Exercises: Inheritance

Problems for exercises and homework for the "C# OOP" course @ SoftUni".

You can check your solutions here: https://judge.softuni.bg/Contests/1500/Inheritance-Exercise

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

Problem 1. 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:

- **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 **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 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 **Child** inherits **Person** and by default has them.

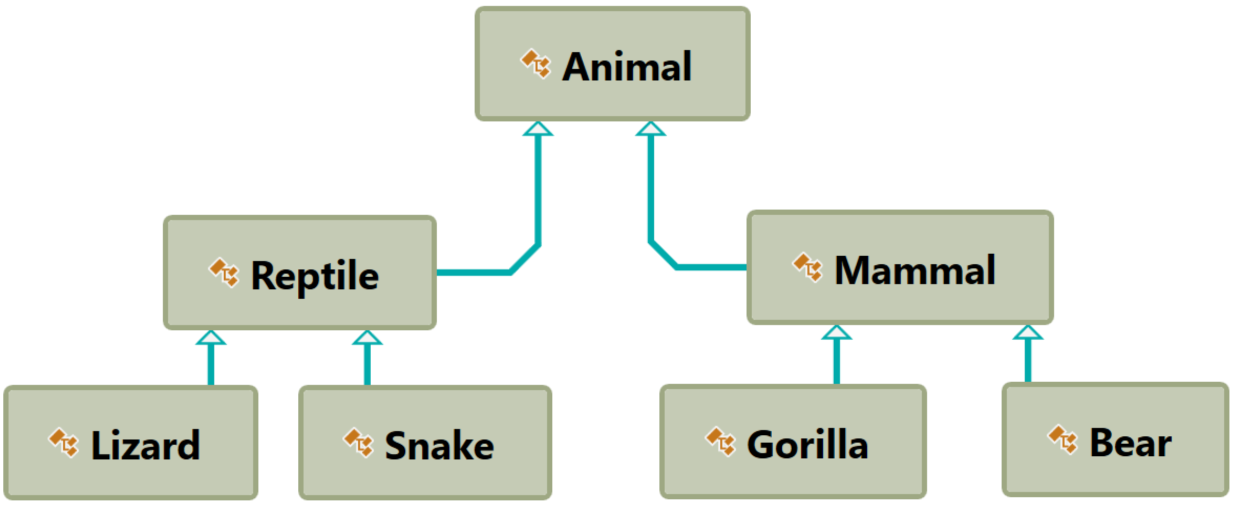
Examples

Input	Output
Pesho 12	Name: Pesho, Age: 13

Problem 2. 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:

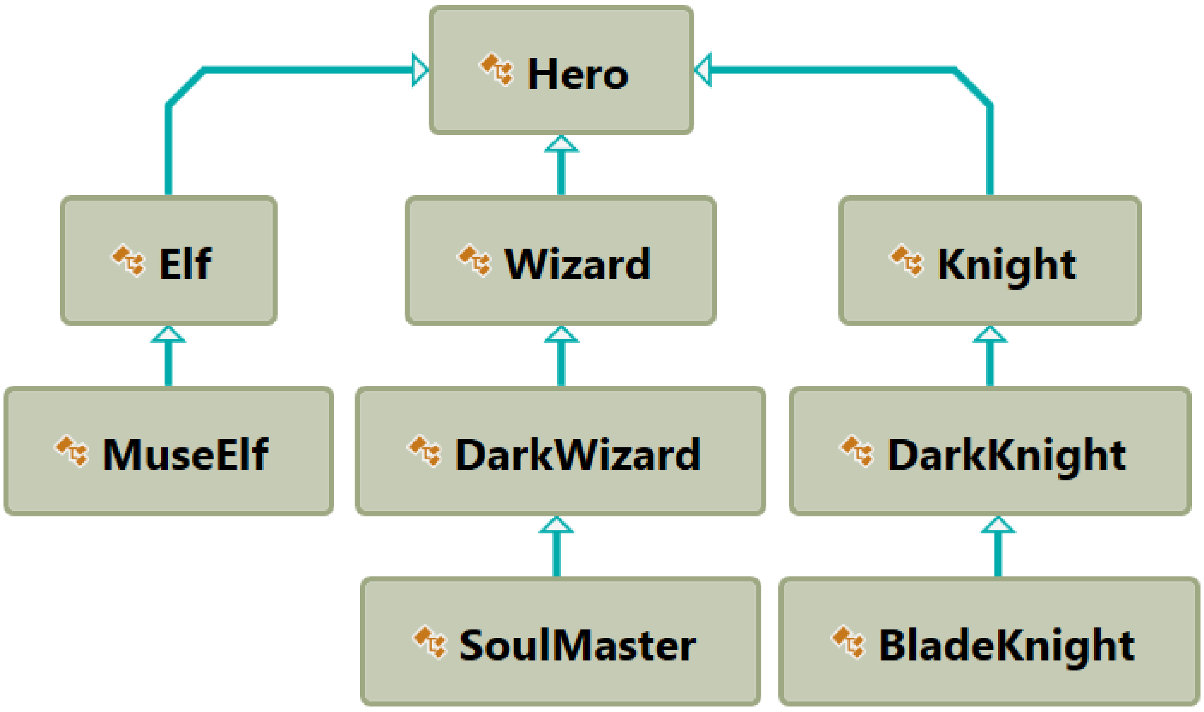
- A constructor, which accepts one parameter: **name**.
- Property **Name - string**.

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

Problem 3. 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:

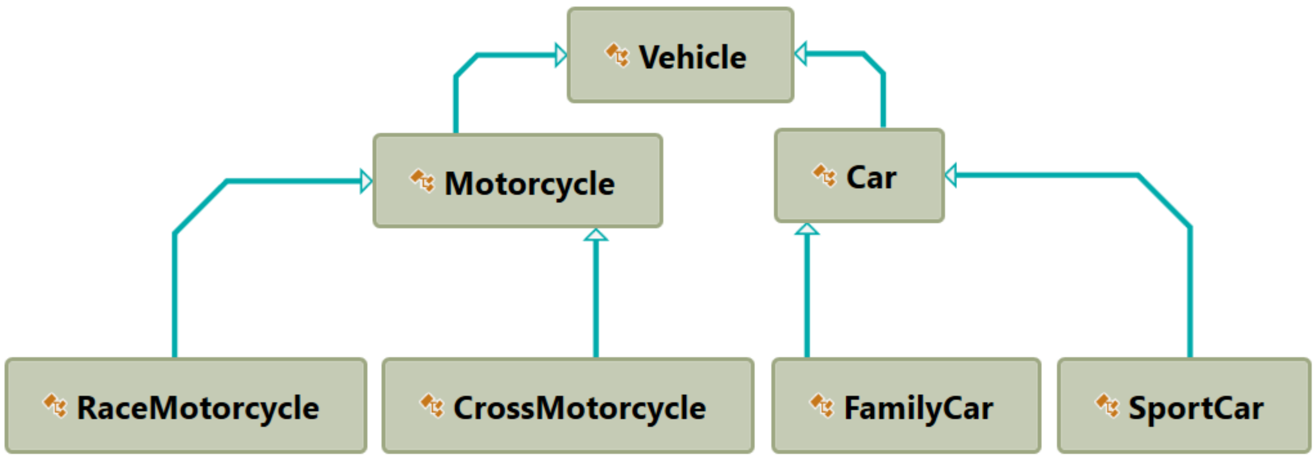
- A constructor, which accepts:
  - **username – string**
  - **level – int**
- The following properties:
  - **Username - string**
  - **Level – int**
- **ToString()** method

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}";
}
```

Problem 4. Need for Speed

NOTE: You need a public class **StartUp**. Create the following **hierarchy** with the following **classes**:



Create a base class **Vehicle**. It should contain the following members:

- A constructor that accepts the following parameters: **int horsePower**, **double fuel**
- **DefaultFuelConsumption – double**
- **FuelConsumption – virtual double**
- **Fuel – double**
- **HorsePower – int**
- **virtual void Drive(double kilometers)**
    - The **Drive** method should have a functionality to reduce the **Fuel** based on the travelled kilometers.

The default fuel consumption for **Vehicle** is **1.25**. Some of the classes have different default fuel consumption values:

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

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

Problem 5. Restaurant

NOTE: You need a public class **StartUp**. Create a **Restaurant** project with the following classes and hierarchy:

There are **Food** and **Beverages** in the restaurant and they are all products.

The **Product** class must have the following members:

- A constructor with the following parameters: **string name, decimal price**
- **Name – string**

- **Price – decimal**

Beverage and **Food** classes are products.

The **Beverage** class must have the following members:

- A constructor with the following parameters**: string name, decimal price, double milliliters**
    - Reuse the constructor of the inherited class
- **Name – string**
- **Price – double**
- **Milliliters – double**

HotBeverage and **ColdBeverage** are beverages and they accept the following parameters upon initialization: **string name, decimal price, double milliliters.** Reuse the constructor of the inherited class.

Coffee and **Tea** are hot beverages. The **Coffee** class must have the following additional members:

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

The **Food** class must have the following members:

- A constructor with the following parameters**: string name, decimal price, double grams**
- **Name – string**
- **Price – decimal**
- **Grams – double**

MainDish, **Dessert** and **Starter** are food. They all accept the following parameters upon initialization: **string name, decimal price, double grams**. Reuse the base class constructor.

Dessert must accept **one more** parameter in its **constructor**: **double calories**, and has a property:

- **Calories**

Make **Fish**, **Soup** and **Cake** inherit the proper classes.

The **Cake** class must have the following default values:

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

A **Fish** must have the following default values:

- **Grams = 22**

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