

## 44. What is Inversion of Control?

**Brief summary:** Inversion of Control is the design approach according to which the control flow of a program is inverted: instead of the programmer controlling the flow of a program, the external sources (framework, services, other components) take control of it.

Inversion of Control is the design approach according to which the control flow of a program is inverted: instead of the programmer controlling the flow of a program, the external sources (framework, services, other components) take control of it.

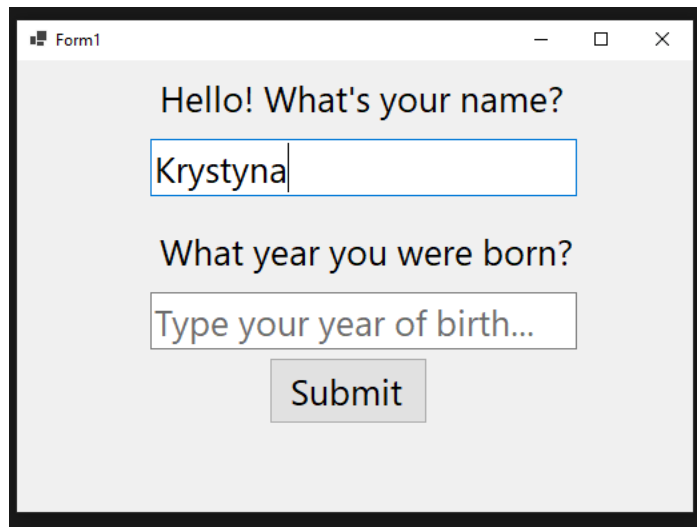
Let's consider two simple examples. First is a simple console application interacting with the user.

```
Hello! What's your name?  
Krystyna  
What year you were born?  
_
```

In this program, the code is in control - it decides when the user answers the questions shown in the console. Here is the implementation of this program:

```
Console.WriteLine("Hello! What's your name?");  
var name = Console.ReadLine();  
int yearOfBirth;  
do  
{  
    Console.WriteLine("What year you were born?");  
}  
while (!int.TryParse(Console.ReadLine(), out yearOfBirth));  
  
Console.WriteLine(  
    $"Hello {name}, you are " +  
    $"{DateTime.Now.Year - yearOfBirth} years old!");  
  
Console.ReadKey();
```

Now, let's see a different approach:



In this application, the code doesn't control when exactly the user will fill in the form, and when the final message will be printed. The action of the user (clicking on the Submit button) will trigger an event that will handle printing the output:

```
private void SubmitButton_Click(object sender, EventArgs e)
{
    SetBackColor(NameTextBox);
    SetBackColor(YearOfBirthTextBox);

    if(!string.IsNullOrEmpty(NameTextBox.Text) &&
        !string.IsNullOrEmpty(YearOfBirthTextBox.Text))
    {
        int age = DateTime.Now.Year - int.Parse(
            YearOfBirthTextBox.Text);
        ResultTextBox.Text = $"Hello {NameTextBox.Text}! You are
            ${age} years old!";
    }
}

2 references
private void SetBackColor(TextBox textBox)
{
    if (string.IsNullOrEmpty(textBox.Text))
    {
        textBox.BackColor = Color.IndianRed;
    }
}
```

The control flow of the program is **inverted** compared to the “traditional” flow, where the code decides when exactly some action happens. Here, the framework (in this case, Windows Forms) is in charge, and it executes particular pieces of code based on the user actions that trigger events.

Inversion of Control is sometimes referred to as “the Hollywood Principle” which says “don’t call us, we will call you”. In this case, we don’t call a method. The framework calls us, letting us know via an event that some code needs to be executed.

According to Martin Fowler (author of the great book “Refactoring” and in general authority in topics of software development, design, etc.) the Inversion of Control is what makes the **difference between a framework and library**:

*“A **library** is essentially a set of functions that you can call, these days usually organized into classes. Each call does some work and returns control to the client.*

*A **framework** embodies some abstract design, with more behavior built in. In order to use it, you need to insert your behavior into various places in the framework either by subclassing or by plugging in your own classes. The framework’s code then calls your code at these points.”*

There are many ways in which the control can be inverted. In the example we’ve seen, this was implemented by using events. Events were triggered by the user’s actions on the GUI, thus executing some particular methods in code.

Speaking more generally, the Inversion of Control happens whenever some kind of a callback is defined. A callback is an executable code (a method in C#) that gets passed as an argument to some other code. Let’s consider this simple example:

```

ReadLineByLine(input, () => Console.WriteLine("Finished!"));
Console.ReadKey();

void ReadLineByLine(string input, Action onFinished)
{
    string[] lines = input.Split(Environment.NewLine);
    for (int i = 0; i < lines.Length; i++)
    {
        string line = lines[i];
        Console.WriteLine($"[Line number {i}] {line}");
    }
    onFinished();
}

```

The ReadLineByLine method uses a callback - an Action passed as a parameter. Once the entire input has been read, the callback will be executed. In real-life projects it often happens that after some data is read (from a database, API, or anything else that takes time to execute) a callback is invoked, informing some other piece of the code that it can start its work, as the data it requires is ready to be used.

Another example of Inversion of Control could be the **Template Method**. In the lecture about it, we mentioned the example of SetUp and TearDown methods from NUnit framework. It's another case when the framework calls the methods we defined. The template is defined in NUnit itself, where it is decided that first the SetUp must be called, then the actual test, and then the TearDown. But the actual implementation of those steps is defined by the programmer.

Dependency Injection is another example of Inversion of Control. The code that some class needs to execute is injected from the outside. We don't have control over what method exactly will be called. This decision is made for us by someone who provides the concrete type as the constructor parameter. We only declare that we need some dependency.

Using an interface is similar to having a callback. After all, an interface is like a bundle of methods. It would actually be possible to have Dependency Injection without interfaces, but by simply providing a class with Funcs that will be executed, similarly as the Action that we saw in an example above.

```

class PersonalDataFormatterWithFunc
{
    private readonly Func<IEnumerable<Person>> _readPeople;

    0 references
    public PersonalDataFormatterWithFunc(
        Func<IEnumerable<Person>> readPeople)
    {
        _readPeople = readPeople;
    }

    0 references
    public string Format()
    {
        var people = _readPeople();

        return string.Join("\n",
            people.Select(p => $"{p.Name} born in" +
                $"{p.Country} on {p.YearOfBirth}"));
    }
}

```

Let's summarize. Inversion of Control is the design approach according to which the control flow of a program is inverted: instead of the programmer controlling the flow of a program, the external sources (framework, services, other components) take control of it.

### Bonus questions:

- **"What is a callback?"**

*A callback is an executable code (a method in C#) that gets passed as an argument to some other code.*

- **"What is the difference between a framework and a library?"**

*According to Martin Fowler: "A **library** is essentially a set of functions that you can call, these days usually organized into classes. Each call does some work and returns control to the client. A **framework** embodies some abstract design, with more behavior built in. In order to use it, you need to insert your behavior into various places in the framework either by subclassing or by plugging in your own classes. The framework's code then calls your code at these points." So in short, the framework relies on Inversion of Control, but the library does not.*