Lab: Basic Inheritance and Polymorphisms

Part I: Inheritance

Single Inheritance

NOTE: You need a public **`StartUp`** class with the namespace **`Farm`**.

Create two classes named **`Animal`** and **`Dog`**:

- **`Animal`** with a single public method **`Eat()`** that prints: **"eating…"**
- **`Dog`** with a single public method **`Bark()`** that prints: **"barking…"**
- **`Dog`** should inherit from **`Animal`**

Sample Main()

```
static void Main()
{
     Dog dog = new Dog();
     dog.Bark();
     dog.Bark();
}
```

Hints

Use the **: operator** to build a hierarchy

Multiple Inheritance

NOTE: You need a public **`StartUp`** class with the namespace **Farm**.

Create three classes named **`Animal`**, **`Dog`** and **`Puppy`**:

- **`Animal`** with a single public method **`Eat()`** that prints: **"eating…"**
- **`Dog`** with a single public method **`Bark()`** that prints: **"barking…"**
- **`Puppy`** with a single public method **`Weep()`** that prints: **"weeping…"**
- **`Dog`** should inherit from **`Animal`**
- **`Puppy`** should inherit from **`Dog`**

Sample Main()

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

Hierarchical Inheritance

NOTE: You need a public **`StartUp`** class with the namespace **Farm**.

Create three classes named **`Animal`**, **`Dog`** and **`Cat`**:

- **`Animal`** with a single public method **`Eat()`** that prints: **"eating…"**
- **`Dog`** with a single public method **`Bark()`** that prints: **"barking…"**
- **`Cat`** with a single public method **`Meow()`** that prints: **"meowing…"**
- **`Dog`** and **`Cat`** should inherit from **`Animal`**

Sample Main()
```
static void Main()
{
    Dog dog = new Dog();
    dog.Eat();
    dog.Bark();

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

Sort Persons by Name and Age

NOTE: You need a public **`StartUp`** class with the namespace **`PersonsInfo`**.

Create a class **`Person`**, which should have **`public`** properties with **`private`** setters for:

- **`FirstName: string`**
- **`LastName: string`**
- **`Age: int`**
- **`ToString(): string - override`**

You should be able to use the class like this:

StartUp.cs

```
public static void Main()
{
    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()));
}
```

Examples

Input	Output
5 Asen Ivanov 65 Boiko Borisov 57 Ventsislav Ivanov 27 Asen Harizanoov 44 Boiko Angelov 35	Asen Harizanoov is 44 years old. Asen Ivanov is 65 years old. Boiko Angelov is 35 years old. Boiko Borisov is 57 years old. Ventsislav Ivanov is 27 years old.

Solution

Create a **new class** and ensure **proper naming**. Define the **`public`** properties:

Person.cs

```
public class Person
{
    public string FirstName { get; private set; }

    public string LastName { get; private set; }

    public int Age { get; private set; }
}
```

Create a constructor for **`Person`**, which takes 3 parameters **`firstName`**, **`lastName`**, **`age`**:

Person.cs

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

Override **`ToString()`** method:

Person.cs

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

Salary Increase

NOTE: You need a public **`StartUp`** class with the namespace **`PersonsInfo`**. **Refactor the project from the last task.**

Create objects of the class **`Person`**. Read their **`name`**, **`age`** and **`salary`** from the console. Read the percentage of the bonus to every **`Person salary`**. People younger than **30 get half the increase**. Expand **`Person`** from the previous task.
New **properties** and **methods:**

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

You should be able to use the class like this:

StartUp.cs

```
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()));
```

Examples

Input	Output
5 Asen Ivanov 65 2200 Boiko Borisov 57 3333 Ventsislav Ivanov 27 600 Asen Harizanoov 44 666.66 Boiko Angelov 35 559.4 20	Asen Ivanov receives 2640.00 leva. Boiko Borisov receives 3999.60 leva. Ventsislav Ivanov receives 660.00 leva. Asen Harizanoov receives 799.99 leva. Boiko Angelov receives 671.28 leva.

Solution

Add a new **`public`** property for **`salary`** and **refactor the constructor**. Add a new **method**, which will **update `salary`** with a bonus:

Person.cs

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

Refactor the **`ToString()`** method for this task.