Problem 1 For each of the following three snippet of code, indicate what would be displayed.

```
void Myst1(int n)
{
    Console.WriteLine(n);
    if (n > 0) { Myst1(n - 1); }
}
Myst1(3);
```

```
void Myst2(int n)
{
    if (n != 0)
    {
        Console.Write($"{n} ");
        Myst2(n-2);
    }
}
Myst2(5);
```

```
int Myst3(int n)
{
    if (n != 0) return n + Myst3(n - 1);
    else return n;
}
Console.WriteLine(Myst3(4));
```

Problem 2 Write a program that

1. Asks the user to enter a path,
2. Checks if there is a file at this path:
   (a) If there is a file, display its content at the screen.
   (b) If there is no file, ask the user to enter text. When the user enters exactly "!DONE!", on a single line, without the quotes, write the text entered before !DONE! into a file located at the given path.

Your program should be able to handle graciously possible issues (such as an invalid path). Below are two examples of execution, taking place one after the other.

Example execution #1

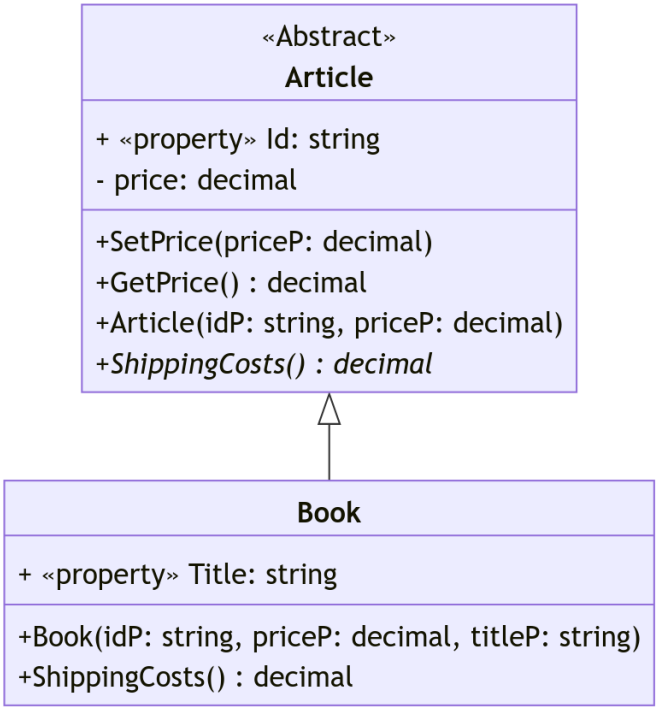
```
Enter a path
/home/user/CSCI_1302/final/test.txt←
Now creating a file at /home/user/CSCI_1302/final/test.txt.
Enter your text, one line at a time. When done, type "!DONE!" (without the
quotes), then enter.
This is a test←
spanning over←
three lines.←
!DONE!←
File correctly written.
```

Example execution #2

This execution takes place *after* the program was executed as in example execution #1 above.

```
Enter a path
/home/user/CSCI_1302/final/test.txt←
Now displaying file at /home/user/CSCI_1302/final/test.txt.
This is a test
spanning over
three lines.
Done displaying file.
```

Problem 3 Consider the following diagram:



1. Write a (partial) implementation of the `Article` abstract class:

   (a) Write an implementation for the `price` attribute: you can either use a getter and a setter (as pictured in the UML diagram), or a property. However, in both cases, setting the price to a negative value should result in an `ArgumentOutOfRangeException` (that you can shorten to AOORE) exception being thrown.

   (b) Write an *abstract* `ShippingCosts()` method.

2. Now, assume given a complete implementation of the `Article` abstract class. Write a **complete** implementation of the `Book` class (header included), containing:
   (a) An implementation of the `Title` property using auto-properties.
   (b) A `Book` constructor that passes the `idP` and `priceP` arguments to the `Article` constructor. The `titleP` argument should be assigned to the `Title` property.
   (c) A `ShippingCosts()` method that returns either 5.0, or 10% of the Book's price, whichever is smallest.

3. Write statements that, if placed in a `Main` method, would
   (a) Create a `Book` with `Id` "AAA001", `price` $12.5, titled "What it's like to be a bird".
   (b) Display (nicely) its shipping costs.
   (c) Display its `Id` (as retrieved from the object).