**Exercise: Inheritance**

Problems for exercise and homework for the ["C# OOP" course @ SoftUni".](https://softuni.bg/trainings/3585/csharp-oop-february-2022)

You can check your solutions <https://judge.softuni.org/Contests/1500/Inheritance-Exercise>

**Use** the **provided** **skeleton** for each of the exercises.

* **Person**

You are asked to model an application for storing data about people. You should be able to have a **person** and a **child**. The **child** derives from the **person**. Your task is to model the application. The only constraints are:

* People should **not** be able to have a **negative age**
* Children should **not** be able to have an age **greater than 15**.
* **Person** – represents the base class by which all of the others are implemented
* **Child** - represents a class, which derives from **Person.**

**Note**

Your class’s names **MUST** be the same as the names shown above!!!

|  |
| --- |
| **Sample Main()** |
| static void Main()  {  string name = Console.ReadLine();  int age = int.Parse(Console.ReadLine());    Child child = new Child(name, age);  Console.WriteLine(child);  } |

Create a new empty class and name it **Person**. Set its access modifier to the **public** so it can be instantiated from any project. Every person has a **name** and an **age**.

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

* Define a **field** for each property the class should have (e.g. **Name**, **Age**)
* Define the **Name** and **Age** properties of a Person.

**Step 1 – Define a Constructor**

Define a constructor that accepts **name and age**.

|  |
| --- |
| **Sample Code** |
| public Person(string name, int age)  {  this.Name = name;  this.Age = age;  } |

**Step 2 – Override ToString()**

As you probably already know, all classes in C# inherit the **Object** class and therefore have all its **public** members (**ToString()**, **Equals(),** and **GetHashCode()** methods). **ToString()** serves to return information about an instance as а string. Let's **override** (change) its behavior for our **Person** class.

|  |
| --- |
| **Sample Code** |
| public override string ToString()  {  StringBuilder stringBuilder = new StringBuilder();  stringBuilder.Append(String.Format("Name: {0}, Age: {1}",  this.Name,  this.Age));  return stringBuilder.ToString();  } |

And voila! If everything is correct, we can now create **Person objects** and display information about them.

**Step 3 – Create a Child**

Create a **Child** class that inherits **Person** and has the same constructor definition. However, do not copy the code from the Person class - **reuse the Person class' constructor**.

|  |
| --- |
| **Sample Code** |
| public Child(string name, int age)  : base(name, age)  {  } |

There is **no need** to rewrite the **Name** and **Age** properties since the **Child** inherits **Person** and by default has them.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter  12 | Name: Peter, Age: 13 |

* **Zoo**

**NOTE**: You need a public class **StartUp**.

Create a project **Zoo**. It needs to contain the following classes:



Follow the diagram and create all of the classes. **Each** of them, except the **Animal** class, should **inherit** from **another** **class**. Every class should have:

* A constructor, which accepts one parameter: **name**.
* Property **Name - string**.

Zip your solution without the bin and obj folders and upload it in Judge.

* **Players and Monsters**

NOTE: You need a public class **StartUp**.

Your task is to create the following game hierarchy:



Create a class Hero. It should contain the following members:

* A constructor, which accepts:
* **username – string**
* **level – int**
* The following properties:
* **Username - string**
* **Level – int**
* **ToString()** method

Hint: Override **ToString()** of the base class in the following way:

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