# Lab: Encapsulation and Inheritance

Tasks for exercise in class and for homework to the course "Programming Advanced for QA" @ SoftUni.

Test your tasks in the Judge system: https://judge.softuni.org/Contests/4461/Encapsulation-Inheritance-Lab

## 1. Person Info

Make sure to use the **provided resources** for the following problems.

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

- **FirstName**: **string**
- **LastName**: **string**
- **Age**: **int**

Each property needs **proper validation**.

- **Name** must be at **least 3 symbols.**
- Age must **not** be **zero or negative.**

If some of the properties are **NOT valid** throw **ArgumentExeption** with the following **messages**:

- "**Age cannot be zero or a negative integer!**"
- "**First name cannot contain fewer than 3 symbols!**"
- "**Last name cannot contain fewer than 3 symbols!**"

Next add a **method**:

**ToString()**: **string** – **override**

Here is an **example** of how the **string** should look like: "Tomas Anderson is 20 years old.".

**Hint**: Because of the **private setters** you will need a **constructor** with **3 parameters**.

## Examples

| Input | Output |
|---|---|
| 5<br>Brandi Anderson 65<br>Andrew Williams 57<br>Newton Holland 27<br>Andrew Clark 44<br>Brandi Scott 35 | Andrew Clark is 44 years old.<br>Andrew Williams is 57 years old.<br>Brandi Scott is 35 years old.<br>Brandi Anderson is 65 years old.<br>Newton Holland is 27 years old. |

## 2. Box Data

Create a class **Box**, with the following properties:

**Length** – **double**, should **not be zero** or **negative number**.

**Width** – **double**, should **not be zero** or **negative number**.

**Height** – **double**, should **not be zero** or **negative number**.

If one of the properties **IS a zero or negative number** throw an **ArgumentException** with the message:

"**{propertyName} cannot be zero or negative.**"

All **properties** are **set by the constructor** and when **set**, they **cannot** be **modified**.

Finally implement the **following methods**:

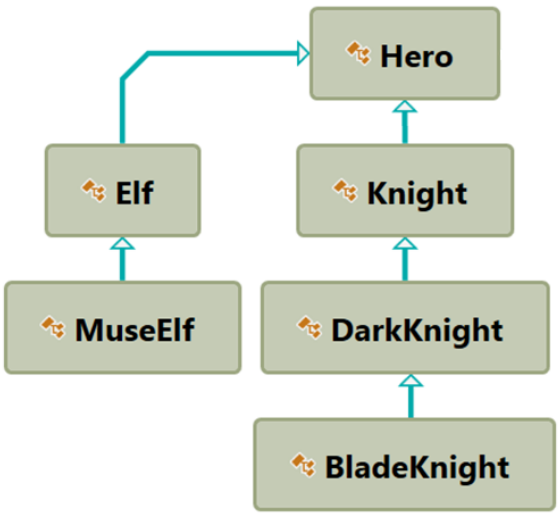
- **`double SurfaceArea()`**: Calculate and return the **surface area** of the **Box**. (2*L*W + 2*L*H + 2*W*H)
- **`double Volume()`**: Calculate and return the **volume** of the **Box**. (L*W*H)
- **`string ToString()`**:
  - "**Surface Area – {area}**"
  - "**Volume – {volume}**"

## Examples

| Input | Output |
|---|---|
| 2<br>3<br>4 | Surface Area - 52.00<br>Volume - 24.00 |
| 1.3<br>1<br>6 | Surface Area - 30.20<br>Volume - 7.80 |
| 2<br>-3<br>4 | Width cannot be zero or negative. |

# 3. 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`**

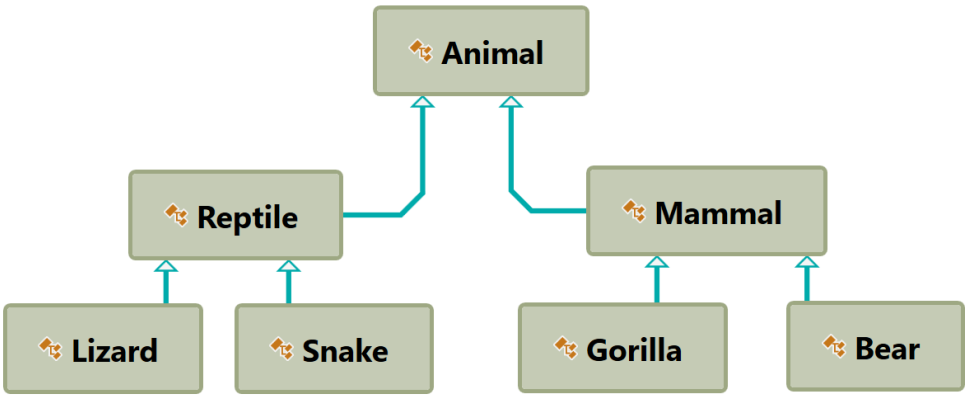
    - **level – int**
- The following properties:
    - **Username - string**
    - **Level – int**
- **ToString()** method

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}";
}
```

# 4. Zoo

Create a class hierarchy **Zoo**. It needs to contain the following structure:




Follow the diagram and create all 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**.