Answer on Question#38521- Programming, C#

1. which feature of c# will allow SoftSols Inc.to reuse the existing application code? Describe in brief.

Solution.

Template Method:

Define the skeleton of an algorithm in an operation, deferring some steps to

Subclasses. Template Method lets subclasses redefine certain steps of an algorithm

without changing the Algorithm's Structure.

Usage:

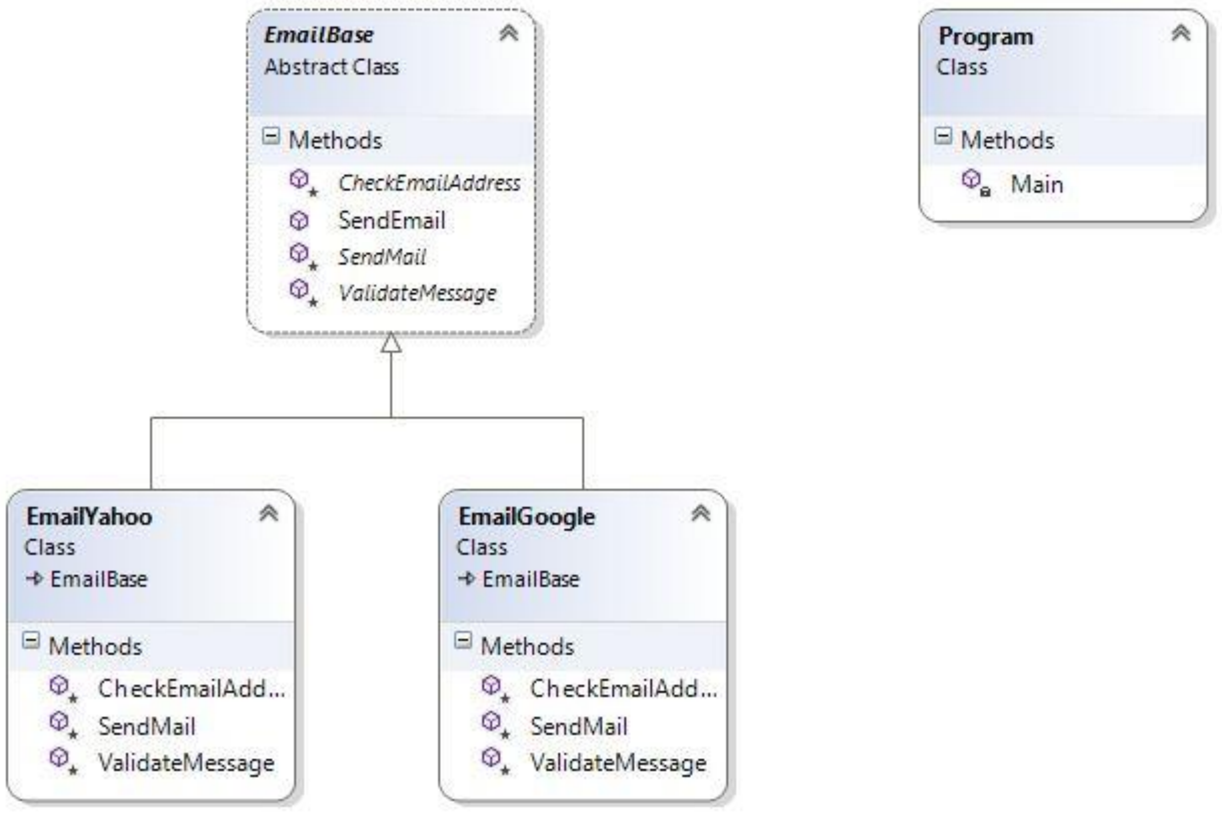
When you are certain about the High Level steps involved in an Algorithm/Work flow you can use the Template Pattern which allows the Base Class to define the Sequence of the Steps but permits the Sub classes to alter the implementation of any/all steps.

Example in the .Net framework:

The most common example is the Asp.Net Page Life Cycle. The Page Life Cycle has a few methods which are called in a sequence but we have the liberty to modify the functionality of any of the methods by overriding them.

Sample implementation of Template Method Pattern:

Let's see the class diagram first:



And here goes the code:

EmailBase.cs

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

namespace TemplateMethod
{
    public abstract class EmailBase
    {

        public bool SendEmail()
        {
            if (CheckEmailAddress() == true) // Method1 in the sequence
            {
                if (ValidateMessage() == true) // Method2 in the sequence
                {
                    if (SendMail() == true) // Method3 in the sequence
                    {
                        return true;
                    }
                    else
                    {
                        return false;
                    }

                }
                else
                {
                    return false;
                }

            }
            else
            {
                return false;

            }


        }

        protected abstract bool CheckEmailAddress();
        protected abstract bool ValidateMessage();
        protected abstract bool SendMail();


    }
}

```

EmailYahoo.cs

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

namespace TemplateMethod
{
    public class EmailYahoo:EmailBase
    {

        protected override bool CheckEmailAddress()
        {
            Console.WriteLine("Checking Email Address : YahooEmail");
            return true;
        }
        protected override bool ValidateMessage()
        {
            Console.WriteLine("Validating Email Message : YahooEmail");
            return true;
        }


        protected override bool SendMail()
        {
            Console.WriteLine("Semding Email : YahooEmail");
            return true;
        }


    }
}

```

EmailGoogle.cs

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

namespace TemplateMethod
{
    public class EmailGoogle:EmailBase
    {

        protected override bool CheckEmailAddress()
        {
            Console.WriteLine("Checking Email Address : GoogleEmail");
            return true;
        }
        protected override bool ValidateMessage()
        {
            Console.WriteLine("Validating Email Message : GoogleEmail");
            return true;
        }


        protected override bool SendMail()
        {
            Console.WriteLine("Semding Email : GoogleEmail");
            return true;
        }


    }
}

```

Program.cs

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

namespace TemplateMethod
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Please choose an Email Account to send an Email:");
            Console.WriteLine("Choose 1 for Google");
            Console.WriteLine("Choose 2 for Yahoo");
            string choice = Console.ReadLine();
            EmailBase email;

            if (choice == "1")
            {
                email = new EmailGoogle(); // Rather than newing it up here, you may use a factory to do so.
                email.SendEmail();

            }
            if (choice == "2")
            {
                email = new EmailYahoo(); // Rather than newing it up here, you may use a factory to do so.
                email.SendEmail();
            }
        }
    }
}

```

Final Words:

It's very obvious that why the Template Method Pattern is a popular pattern, everything at last revolves around Algorithms and if you are clear with the steps involved it makes real sense to delegate the duty of implementing the step's functionality to the sub classes.

We will now explain the Template Method Pattern. Like the Strategy Pattern, this pattern is part of the behavioral design patterns. As the name Template suggests, we know a template is a kind of layout available for some kind of process, that we can modify as needed and use it. The case here is similar. We have some sub-tasks to complete a big task, that we can use as per our situation requirements. So technically speaking, we make the sub-tasks as abstract and implement them, as per our needs, in different cases. But one thing remains common; the overall steps of the entire procedure or the algorithm, in other words first Step 1, then Step 2, then Step 3 and so on, that remain the same in their execution order.

Let's explain this concept in detail now.

What the Template Method Pattern is

According to the GoF's definition, this pattern:

Defines the skeleton of an algorithm in an operation, deferring some steps to sub-classes. The Template Method lets sub-classes redefine certain steps of an algorithm without changing the algorithm's structure.

A real-world example

An organization goes to colleges and hires beginner. They have a number of steps defined for the process of hiring in the following order:

Step 1: Take a test in the first round

Step 2: Group discussion in the second round
Step 3: Technical interview
Step 4: HR interview

They have a common process of group discussions and HR interviews for all the departments, but a different test and technical interview process, depending on whether it is for computer or electronics departments. So we create a hiring process template that can be used by the company for two different departments.

Let's convert the preceding real-world example into technical code.

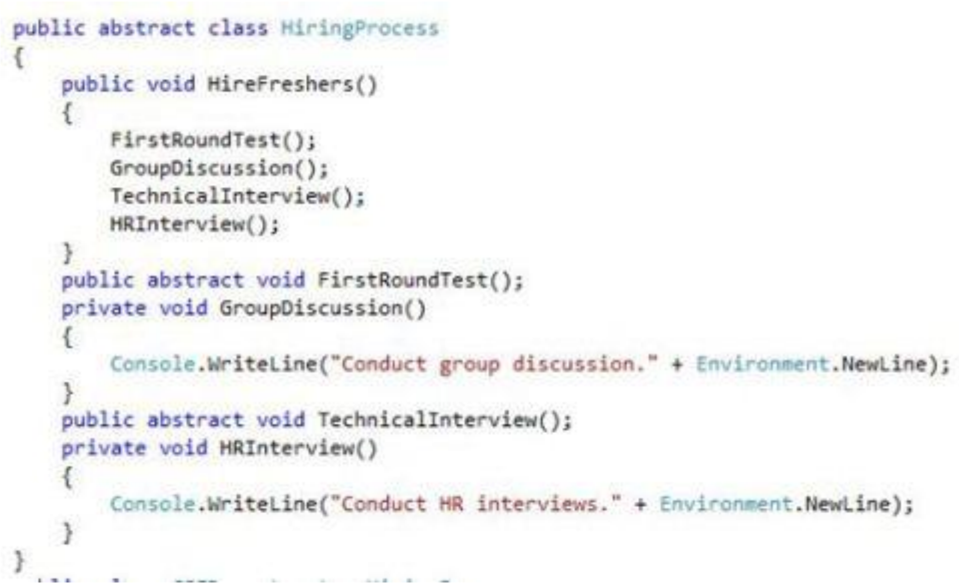
We will be dividing the system into the following components for our implementation of this pattern:

1. **Abstract Class:** This will be an abstract class, that will have the abstract methods of our technical and test rounds, that will be implemented in the sub-classes based on the departments and the non-abstract methods of the group discussion and the HR rounds.

   Most importantly, this class will contain a Non Abstract method that will be calling the abstract methods or any other method defined in this class, to outline the flow of the overall process in a step-wise call to these methods, or you can say, it will contain the sequence of an algorithm.

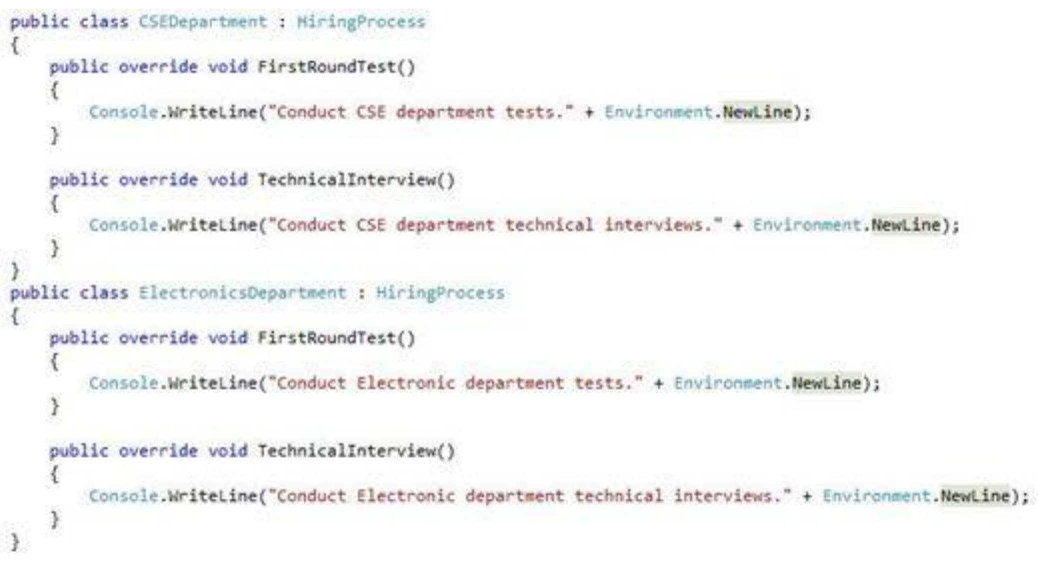
2. **Concrete Classes:** These will be concrete implementations of our abstract steps of the tests and technical interviews, depending on the departments.

So our initial code setup for defining the template of the hiring process will be like the following:



```
public abstract class HiringProcess
{
    public void HireFreshers()
    {
        FirstRoundTest();
        GroupDiscussion();
        TechnicalInterview();
        HRInterview();
    }
    public abstract void FirstRoundTest();
    private void GroupDiscussion()
    {
        Console.WriteLine("Conduct group discussion." + Environment.NewLine);
    }
    public abstract void TechnicalInterview();
    private void HRInterview()
    {
        Console.WriteLine("Conduct HR interviews." + Environment.NewLine);
    }
}
```

We will next be implementing the template, to define the test and technical interview rounds, based on the type of department as in the following:

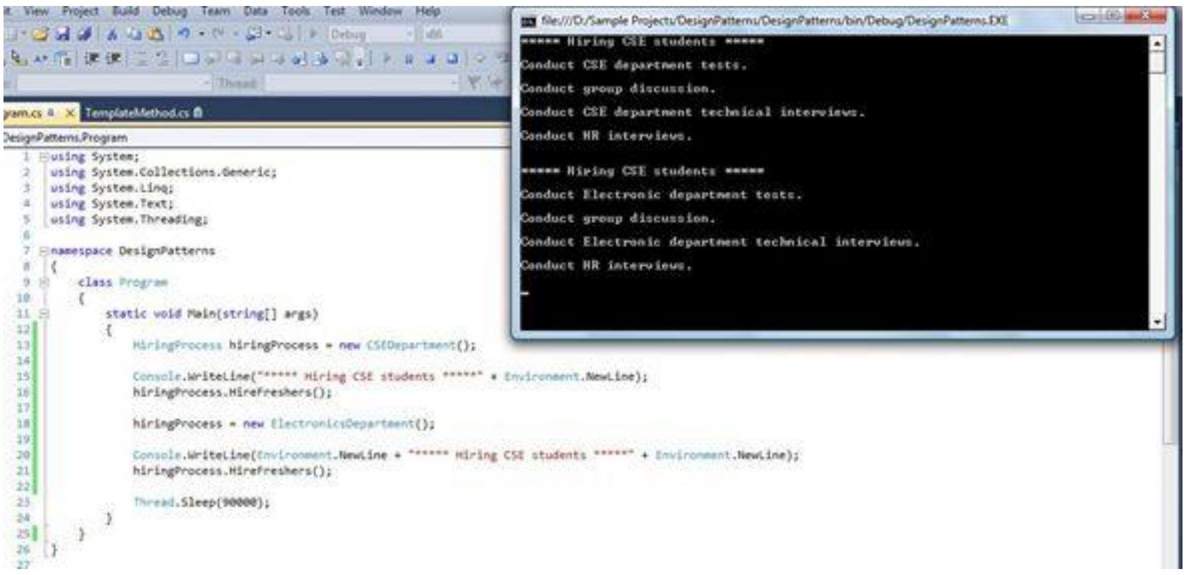


```
public class CSEDepartment : HiringProcess
{
    public override void FirstRoundTest()
    {
        Console.WriteLine("Conduct CSE department tests." + Environment.NewLine);
    }

    public override void TechnicalInterview()
    {
        Console.WriteLine("Conduct CSE department technical interviews." + Environment.NewLine);
    }
}
public class ElectronicsDepartment : HiringProcess
{
    public override void FirstRoundTest()
    {
        Console.WriteLine("Conduct Electronic department tests." + Environment.NewLine);
    }

    public override void TechnicalInterview()
    {
        Console.WriteLine("Conduct Electronic department technical interviews." + Environment.NewLine);
    }
}
```

So now we have set up the complete process of hiring for two different departments, with a few common steps. Now we just need to start with the interview process for both the departments, using the client code. So we begin the process with the following code implementation:



file:///D:/Sample Projects/DesignPatterns/DesignPatterns/bin/Debug/DesignPatterns.EXE

```
***** Hiring CSE students *****
Conduct CSE department tests.
Conduct group discussion.
Conduct CSE department technical interviews.
Conduct HR interviews.

***** Hiring CSE students *****
Conduct Electronic department tests.
Conduct group discussion.
Conduct Electronic department technical interviews.
Conduct HR interviews.
```

DesignPatterns.Program

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

namespace DesignPatterns
{
    class Program
    {
        static void Main(string[] args)
        {
            HiringProcess hiringProcess = new CSEDepartment();

            Console.WriteLine("***** Hiring CSE students *****" + Environment.NewLine);
            hiringProcess.HireFreshers();

            hiringProcess = new ElectronicsDepartment();

            Console.WriteLine(Environment.NewLine + "***** Hiring CSE students *****" + Environment.NewLine);
            hiringProcess.HireFreshers();

            Thread.Sleep(90000);
        }
    }
}
```

So this was all about the Template Method Pattern. For more design patterns, you can check out here.