# Exercise: Inheritance and Abstraction

This document defines an in-class exercise from the "OOP" Course @ Software University.

## Problem 1. Book Shop

Our program will work with the following classes:

- **Book** - represents a book that holds **title**, **author** and **price**. Validate that the title and author are not null. The price should never be a negative number. A book should offer **information** about itself in the format shown in the output below.
- **GoldenEditionBook** - represents a special book holds the same properties as any **Book**, but its **price** is always **30% higher**.

| Sample Code | Output |
|---|---|
| `static void Main()`<br>`{`<br>`  Book book = new Book("Pod Igoto", "Ivan Vazov", 15.90);`<br>`  Console.WriteLine(book);`<br><br>`  GoldenEditionBook goldenBook =`<br>`  new GoldenEditionBook("Tutun", "Dimitar Dimov", 22.90);`<br>`  Console.WriteLine(goldenBook);`<br>`}` | `-Type: Book`<br>`-Title: Pod Igoto`<br>`-Author: Ivan Vazov`<br>`-Price: 15.90`<br><br>`-Type: GoldenEditionBook`<br>`-Title: Tutun`<br>`-Author: Dimitar Dimov`<br>`-Price: 29.77` |

### Step 1 - Create a Book Class

Create a new empty class a name it **Book**. Set its access modifier to **public** so it can be instantiated from any project.

```
namespace BookShop
{
    public class Book
    {
        // TODO: Add constructor

        // TODO: Add properties with validation

        // TODO: Override ToString()
    }
}
```

### Step 2 - Define the Properties of a Book

Define the **Title**, **Author** and **Price** properties of a Book. Ensure that they can only be **changed by the class itself or its descendants** (pick the most appropriate access modifier).

```
// TODO: Fix setter access modfiers
public string Title { get; set; }

public string Author { get; set; }

public double Price { get; set; }
```

### Step 3 - Define a Constructor

Define a constructor that accepts **title**, **author** and **price** arguments.

```
public Book(string title, string author, double price)
{
    // TODO: Set the values to the corresponding properties
}
```

## Step 4 - Perform Validations

Create a **field** for each property (**Price**, **Title** and **Author**) and **perform validations** for each one. The **getter should return the corresponding field** and the **setter should validate** the input data before setting it. Do this for every property.

```
private double price;

public double Price
{
    get
    {
        return this.price;
    }
    protected set
    {
        if (value < 0)
        {
            // TODO: Throw appropriate exception with descriptive message
        }

        // TODO: Set value to field
    }
}
```

## Step 5 - Override `ToString()`

As you probably already know, all classes in C# inherit the **`System.Object`** class and therefore have all its **public** members (**`ToString()`**, **`Equals()`** and **`GetHashCode()`** methods). **`ToString()`** serves to return information about an instance as string and is declared **virtual** (therefore can be changed in descendant classes). Let's **override** (change) its behavior for our **Book** class.

```
public override string ToString()
{
    StringBuilder output = new StringBuilder();
    output.AppendFormat("-Type: {0}{1}", this.GetType().Name, Environment.NewLine);
    // TODO: Append the other information

    return output.ToString();
}
```

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

## Step 6 - Create a GoldenEditionBook

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

```
namespace BookShop
{
    public class GoldenEditionBook : Book
    {
        public GoldenEditionBook(string title, string author, double price)
            // TODO: Reuse base constructor
        {
        }
    }
}
```

There is **no need** to rewrite the Price, Title and Author properties since **GoldenEditionBook** inherits **Book** and by default has them.

## Step 7 - Override the Price Property

Golden edition books should return a **30%** higher **price** than the original price. In order for the getter to return a different value, we need to override the Price property. To do that, go back to the **Book** class and declare it **virtual**.

```
public virtual double Price
{
```

**Virtual properties/methods** can be **overridden** in descendant classes.

Back to the **GoldenEditionBook** class, let's override the Price property and change the getter body.

```
public override double Price
{
    get
    {
        // TODO: Return increased price
    }
}
```