Exercises: Inheritance

You can check your solutions in **Judge system**: https://judge.softuni.bg/Contests/3164/Inheritance

Use the **provided skeleton** for the last six exercises!

1. Dog Inherits Animal

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**

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

Hints

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

2. Inheritance Chain

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**

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

3. Inheritance Hierarchy

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

SoftUni

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:

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**

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

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

4. Random List

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

Create a **RandomList** class that has all the functionality of **List<string>**. Add additional function that **returns** and **removes** a random element from the list.

RandomList class elements:

- Private **field**
- Public **Constructor**
- Public method: **RandomString(): string**

```
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. Stack of Strings

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

Create a class **StackOfStrings** that extends **Stack**, can store only strings, and has the following functionality:

- Public method: **IsEmpty(): bool**
- Public method: **AddRange(): Stack<string>**

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

```
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. Exception Trace

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

Read all lines from a file and sum the numbers in it. Use class **MyFileReader** who has field and property **path**, constructor and void method **ReadAndSum()**. If the file path is null or empty throw new **ArgumentException** with message **"Invalid Path or File Name."**.

The method **ReadAndSum()** should read the file and parse each number, if any value in the file cannot be parsed throw new **ArgumentException** with message **"Error: On the line {line number} of the file the value was not in the correct format."**.

If everything is successful, print: **"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;
        }
    }
```


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

```
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. 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. It should contain:

- **Person** – represents the base class by which all of the others are implemented.
- **Child** - represents a class, which derives from **Person**.

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

Note

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

```
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);

}
```

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**.

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


© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

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.

```
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,** reuses part of the base class constructor and accepts two **Person as mother and father**. Define a properties **Mother** and **Father** of type **Person** in the **Child** class.

However, do not copy the code from the **`Person`** class - **reuse the `Person` class' constructor and supplemented it**.

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

```
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;
}
```

Overwrite method **`ToString()`** and complete it using the already overwritten method in the **base** class. Add the following string: **`", Mother: { Mother Name }, Father: { Father Name }"`**.

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

    return stringBuilder.ToString();
}
```

You will receive the following data each on a new line: child's name, his age, mother's name, her age, father's names and his age. Print object of class **Child**.

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

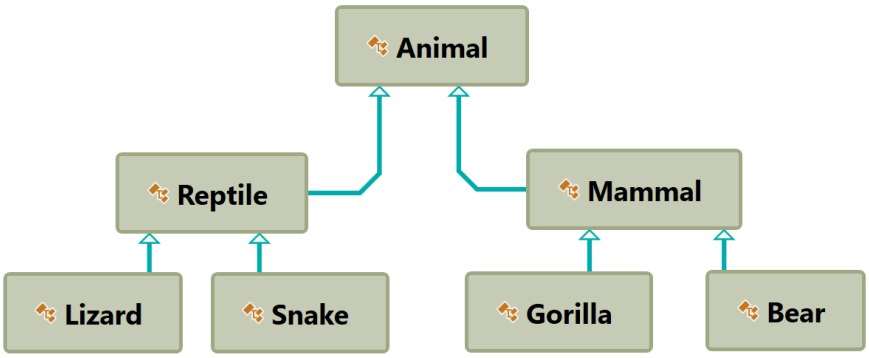

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Examples

Input	Output
Peter 12 Maria 36 George 39	Name: Pesho, Age: 12, Mother: Maria, Father: George

8. Zoo

Use the project **Zoo**. Following the picture, create the following **hierarchy** of **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`**.

You will receive each on a new line, the names for: **Gorilla**, **Snake**, **Lizard** and **Bear**. Print them each on a new line in the following formats:

- **`"Gorilla's name: {gorilla name}"`**
- **`"Snake's name: {snake name}"`**
- **`"Lizard's name: {lizard name}"`**
- **`"Bear's name: {bear name}"`**

Examples

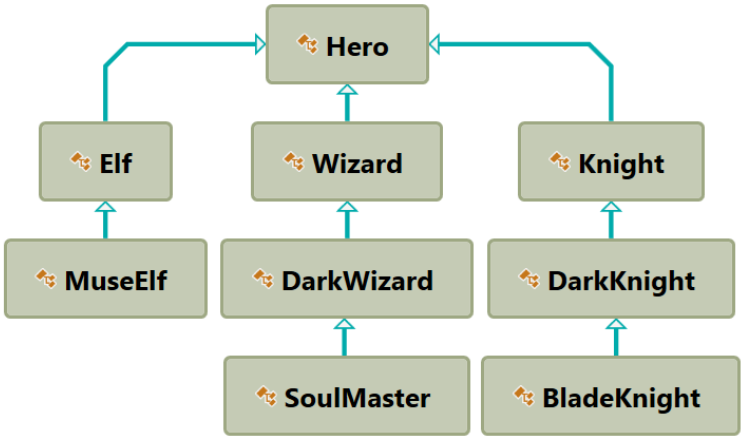
Input	Output
Isabel Jorge Miranda Carlos	Gorilla's name: Isabel Snake's name: Jorge Lizard's name: Miranda Bear's name: Carlos

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

SoftUni
© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

9. Players and Monsters

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}";
}
```

On the first line you will receive the **hero type ()**, on the second line you will receive the **name** of the hero and on the third line you will receive the hero **level**. **Print** the given **hero**.

Example: `Console.WriteLine(hero);`

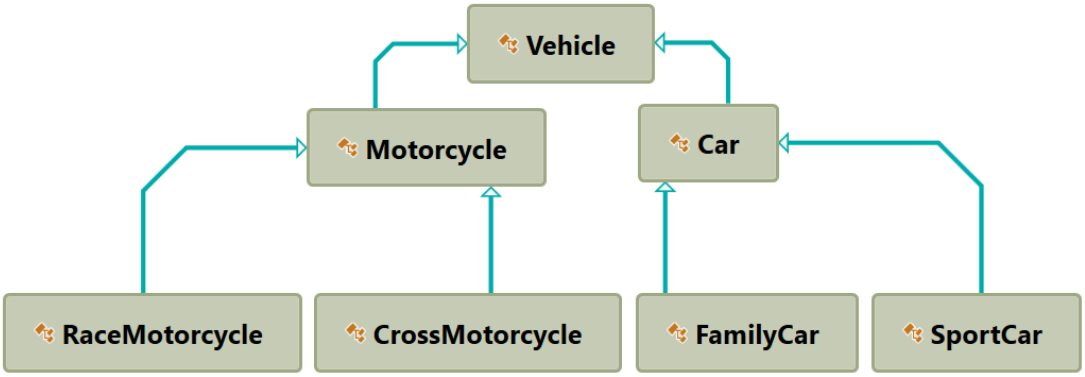
Examples

Input	Output
BladeKnight Fenris 24	Type: BladeKnight Username: Fenris Level: 24
Wizard Fredo 215	Type: Wizard Username: Fredo Level: 215

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

10. Need for Speed

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`**

On the **first line** you will receive the **vehicle type** (Vehicle, Motorcycle, Car, RaceMotorcycle, CrossMotorcycle, FamilyCar, SportCar), on the **second line** you will receive the **horsepower,** on the **third line** you will receive the **fuel** and on the **last line** the **driven kilometers**. Print the remaining fuel in the following format: **`"Left fuel {remaining fuel}"`**. **Format** the value of the remaining fuel to the **second number** after the decimal point.

Examples

Input	Output
FamilyCar 80 73.5 7	Left fuel 52.50
RaceMotorcycle 95 55.5 5	Left fuel 15.50

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

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

11. Restaurant

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**

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

You will receive an unknown amount of lines from the console until the command "**End**" is received, on **each line** there will be an **order** of: Fish, Soup, Cake, Coffee or Tea. You will receive the order information separated by a **single space** in one of the following formats:

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

After the command "**End**" is received print the order in one of the following **formats**:

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

If there is **information** about **calories**, print:

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

Examples

Input	Output

Coffee Frappe 1.3
Tea IceTea 1.50 200
Soup Chicken 4.50 250
End
	Your order contains:
	  Quantity of liquids: 200
	  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

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

12. Animals

Note: in this problem you should define **virtual** method in the base class and **override** it in the derived classes. Learn more at https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/virtual.

Create a hierarchy of **Animals**. Your program should have three different animals – **Dog**, **Frog** and **Cat**. Deeper in the hierarchy you should have two additional classes – **`Kitten`** and **`Tomcat`**. **Kittens are female and Tomcats are male.** All types of animals should be able to produce some kind of sound – **`virtual ProduceSound()`**. For example, the

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.
Follow us:

dog should be able to bark. Your task is to model the hierarchy and test its functionality. Create an animal of each kind and make them all produce sound.

You will be given some lines of input. Each two lines will represent an animal. On the first line will be the type of animal and on the second – the name, the age and the gender. When the command **"Beast!"** is given, stop the input and print all the animals in the format shown below.

Output

- Print the information for each animal on three lines. On the first line, print: **"{AnimalType}"**
- On the second line print: **"{Name} {Age} {Gender}"**
- On the third line print the sounds it produces: **"{ProduceSound()}"**

Constraints

- Each **Animal** should have a **name**, an **age** and a **gender**
- **All** input values should **not be blank** (e.g. name, age and so on…)
- If you receive an input for the **gender** of a **Tomcat** or a **Kitten**, ignore it but **create** the animal
- If the input is invalid for one of the properties, throw an exception with message: **"Invalid input!"**
- Each animal should have the functionality to **ProduceSound()** as method **override**.
- Here is the type of sound each animal should produce:
  - **Dog: "Woof!"**
  - **Cat: "Meow meow"**
  - **Frog: "Ribbit"**
  - **Kittens: "Meow"**
  - **Tomcat: "MEOW"**

Examples

Input	Output
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!

© SoftUni – https://softuni.org. Copyrighted document. Unauthorized copy, reproduction or use is not permitted.