteLine(list.RandomString());
    Console.WriteLine(list.Count);
}
```

5. 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>**

```
static void Main(string[] args)
{
    StackOfStrings stackOfStrings = new StackOfStrings();

    Console.WriteLine(stackOfStrings.IsEmpty()); //True

    Stack<string> fullStack = new Stack<string>();

    fullStack.Push("b");
    fullStack.Push("c");

    stackOfStrings.AddRange(fullStack);

    Console.WriteLine(stackOfStrings.IsEmpty()); //False

}
```

6. 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}"**.

```
public class MyFileReader
{
    private string path;
    2 references
    public MyFileReader(string path)
    {
        this.Path = path;
    }
    2 references
    public string Path
    {
        get { return path; }
        set
        {
            if (string.IsNullOrEmpty(value))
            {
                throw new ArgumentException("Invalid Path or File Name.");
            }
            path = value;
        }
    }
```

```
public void ReadAndSum()
{
    string[] inputFromFile = File.ReadAllLines(this.Path);
    List<int> numbers = new List<int>();
    int countRow = 0;

    foreach (var value in inputFromFile)
    {
        countRow++;
        try
        {
            int currentNum = int.Parse(value);
            numbers.Add(currentNum);

        }
        catch (Exception)
        {
            throw new ArgumentException($"Error: On the line {countRow} " +
                $"of the file the value was not in the correct format.");
        }
    }
```

```
static void Main(string[] args)
{
    try
    {
        MyFileReader reader1 = new MyFileReader(@"C:\temp\numbers.txt");
        reader1.ReadAndSum();
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine("Error: " + ex.Message);
    }

    try
    {
        MyFileReader reader2 = new MyFileReader(@"");
        reader2.ReadAndSum();
    }
    catch (Exception ex)
    {
        Console.Error.WriteLine("Error: " + ex.Message);
    }
```

7. Човек

Създайте приложение, което съхранява данните за различни хора. Трябва да имате **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:

```
public static void Main(string[] args)
{
    string childName = Console.ReadLine();
    int childAge = int.Parse(Console.ReadLine());

    string motherName = Console.ReadLine();
    int motherAge = int.Parse(Console.ReadLine());

    string fatherName = Console.ReadLine();
    int fatherAge = int.Parse(Console.ReadLine());

    Person mother = new Person(motherName, motherAge);
    Person father = new Person(fatherName, fatherAge);

    Child child = new Child(childName, childAge, mother, father);
    Console.WriteLine(child);

}
```

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**.

```
public Person(string name, int age)
{
    this.Name = name;
    this.Age = age;
}
```

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`**.

```
public override string ToString()
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.Append(String.Format("Name: {0}, Age: {1}",
                         this.Name,
                         this.Age));

    return stringBuilder.ToString();
}
```

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.

```
public Person Mother { get; set; }
2 references
public Person Father { get; set; }
1 reference
public Child(string name, int age, Person mother, Person father)
    : base(name, age)
{
    this.Mother = mother;
    this.Father = father;
}
```

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

```
public override string ToString()
{
    StringBuilder stringBuilder = new StringBuilder();
    stringBuilder.Append(String.Format(base.ToString() +
        [illegible]

    return stringBuilder.ToString();
}
```

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**.

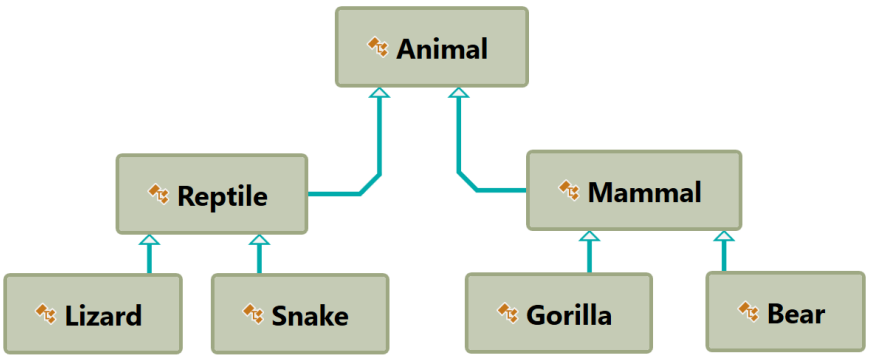
```
Child child = new Child(childName, childAge, mother, father);
Console.WriteLine(child);
```

Примери

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

8. Зоологическа градина

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



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}"`**

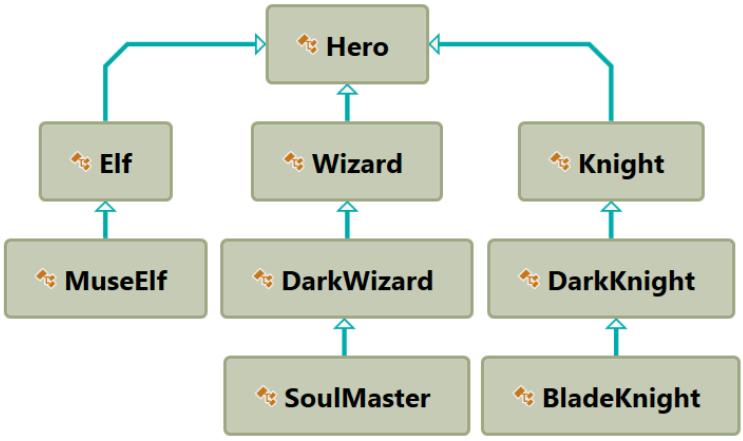
Примери

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.

9. Играчи и чудовища

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



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()**

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

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

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)**

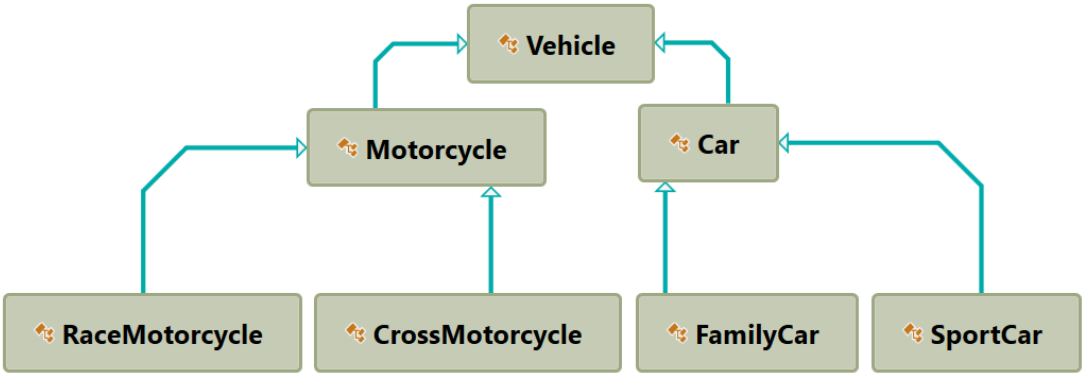
Примери

Вход	Изход
BladeKnight Fenris 24	Type: BladeKnight Username: Fenris Level: 24

Wizard Fredo 215	Type: Wizard Username: Fredo Level: 215

10. Need for Speed

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



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`**

Ще получите следния вход:

1. Първи ред – тип на превозното средство (**vehicle type**) - Vehicle, Motorcycle, Car, RaceMotorcycle, CrossMotorcycle, FamilyCar или SportCar
2. Втори ред – конски сили (**horsepower**)
3. Трети ред – гориво (**fuel**)
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.

11. Ресторант

Създайте проект **Restaurant** с **йерархия** от следните **класове**:

Има **Food (храна)** и **Beverages (напитки)**– това са продуктите в ресторанта.

Класът **Product** трябва да има следните членове:

- Конструктор, който приема следните параметри:
  - **name - string**
  - **price – decimal**
- Свойства за:
  - **Name – string**
  - **Price – decimal**

Класовете **Beverage** и **Food** са продукти.

Клас **Beverage** трябва да има следните членове:

- Конструктор, който приема следните параметри
  - **name – string**
  - **price – decimal**
  - **milliliters – double**

- Преизползвайте конструктора от базовия клас

- Свойства за:
  - **Name – string**
  - **Price – double**
  - **Milliliters – double**

HotBeverage (гореща напитка) и **ColdBeverage (студена напитка)** са напитки и приемат следните параметри при инициализация:

- **name - string**
- **price – decimal**
- **milliliters – double**
- Преизползвайте конструктора на родителския клас.

Coffee и **Tea** са **hot bevarages (горещи напитки).** Класът **Coffee** трябва да има следните допълнителни членове:

- **CoffeeMilliliters - double = 50**
- **CoffeePrice - decimal = 3.50**
- **Caffeine – double**

Класът **Food** трябва да има следните членове:

- Конструктор, който приема следните параметри
  - **name – string**
  - **price – decimal**
  - **grams – double**

- Свойства за:
  - **Name – string**
  - **Price – decimal**
  - **Grams – double**

MainDish, **Dessert** и **Starter** са видове храна. При инициализация всеки от тези класове приема следните параметри:

- **name – string**
- **price – decimal**
- **grams - double**
- Преизползвайте конструктора на родителския клас.

Dessert трябва да получи **още един** параметър в своя **конструктор**: **double calories**, и да има свойство **Calories**

Добавете класовете **Fish**, **Soup** и **Cake** , които трябва да наследяват съответния родителски клас (**MainDish**, **Starter** и **Dessert**).

Класът **Cake** трябва да има следните стойности по подразбиране:

- **Grams = 250**
- **Calories = 1000**
- **CakePrice = 5**

Класът **Fish** трябва да има следните стойности по подразбиране:

- **Grams = 22**

До получаване на команда "**End**", на всеки нов ред ще получавате **поръчка** за Fish, Soup, Cake, Coffee или Tea. Данните за поръчката ще бъдат разделени с интервал и ще бъдат в следния формат:

- Coffee <name> <caffeine>
- Tea <name> <price> <millilitres>
- Fish <name> <price>
- Soup <name> <price> grams>
- Cake <name>

След получаване на команда "**End**" отпечатайте поръчката в следния **формат**:

```
"Your order contains:"
"  Quantity of liquids: {millilitres beverage}"
"  Grams of food {grams food}"
"  Final amount {amount}"
```

Ако **няма информация** за **калориите**, отпечатайте:

```
"Your order contains:"
"  Quantity of liquids: {millilitres beverage}"
"  Grams of food {grams food}"
"  Calories {calories}"
"  Final amount {amount}"
```

Примери

Вход	Изход
Coffee Frappe 1.3 Tea IceTea 1.50 200	Your order contains: Quantity of liquids: 200

Soup Chicken 4.50 250 End	Grams of food 250 Final amount 9.50
Coffee Espresso 2.5 Fish Tuna 5.20 Cake Cheesecake Cake Gingerbread End	Your order contains: Quantity of liquids: 50 Grams of food 522 Calories 2000 Final amount 18.70

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

12. 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)**, а **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** на животното
- 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!