

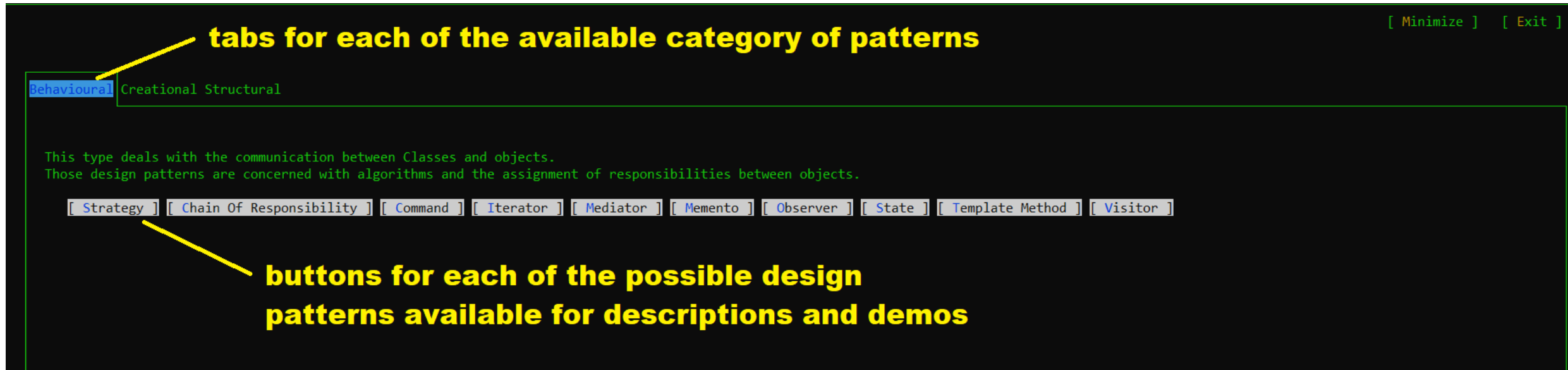
Design Patterns RESUME

User and Developer Manual

Console Project – application overview

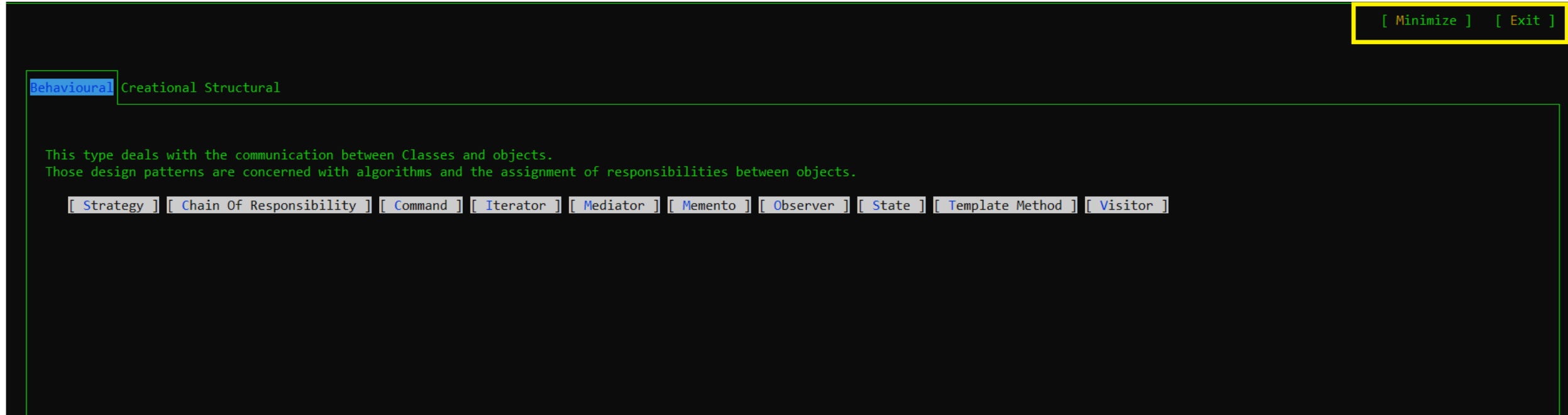
The application is simply divided in 3 tab sections in which the main design patterns categories are presented.

For each category there are the list (buttons) of available design patterns



Console Project – fixed buttons

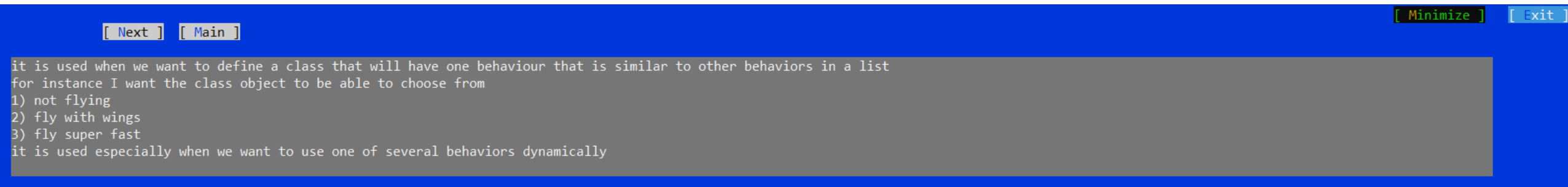
Exit and minimize buttons are always available in the context: you can minimize the application (which is set to full screen) or exit from it at anytime



Console Project – page description

For each of the available pattern there are 2 types of pages. The first one is the description page, in which only a main description is shown.

The description page has the following view (blue screen and gray characters):

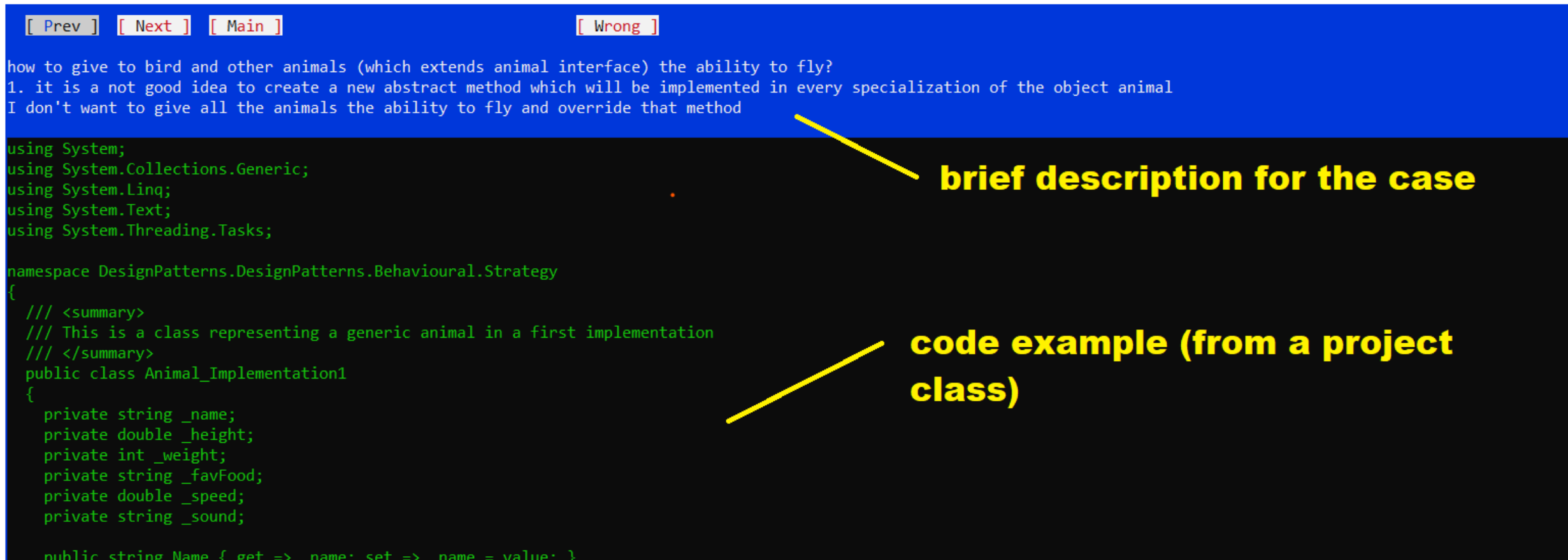


```
[ Next ] [ Main ] [ Minimize ] [ Exit ]  
it is used when we want to define a class that will have one behaviour that is similar to other behaviors in a list  
for instance I want the class object to be able to choose from  
1) not flying  
2) fly with wings  
3) fly super fast  
it is used especially when we want to use one of several behaviors dynamically
```

Console Project – example description

The second available page is a page of example. In this context a brief description of the example is given and a second view renders the class of the example at hand.

The example page has the following characteristics:



```
[ Prev ] [ Next ] [ Main ] [ Wrong ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
    /// <summary>
    /// This is a class representing a generic animal in a first implementation
    /// </summary>
    public class Animal_Implementation1
    {
        private string _name;
        private double _height;
        private int _weight;
        private string _favFood;
        private double _speed;
        private string _sound;

        public string Name { get => _name; set => _name = value; }
    }
}
```

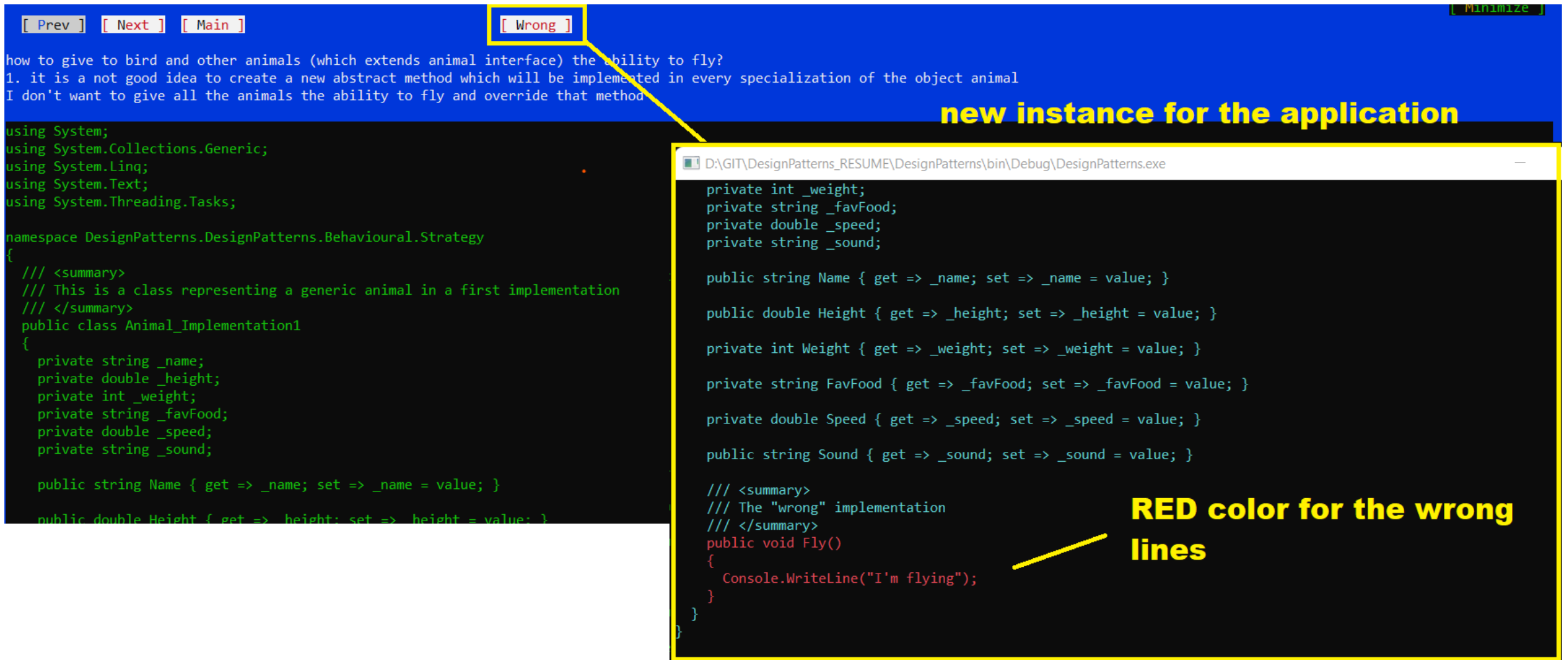
brief description for the case

code example (from a project class)

Console Project – wrong button

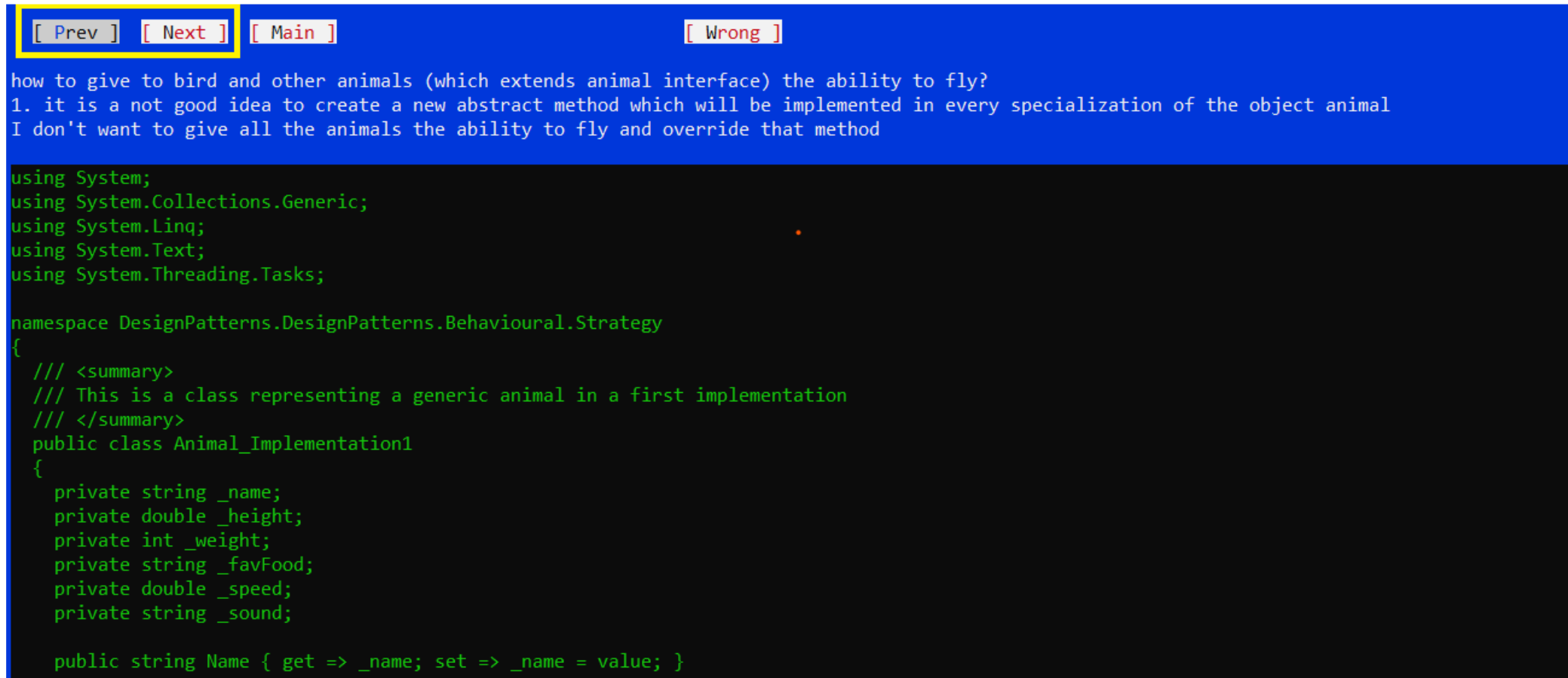
When the current example has a counterexample, it is often showed clicking on the wrong button.

Another instance of the application will start showed a console with the wrong class case:



Console Project – prev – next buttons

For moving forward and behind in the demo, the previous and next page buttons are rendered in the page (previous on left and next on right). Sometimes they change a bit when passing from a description to an example page and viceversa:



The screenshot shows a console application window with a blue title bar. At the top, there are four buttons: "[Prev]", "[Next]", "[Main]", and "[Wrong]". The "[Prev]" and "[Next]" buttons are highlighted with a yellow rectangle. Below the buttons, the text "how to give to bird and other animals (which extends animal interface) the ability to fly?" is displayed. This is followed by a numbered list item: "1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal". Below the list, the text "I don't want to give all the animals the ability to fly and override that method" is shown. The main area of the console displays C# code. The code starts with several using statements: "using System;", "using System.Collections.Generic;", "using System.Linq;", "using System.Text;", and "using System.Threading.Tasks;". This is followed by a namespace declaration: "namespace DesignPatterns.DesignPatterns.Behavioural.Strategy". The code then defines a class "Animal_Implementation1" with several private fields: "_name", "_height", "_weight", "_favFood", "_speed", and "_sound". Finally, there is a public property "Name" with get and set methods.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
    /// <summary>
    /// This is a class representing a generic animal in a first implementation
    /// </summary>
    public class Animal_Implementation1
    {
        private string _name;
        private double _height;
        private int _weight;
        private string _favFood;
        private double _speed;
        private string _sound;

        public string Name { get => _name; set => _name = value; }
    }
}
```

Console Project – Main button

For the presentation the main button is always available: with it, it is always possible to come back to the original context (the tabular page with all the alternatives available):

```
[ Prev ] [ Next ] [ Main ] [ Wrong ]

how to give to bird and other animals (which extends animal interface) the ability to fly?
1. it is a not good idea to create a new abstract method which will be implemented in every specialization of the object animal
I don't want to give all the animals the ability to fly and override that method

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

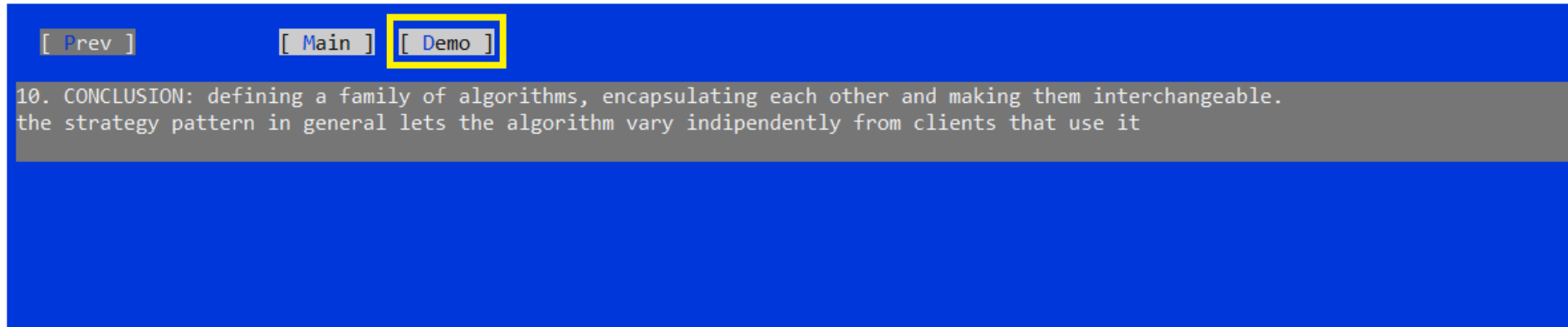
namespace DesignPatterns.DesignPatterns.Behavioural.Strategy
{
    /// <summary>
    /// This is a class representing a generic animal in a first implementation
    /// </summary>
    public class Animal_Implementation1
    {
        private string _name;
        private double _height;
        private int _weight;
        private string _favFood;
        private double _speed;
        private string _sound;

        public string Name { get => _name; set => _name = value; }

        public double Height { get => _height; set => _height = value; }
```


Console Project – demo button

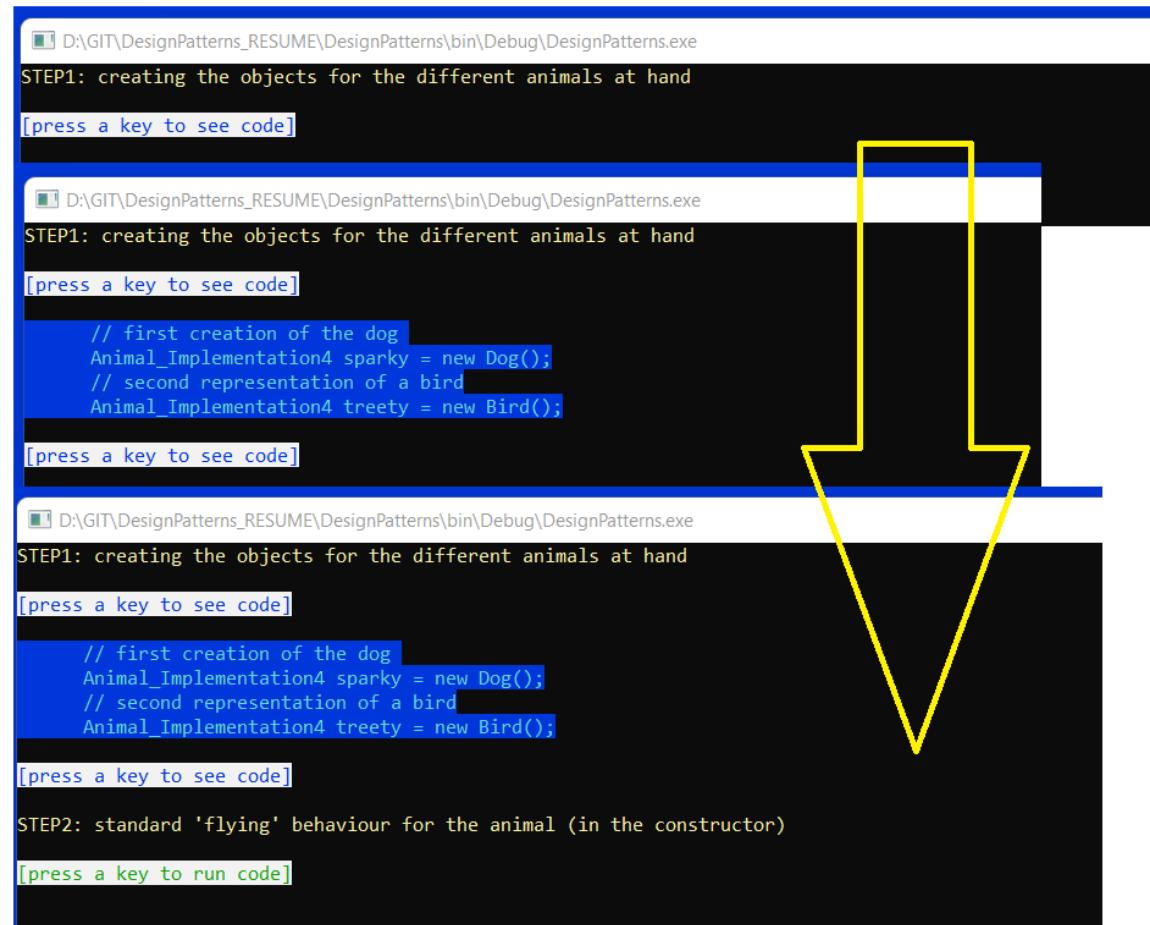
When concluding the presentation part, it will be possible seeing the demo too. This is available on the last page of each of the presented patterns:



Console Project – demo

Demo is contained in another instance of the application: it could vary depending on the selected pattern.

Usually a DEMO is made such that it is stopped at each line or block of lines of relevant code, given precise information to the user concerning what is happening:



```
D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe
STEP1: creating the objects for the different animals at hand
[press a key to see code]

D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe
STEP1: creating the objects for the different animals at hand
[press a key to see code]

// first creation of the dog
Animal_Implementation4 sparky = new Dog();
// second representation of a bird
Animal_Implementation4 treety = new Bird();

[press a key to see code]

D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe
STEP1: creating the objects for the different animals at hand
[press a key to see code]

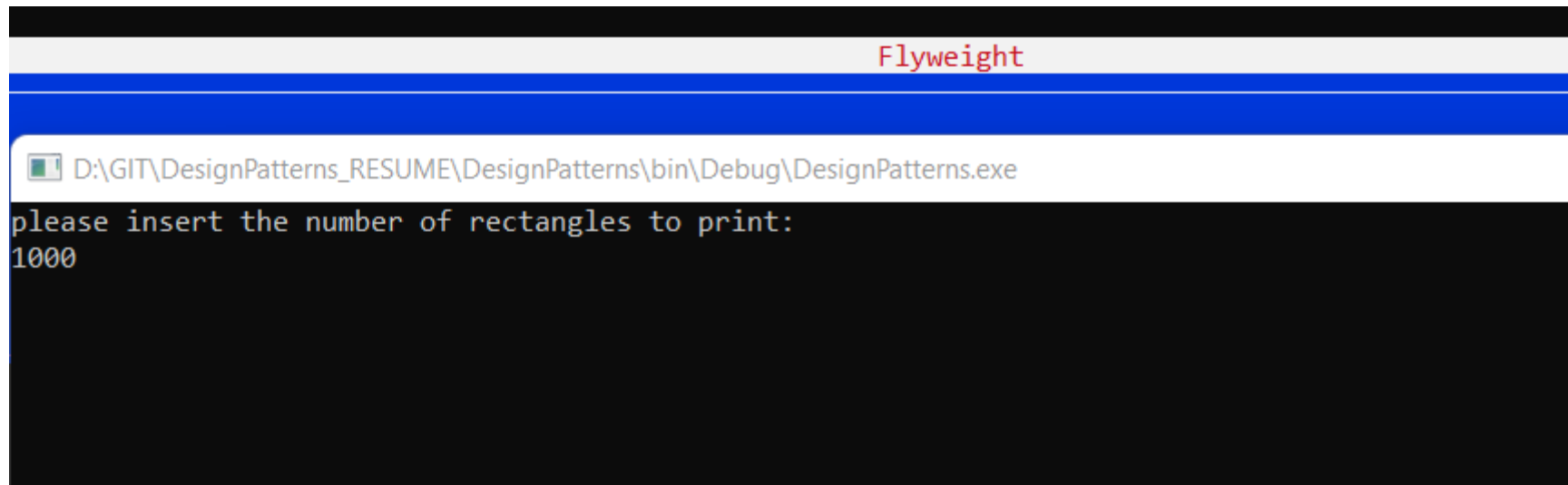
// first creation of the dog
Animal_Implementation4 sparky = new Dog();
// second representation of a bird
Animal_Implementation4 treety = new Bird();

[press a key to see code]

STEP2: standard 'flying' behaviour for the animal (in the constructor)
[press a key to run code]
```

Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user:



Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user:

D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe

writing 1000 random rectangles without optimized structure... this might takes sometime

employed time: 00:00:00.2241191
press ENTER to continue

Console Project – demo

But sometimes it could be also represented by a different application with a different logic. In any case instructions are presented to the user:



```
D:\GIT\DesignPatterns_RESUME\DesignPatterns\bin\Debug\DesignPatterns.exe
writing 1000 random rectangles with optimized structure... this might takes sometime

employed time: 00:00:00.2060738
the test ended well... the elapsed time for the optimized case is less than the first one! (difference 00:00:00.0180453)
```