Exercise 6

1. **Interfaces and abstract classes**

Interfaces:

Read: https://www.tutorialspoint.com/csharp/csharp_interfaces.htm

- An interface contains definitions for a group of related functionalities that a class or can implement.
- Interface name starts with I
- Interface declaration is like class declaration (with keyword interface)
- Interface methods do not have content, they just have signature, return type and name
- Interfaces help to structure the code
- Using of interface: after class name add ':' and interface name
- Possible real-life using scenario: different parts of the system are developed by different teams/companies and interfaces help to structure the code.

Example:

```
interface ICar //definition: interface InterfaceName, starting with I
{
    void Drive(); //all classes that implementt his interface must have method
Drive()
}

class Car : ICar //we define that class car implements interface ICar
{
    public void Drive() //creating the method Drive() that is needed by the interface
    {
        Console.WriteLine("im driving");
    }
}
```

Abstract class:

Read: http://csharp.net-tutorials.com/classes/abstract-classes/

- Abstract class contains a base for a class.
- An abstract class cannot be instantiated.
- For example:

You can make a generic abstract base account called "Account", this holds basic information such as customer details.

You can then create two derived classes called "SavingAccount" or "DebitAccount" which can have their own specific behaviour whilst benefiting from the base class behaviour.

This is a situation where the customer must have either a Savings Account or a Debit Account, a generic "Account" is not allowed as it is not very popular in the real world to have just an account of no description.

Example:

```
abstract class BaseClass   // Abstract class
{
    protected int _x = 100;
    protected int _y = 150;
    public abstract void AbstractMethod();   // Abstract method
    public abstract int X    { get; }
    public abstract int Y    { get; }
}

class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X   // overriding property
    {
        get
        {
            return _x + 10;
        }
    }

    public override int Y   // overriding property
    {
        get
        {
            return _y + 10;
        }
    }

    static void Main()
    {
        DerivedClass o = new DerivedClass();
        o.AbstractMethod();
        Console.WriteLine("x = {0}, y = {1}", o.X, o.Y);
    }
}
// Output: x = 111, y = 161
```

- Interfaces cannot contain fields and properties, abstract classes can.

Exercise 1:

Create an interface representing an animal with 2 methods:

- Travel, with parameter string and no return type
- Eat, with parameter string and no return type

Create 2 classes implementing this interface. Methods only print out class name and activity. Method parameters indicate the destination or food. For example: "Dog eats meat", "Rabbit eats carrot".

- Class Rabbit
- Class Dog

In main method create objects of both classes and call out the methods.

2. **Polymorphism and class inheritance**
   Read: https://www.tutorialspoint.com/csharp/csharp_polymorphism.htm
   Example of class inheritance: read more: http://csharp-station.com/Tutorial/CSharp/Lesson08
   Example of method overwriting: http://csharp-station.com/Tutorial/CSharp/Lesson09

Sometimes objects are very similar but have slight differences. In this case it is not useful to duplicate code. To solve it we can use class inhertiance: we can take base methods from one class and overwrite them if we need to.

For example bank card and credit card are very similar, the main difference is in the logic of making the payment.

- Virtual method: a method that can be overwritten.
- Child classes use the constructors and other content from their parent.

Example of method overriding:

```
public class ParentClass
{
    public virtual void print() // method that can be overwritten
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class ChildClass : ParentClass // extending class
{
    public static void Main() // main method
    {
        ChildClass child = new ChildClass();
        child.print();
    }

    public override void print()
    {
        base.print();// we say to use content from base class print method
        Console.WriteLine("Printing from child class"); // we add our custom code
    }
}
```

Exercise 2: copy-paste this code to your solution and see what is the output:

```
public class ParentClass
{
    public ParentClass()
    {
        Console.WriteLine("Parent Constructor.");
    }

    public void print()
    {
        Console.WriteLine("I'm a Parent Class.");
    }
}

public class ChildClass : ParentClass
{
    public ChildClass()
    {
        Console.WriteLine("Child Constructor.");
    }

}

public static void Main()
{
    ChildClass child = new ChildClass();

    child.print();
}
```

Exercise 3:

Copy this code to your code:

```
class BaseClass
{
    public void Method1()
    {
        Console.WriteLine("Base - Method1");
    }
}

class DerivedClass : BaseClass
{
    public void Method2()
    {
        Console.WriteLine("Derived - Method2");
    }
}
```

- In the Main method, declare variables bc, dc, and bcdc.
    - bc is of type BaseClass, and its value is of type BaseClass.
    - dc is of type DerivedClass, and its value is of type DerivedClass.

    - bcdc is of type BaseClass, and its value is of type DerivedClass. This is the variable to pay attention to.
- Try calling out Method1 and Method2 from different objects.
- Next, add the following Method2 method to BaseClass. The signature of this method matches the signature of the Method2 method in DerivedClass.
- Try calling out Method1 and Method2 from different objects, see what changed.
- In base class change `public void Method2() to public virtual void Method2()`
- In derived class change „`public void Method2()`“ to „`public override void Method2()`”
- Try calling out Method1 and Method2 from different objects, see what changed.

Exercise 4:

- Write an interface for Shape with parameterless method Draw
- Write a base class Shape with overwriteable method Draw that prints out “I am a shape”
- Write 3 subclasses that extend the base class:
    - Circle, overwriting the method “Draw” and printing “I am a circle”
    - Square, overwriting the method “Draw” and printing “I am a square”
    - Triangle, overwriting the method “Draw” and printing “I am a triangle”

- In main method create new instances of the subclasses. Add them to array and call out Draw methods.
- Change the methods in a way that they also print out content from the base class.
- Add a method for setting the color for all the shapes. Add a property called ‘_color’. The method takes the color value as parameter and sets the value to _color. It also prints out info “color was set to ..”. Can we add this method to interface? In what class do we have to define this method?
- Add a method for setting the height of the shapes. Can we add this method to interface? In what class do we have to define this method?
- Add method for calculating the area for all derived classes. Think of parameters. Can we add this method to interface?

Exercise 5: BankCard

Create an interface for BankCard.

The interface should define following methods:

- Method for getting account balance; method prints out the account balance
- Method for setting the account balance; method takes the value as parameter and sets it
- Method for making payment; takes the amount for payment as a parameter and changes account balance accordingly
- Method for printing out card info

- Method for closing the card
- Method re-opening the card

Create classes DebitCard and CreditCard

- Think which class should be the base class and which class should be derived class
- Create body for the methods described in the interface and neccessary properties
- Payments can only be made and balance changed when the card status is open
- Create a property for credit card which stores the credit limit (the maximum negative balance the card can have).
- When making a payment:
    - With debit card the upper limit for payment is account balance.
    - With credit card the amount can also be negative (limit is the credit limit)
- Create a constructor that sets the initial balance as 0 and takes card type (string) as parameter. The constructor also sets the card status as opened for debit card and closed for credit card.
- Create additional constructor for credit card which takes card type and credit limit as parameter
- When printing card info:
    - Debit card: Card type, account balance, status (Only If card is closed)
    - Credit card prints the same info as debit card and also the credit limit

Create both card types in main method and test them!