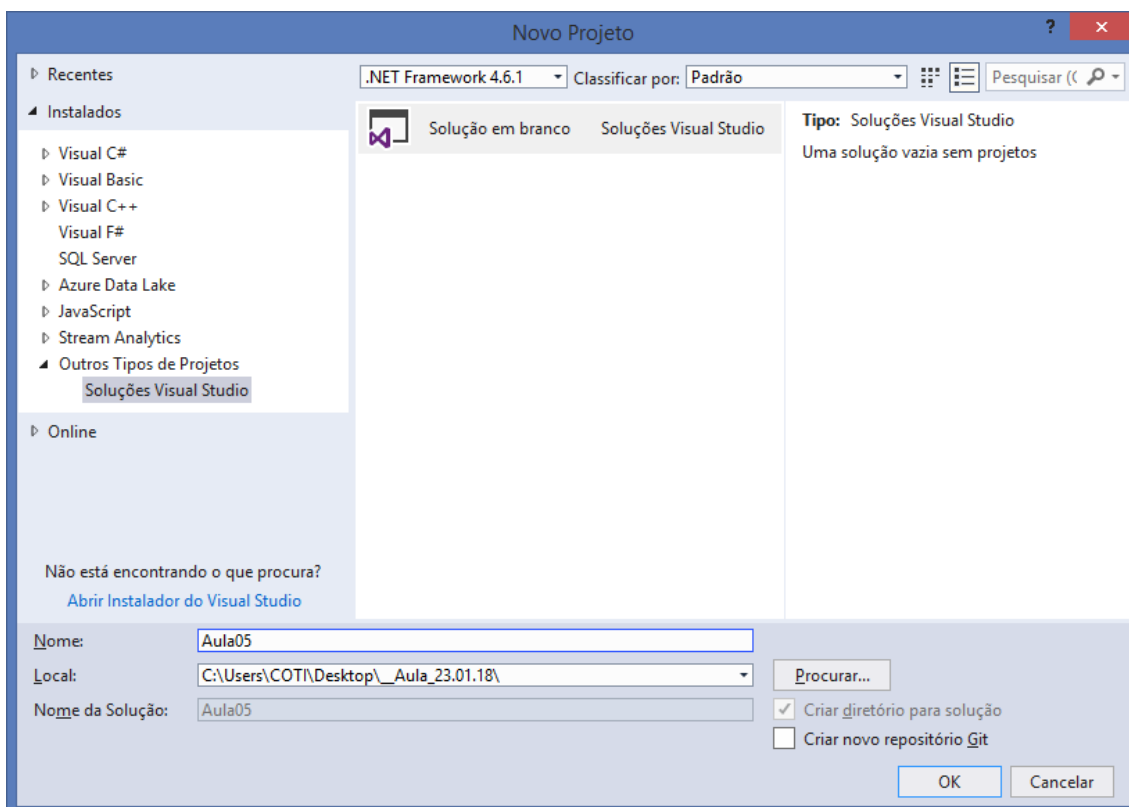
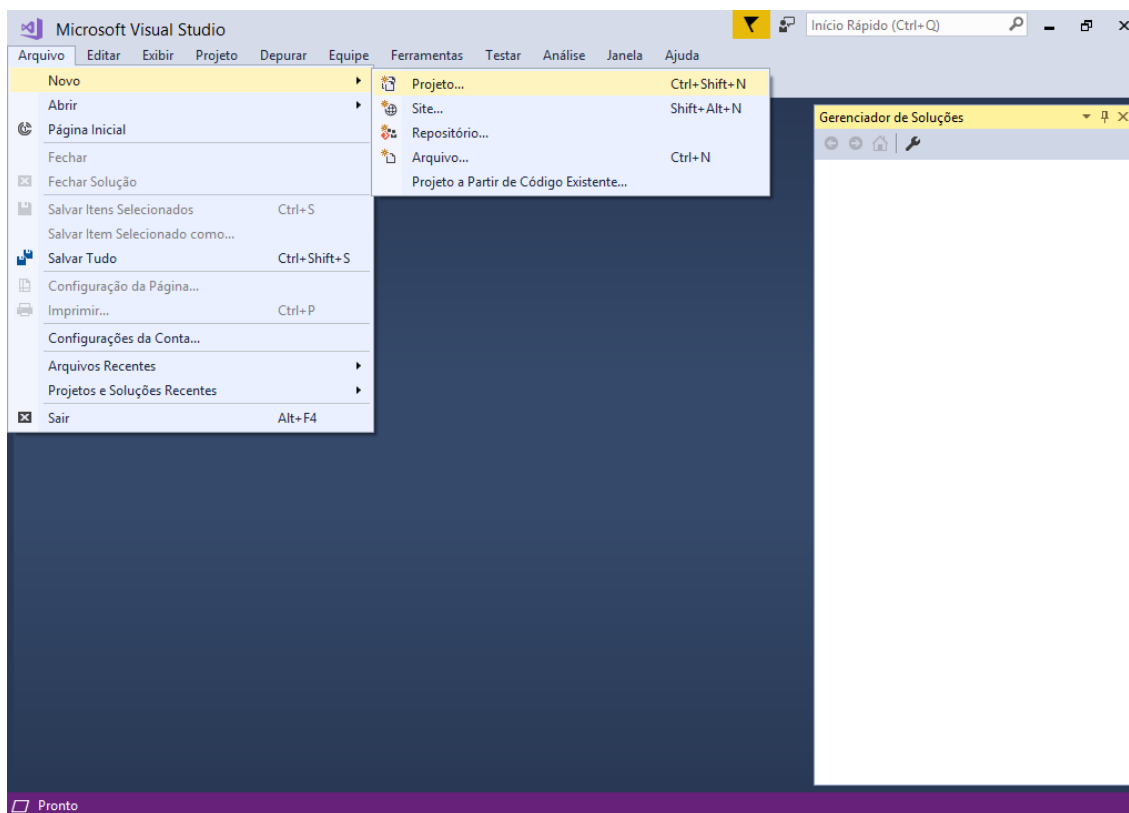
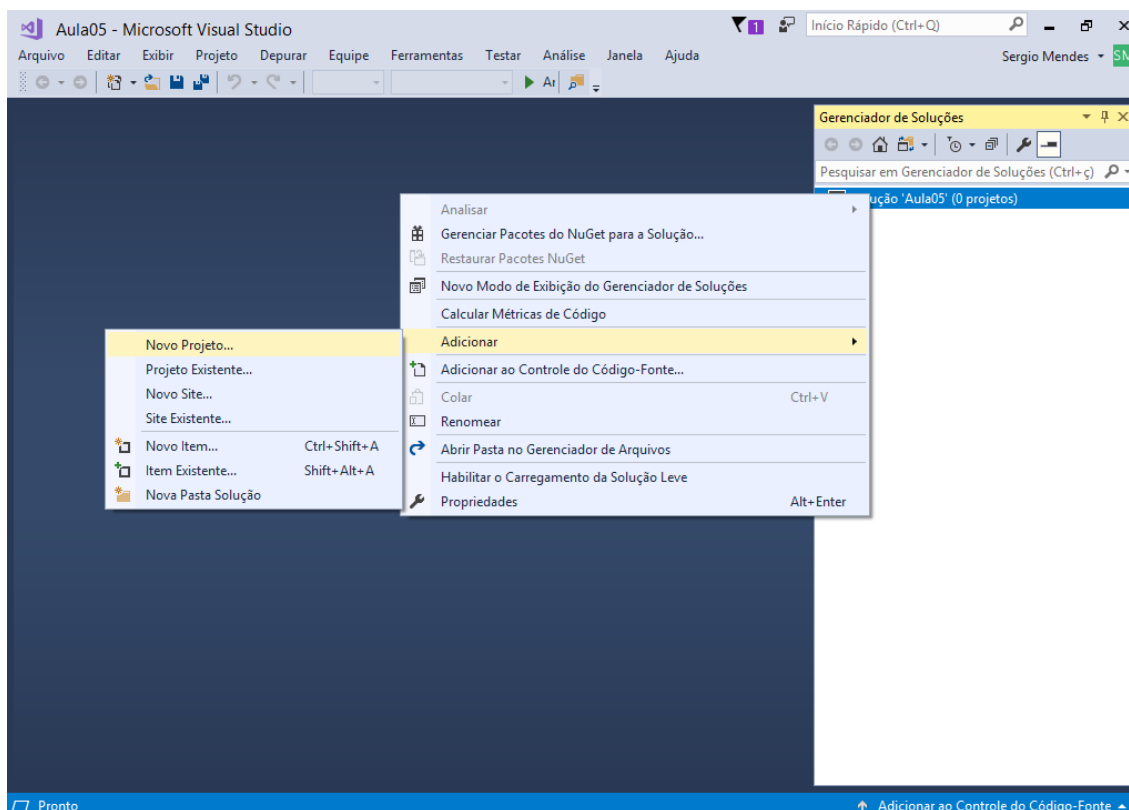


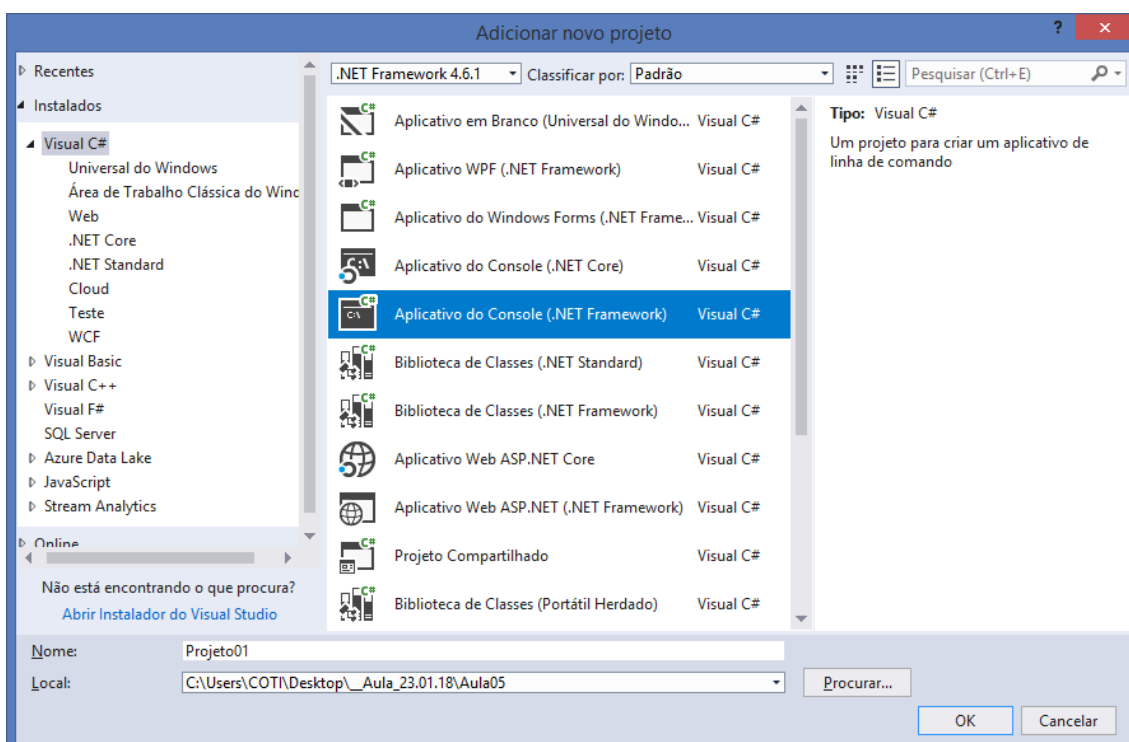
Nova solution em branco:

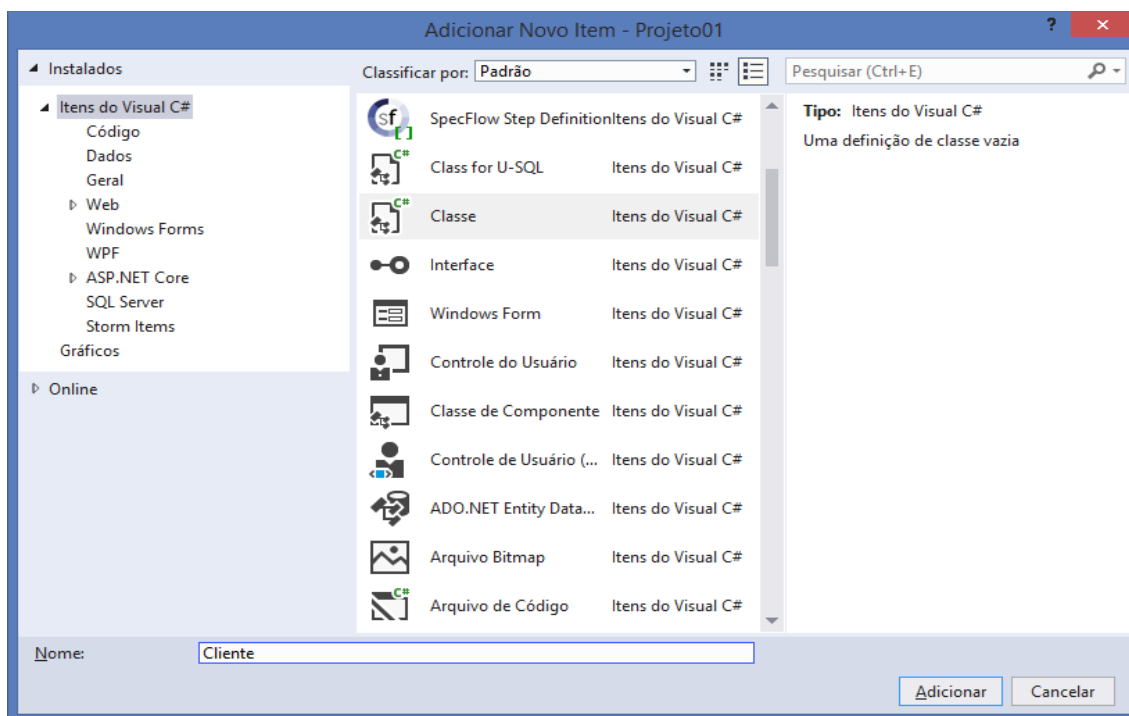


Criando o primeiro projeto Console Application



Aplicativo do Console .NET Framework





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

namespace Projeto01.Entidades
{
    public class Cliente
    {
        //propriedades [prop] + 2x[tab]
        public int IdCliente { get; set; }
        public string Nome { get; set; }

        //construtor default [ctor] + 2x[tab]
        public Cliente()
        {
            //vazio..
        }

        //sobrecarga de construtores (overloading)
        public Cliente(int idCliente, string nome)
        {
            IdCliente = idCliente;
            Nome = nome;
        }

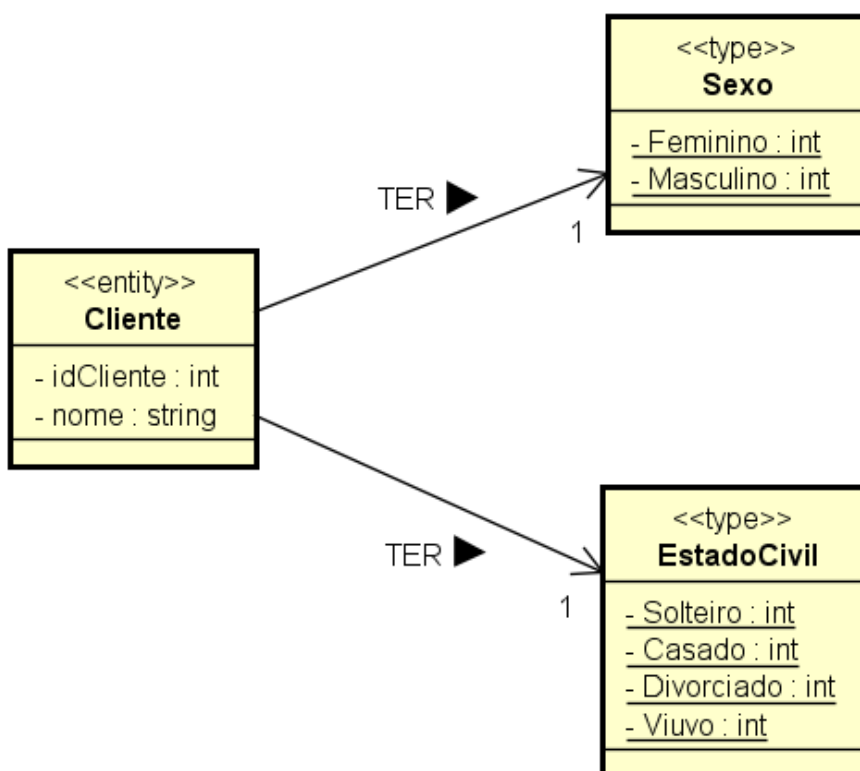
        //sobrescrita (override) do método ToString()
        public override string ToString()
        {
            return $"Id do Cliente: {IdCliente}, Nome: {Nome}";
        }
    }
}
```

ENUMs

São tipos de dados multivalorados. Utilizados para definir atributos ou propriedades com valores já predefinidos. Exemplo:

Cliente TEM Sexo (Feminino, Masculino)

Cliente TEM Estado Civil (Solteiro, Casado, Viuvo, Divorciado)



```

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

namespace Projeto01.Entidades.Tipos
{
    public enum Sexo
    {
        Feminino = 1,
        Masculino = 2
    }
}

```

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

```
namespace Projeto01.Entidades.Tipos
```

```
{
    public enum EstadoCivil
    {
        Solteiro = 1,
        Casado = 2,
        Divorciado = 3,
        Viuvo = 4
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades.Tipos; //importando..

namespace Projeto01.Entidades
{
    public class Cliente
    {
        //propriedades [prop] + 2x[tab]
        public int IdCliente { get; set; }
        public string Nome { get; set; }
        public Sexo Sexo { get; set; }
        public EstadoCivil EstadoCivil { get; set; }

        //construtor default [ctor] + 2x[tab]
        public Cliente()
        {
            //vazio..
        }

        //sobrecarga de construtores (overloading)
        public Cliente(int idCliente, string nome,
            Sexo sexo, EstadoCivil estadoCivil)
        {
            IdCliente = idCliente;
            Nome = nome;
            Sexo = sexo;
            EstadoCivil = estadoCivil;
        }
    }
}
```

```
//sobrescrita (override) do método ToString()
public override string ToString()
{
    return $"Id do Cliente: {IdCliente}, Nome: {Nome},
        Sexo: {Sexo}, Estado Civil: {EstadoCivil}";
}
}
```

Interfaces

São estruturas de programação Orientada a Objetos que tem como objetivo definir um contrato. Em uma interface só podemos declarar **métodos abstratos**, ou seja, métodos que não possuem corpo, apenas assinatura.

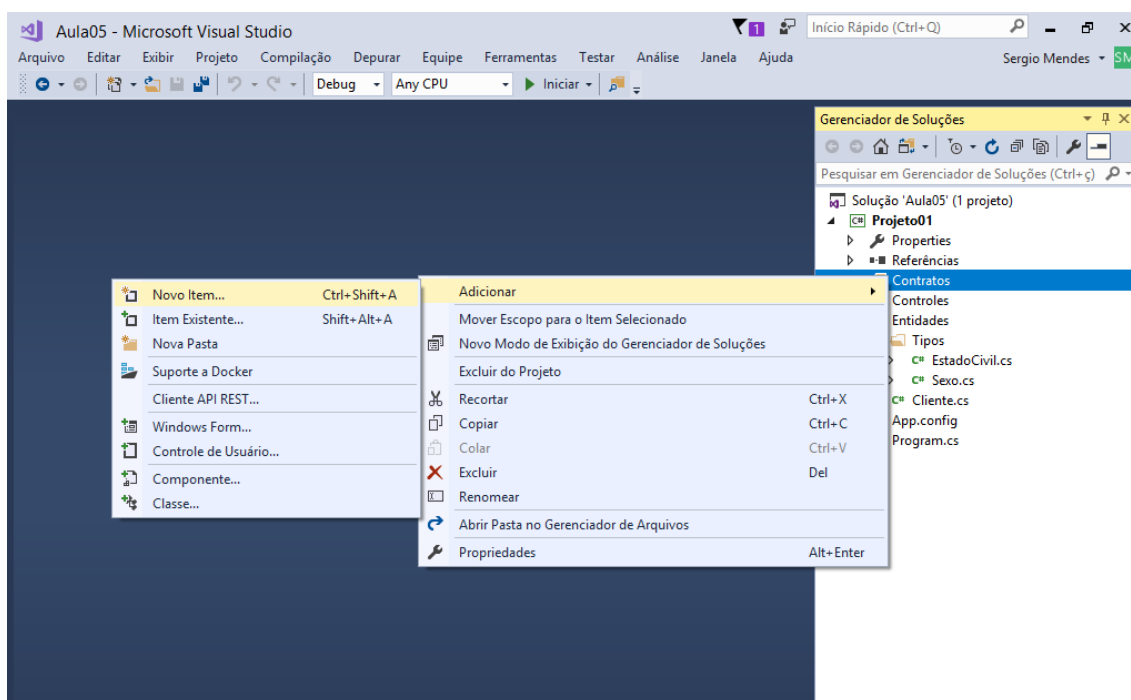
**** Observação:** nomes de interfaces começam com a letra **I**

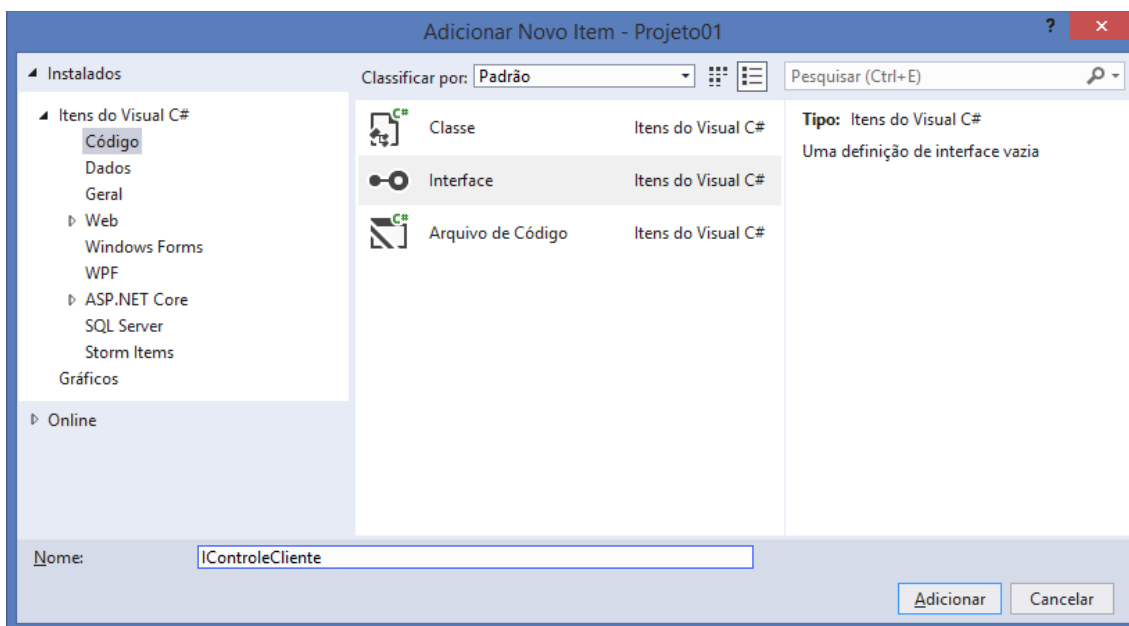
Regras sobre interfaces:

- Interfaces não podem ter atributos
- Interfaces não podem ter construtores
- Métodos de interface só podem ser abstratos
- Quando uma classe herda uma interface, a classe é obrigada pelo compilador a implementar (fornecer corpo) para todos os métodos da interface.

Criando uma interface:

Esta será um interface que terá com objetivo manipular clientes.





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades; //importando..
using Projeto01.Entidades.Tipos; //importando..

namespace Projeto01.Contratos
{
    public interface IControleCliente
    {
        //método abstrato..
        void Inserir(Cliente c);

        //método abstrato..
        void Atualizar(Cliente c);

        //método abstrato..
        void Excluir(int idCliente);

        //método abstrato..
        List<Cliente> ConsultarTodos();

        //método abstrato..
        Cliente ConsultarPorId(int idCliente);

        //método abstrato..
        List<Cliente> ConsultarPorNome(string nome);

        //método abstrato..
        List<Cliente> ConsultarPorSexo(Sexo sexo);

        //método abstrato..
        List<Cliente> ConsultarPorEstadoCivil(EstadoCivil estadoCivil);
    }
}
```

LINQ e LAMBDA

São ferramentas da linguagem C# para manipulação de coleções de objetos. Podemos através de LINQ ou de LAMBDA executar rotinas de varredura, filtro, ordenação, agrupamento, junção etc em nossas coleções de objetos.

LINQ (Language Integrated Query)

É uma sintaxe do C# muito parecida com a linguagem SQL, pois possui comandos (palavras-reservadas da linguagem) que permitem manipular coleções de objetos de forma parecida com SQL, pois possui comandos **where**, **select**, **orderby**, **join**, etc...

Atualmente, a microsoft não oferece mais suporte ao LINQ. Ele esta na linguagem mas ja não será atualizado, embora ainda existam projetos em mercado que utilizam o LINQ.

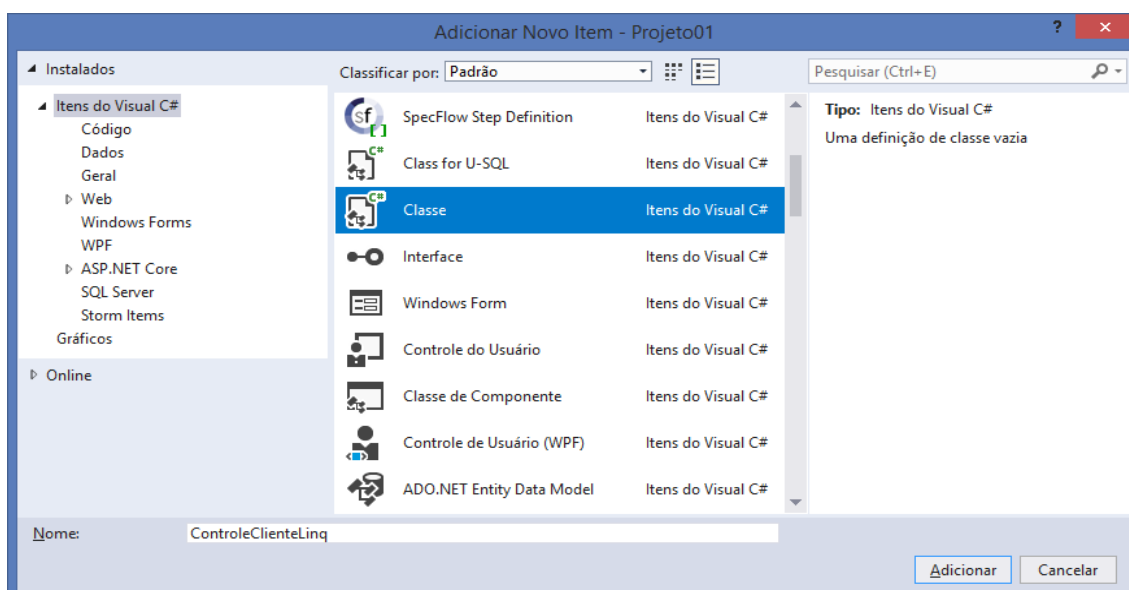
LAMBDA

É uma sintaxe mais moderna em relação ao LINQ que também permite manipular coleções de objetos. Atualmente a Microsoft tem difundido o uso do LAMBDA nas suas tecnologias.

São tecnologias baseadas em LAMBDA:

- Asp.Net MVC
- EntityFramework
- etc...

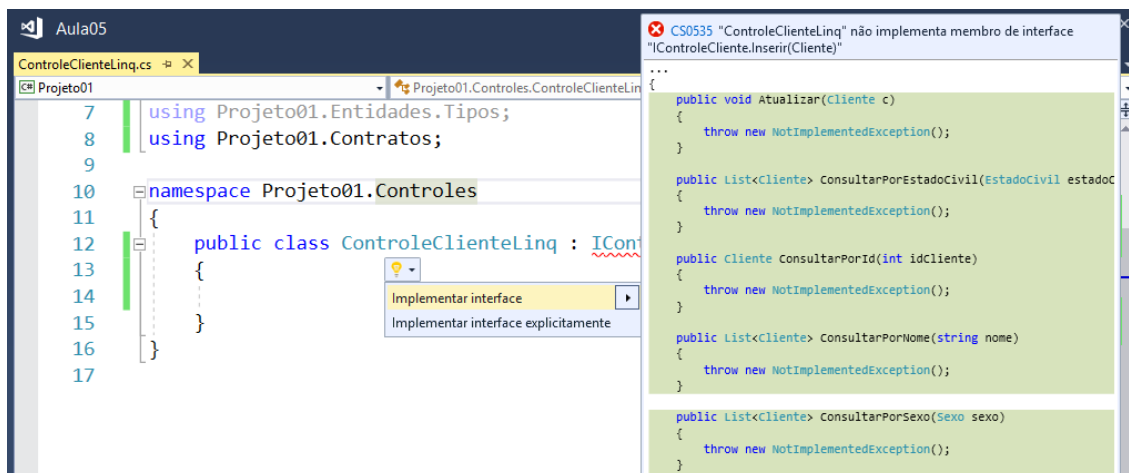
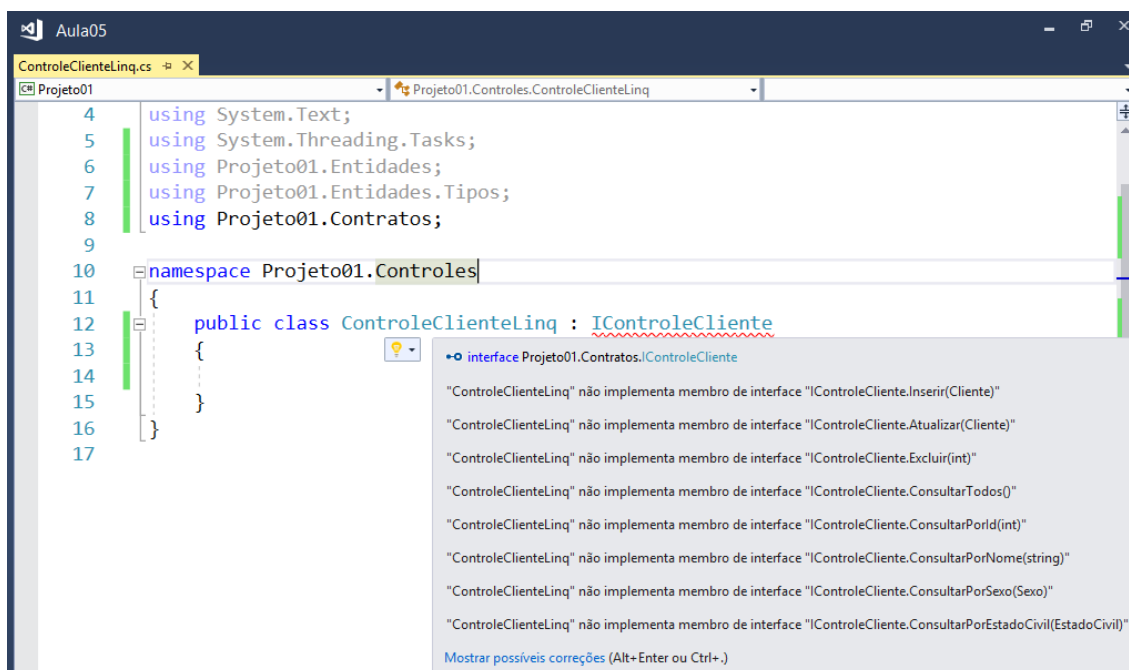
Implementando a interface IControleCliente utilizando LINQ:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;

namespace Projeto01.Controles
{
    public class ControleClienteLinq : IControleCliente
    {
    }
}
```

Implementando os metodos da interface:



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;

namespace Projeto01.Controles
{
    public class ControleClienteLinq : IControleCliente
    {
        public void Inserir(Cliente c)
        {
            throw new NotImplementedException();
        }

        public void Atualizar(Cliente c)
        {
            throw new NotImplementedException();
        }

        public void Excluir(int idCliente)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> ConsultarTodos()
        {
            throw new NotImplementedException();
        }

        public Cliente ConsultarPorId(int idCliente)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> ConsultarPorNome(string nome)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> ConsultarPorSexo(Sexo sexo)
        {
            throw new NotImplementedException();
        }

        public List<Cliente> ConsultarPorEstadoCivil(EstadoCivil estadoCivil)
        {
            throw new NotImplementedException();
        }
    }
}
```

Implementando os métodos:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;

namespace Projeto01.Controles
{
    public class ControleClienteLinq : IControleCliente
    {
        //atributo do tipo lista de cliente..
        private List<Cliente> listagemClientes;

        //construtor [ctor] + 2x[tab]
        public ControleClienteLinq()
        {
            //inicializar a lista de clientes..
            listagemClientes = new List<Cliente>();
        }

        public void Inserir(Cliente c)
        {
            //buscar dentro da lista se ha algum cliente com o id ja existente..
            var query = from cli in listagemClientes
                        where cli.IdCliente == c.IdCliente
                        select cli;

            if(query.Count() == 0) //se não encontrou registros..
            {
                listagemClientes.Add(c); //adicionando na lista..
            }
        }

        public void Atualizar(Cliente c)
        {
            //buscar o cliente dentro da lista..
            var query = from cli in listagemClientes
                        where cli.IdCliente == c.IdCliente
                        select cli;

            //se exatamente 1 registro foi encontrado..
            if (query.Count() == 1)
            {
                //retornar o cliente encontrado..
                Cliente registro = query.First();

                registro.Nome = c.Nome;
                registro.Sexo = c.Sexo;
                registro.EstadoCivil = c.EstadoCivil;
            }
        }
    }
}
```

```
public void Excluir(int idCliente)
{
    //buscar o cliente dentro da lista..
    var query = from cli in listagemClientes
                where cli.IdCliente == idCliente
                select cli;

    //se foi encontrado..
    if(query.Count() == 1)
    {
        //obter o elemento da lista..
        Cliente registro = query.First();

        //removendo..
        listagemClientes.Remove(registro);
    }
}

public List<Cliente> ConsultarTodos()
{
    //consulta de clientes ordenado pelo id..
    var query = from cli in listagemClientes
                orderby cli.IdCliente ascending
                select cli;

    //retornando os dados da consulta..
    return query.ToList();
}

public Cliente ConsultarPorId(int idCliente)
{
    //consulta de cliente por id..
    var query = from cli in listagemClientes
                where cli.IdCliente == idCliente
                select cli;

    //verificar se foi encontrado..
    //FirstOrDefault() -> retorna o primeiro elemento encontrado..
    //caso nenhum tenha sido encontrado, retorna null..
    return query.FirstOrDefault();
}

public List<Cliente> ConsultarPorNome(string nome)
{
    //consulta de clientes pelo nome..
    var query = from cli in listagemClientes
                where cli.Nome.Contains(nome)
                orderby cli.Nome ascending
                select cli;

    //retornando os dados..
    return query.ToList();
}

public List<Cliente> ConsultarPorSexo(Sexo sexo)
{
    //consulta de clientes pelo sexo..
    var query = from cli in listagemClientes
                where cli.Sexo == sexo
                orderby cli.Sexo ascending
```

```

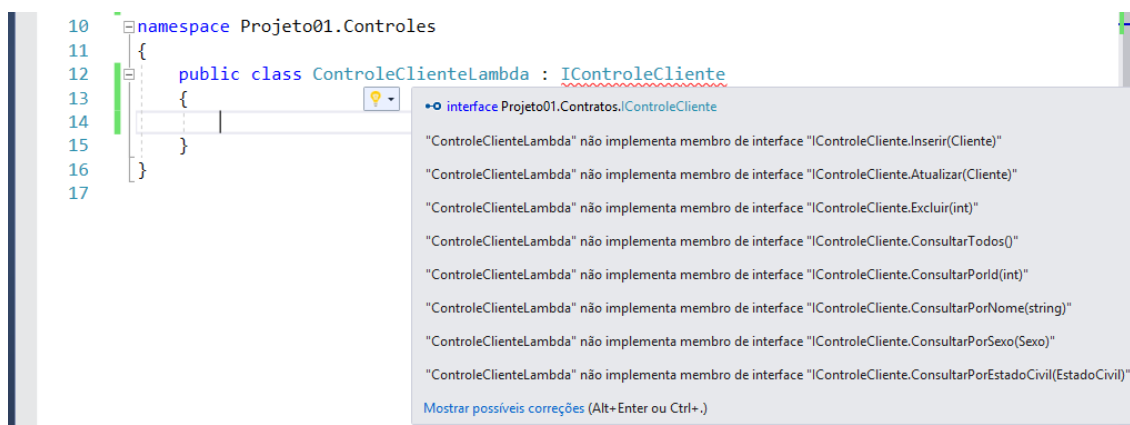
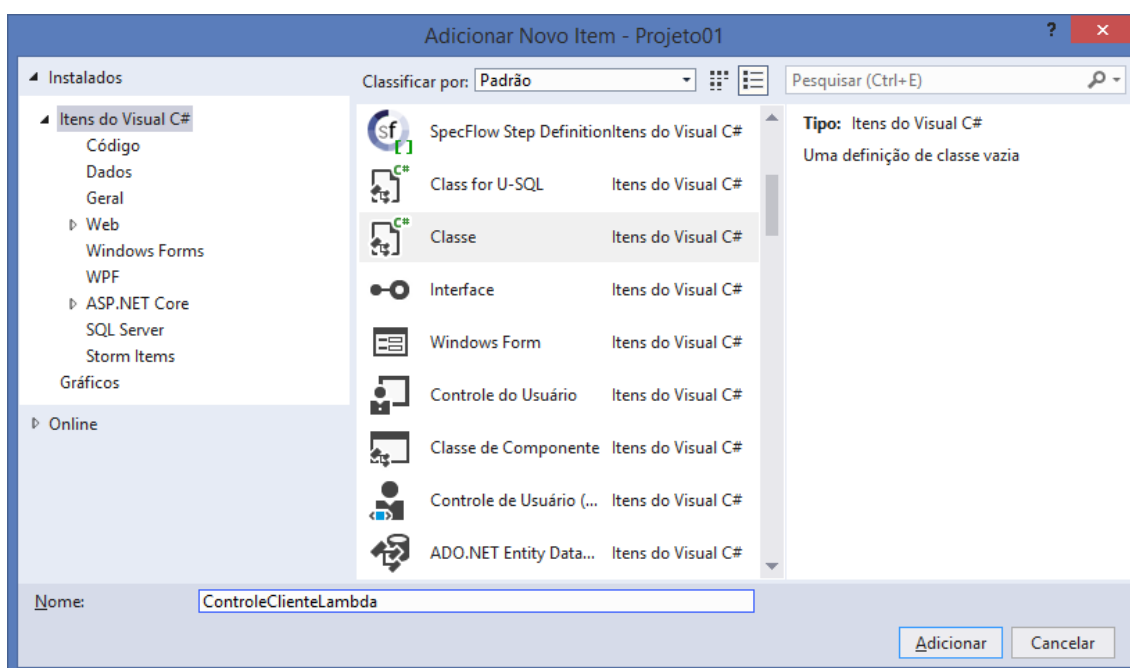
        select cli;

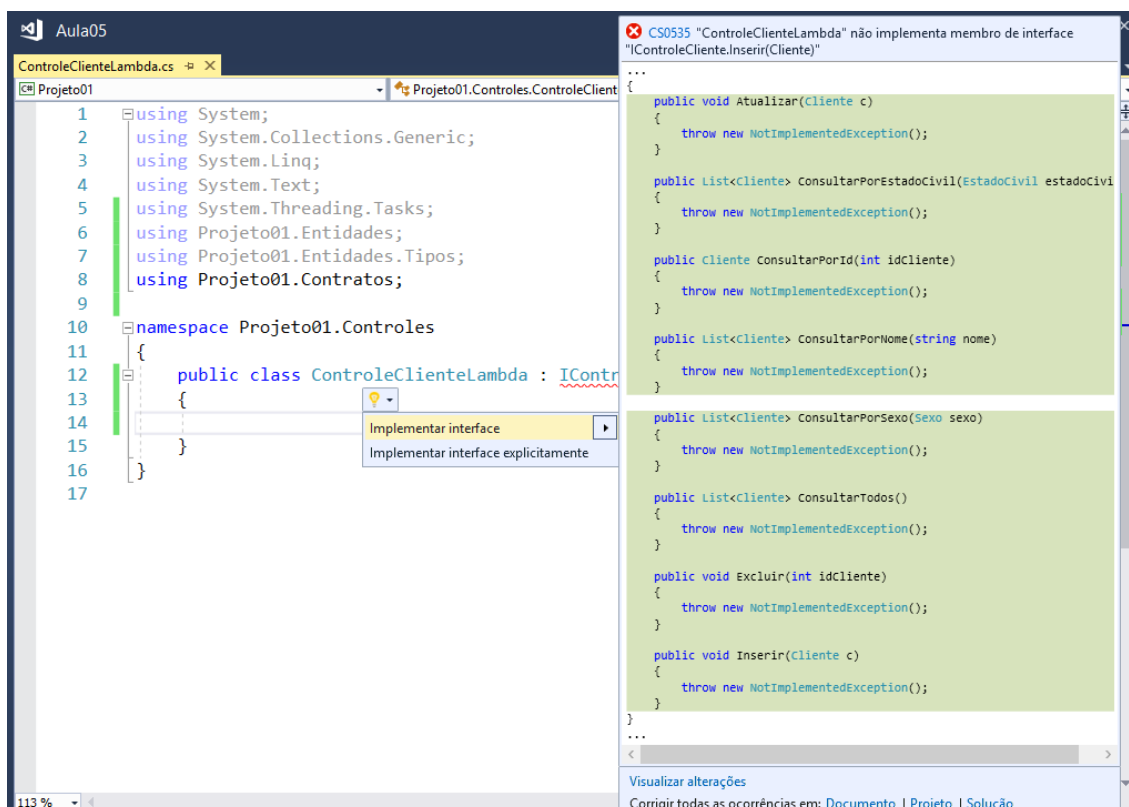
        //retornando os dados..
        return query.ToList();
    }

    public List<Cliente> ConsultarPorEstadoCivil(EstadoCivil estadoCivil)
    {
        //consulta de clientes pelo estado civil..
        var query = from cli in listagemClientes
                     where cli.EstadoCivil == estadoCivil
                     orderby cli.EstadoCivil ascending
                     select cli;

        //retornando os dados..
        return query.ToList();
    }
}

```





```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;

namespace Projeto01.Controles
{
    public class ControleClienteLambda : IControleCliente
    {
        //atributo..
        private List<Cliente> listagemClientes;

        //construtor..
        public ControleClienteLambda()
        {
            //inicializando o atributo..
            listagemClientes = new List<Cliente>();
        }

        public void Inserir(Cliente c)
        {
            //verificar se ja existe o cliente com o id..
            var query = listagemClientes
                .Where(cli => cli.IdCliente == c.IdCliente);
        }
    }
}
```

```
        if(query.Count() == 0) //se não existe..
        {
            listagemClientes.Add(c);
        }
    }

    public void Atualizar(Cliente c)
    {
        //verificar se ja existe o cliente com o id..
        var query = listagemClientes
            .Where(cli => cli.IdCliente == c.IdCliente);

        if (query.Count() == 1) //se existe..
        {
            Cliente registro = query.First();

            registro.Nome = c.Nome;
            registro.Sexo = c.Sexo;
            registro.EstadoCivil = c.EstadoCivil;
        }
    }

    public void Excluir(int idCliente)
    {
        //verificar se ja existe o cliente com o id..
        var query = listagemClientes
            .Where(cli => cli.IdCliente == idCliente);

        if (query.Count() == 1) //se existe..
        {
            Cliente registro = query.First();

            //removendo da lista..
            listagemClientes.Remove(registro);
        }
    }

    public List<Cliente> ConsultarTodos()
    {
        //selecionar todos os elementos da lista..
        var query = listagemClientes.OrderBy(cli => cli.IdCliente);
        //retornar os resultados..
        return query.ToList();
    }

    public Cliente ConsultarPorId(int idCliente)
    {
        var query = listagemClientes
            .Where(cli => cli.IdCliente == idCliente);

        //retornar o primeiro elemento encontrado ou
        //null se nenhum for encontrado..
    }
```

```
        return query.FirstOrDefault();
    }

    public List<Cliente> ConsultarPorNome(string nome)
    {
        var query = listagemClientes
            .Where(cli => cli.Nome.Contains(nome))
            .OrderBy(cli => cli.Nome);

        //retornando os resultados..
        return query.ToList();
    }

    public List<Cliente> ConsultarPorSexo(Sexo sexo)
    {
        var query = listagemClientes
            .Where(cli => cli.Sexo == sexo)
            .OrderBy(cli => cli.Sexo);

        //retornando os resultados..
        return query.ToList();
    }

    public List<Cliente> ConsultarPorEstadoCivil(EstadoCivil estadoCivil)
    {
        var query = listagemClientes
            .Where(cli => cli.EstadoCivil == estadoCivil)
            .OrderBy(cli => cli.EstadoCivil);

        //retornando os resultados..
        return query.ToList();
    }
}

}
```


Polimorfismo

Capacidade de instanciar um objeto utilizando construtores de classes que herdam este objeto. Atraves do polimorfismo podemos alterar a instancia de um mesmo objeto fazendo com que os sus metodos apresentem comportamentos diferentes em tempo de execução.

Exemplo:

Declarando um objeto da interface sendo que sem que ele seja instanciado. Seu valor é null, não possuindo espaço de memória.

IControleCliente ct = null;
[Tipo - Interface] [Objeto] [Sem ponteiro]

Podemos instanciar o objeto acima utilizando as classes que o implementaram, por exemplo:

ct = new ControleClienteLinq();
[Objeto] [Classe que implementa a interface]

Ou

ct = new ControleClienteLambda();
[Objeto] [Classe que implementa a interface]

Neste caso podemos dizer que temos polimorfismo do objeto 'ct', pois os metodos que ele executar terão o seu comportamento definido pela forma como o objeto foi instanciado.

Outro exemplo:

```
public interface ISerVivo
{
    void Respirar();
}

public class Animal : ISerVivo
{
    public void Respirar()
    {
        Console.WriteLine("Animal Respirando");
    }
}
```

```
public class Planta : IServivo
{
    public void Respirar()
    {
        Console.WriteLine("Planta Respirando");
    }
}
```

Logo:

```
public class Test
{
    void Main()
    {
        IServivo s = new Animal();

        s.Respirar();
    }
}
```

Ou:

```
public class Test
{
    void Main()
    {
        IServivo s = new Planta();

        s.Respirar();
    }
}
```

O comportamento do método 'Respirar()' acima depende de qual instancia foi dada para o objeto 's' da interface.

Executando:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;
using Projeto01.Controles;
```

```
namespace Projeto01
{
    class Program
    {
        static void Main(string[] args)
        {
            IControleCliente ct = null;

            Console.WriteLine("Escolha (1)LINQ ou (2)LAMBDA..: ");
            int opcao = int.Parse(Console.ReadLine());

            switch(opcao)
            {
                case 1:
                    ct = new ControleClienteLinq(); //POLIMORFISMO..
                    break;

                case 2:
                    ct = new ControleClienteLambda(); //POLIMORFISMO..
                    break;
            }

            //criando clientes..
            Cliente c1 = new Cliente
                (1, "Ana", Sexo.Feminino, EstadoCivil.Solteiro);

            Cliente c2 = new Cliente
                (2, "Leo", Sexo.Masculino, EstadoCivil.Casado);

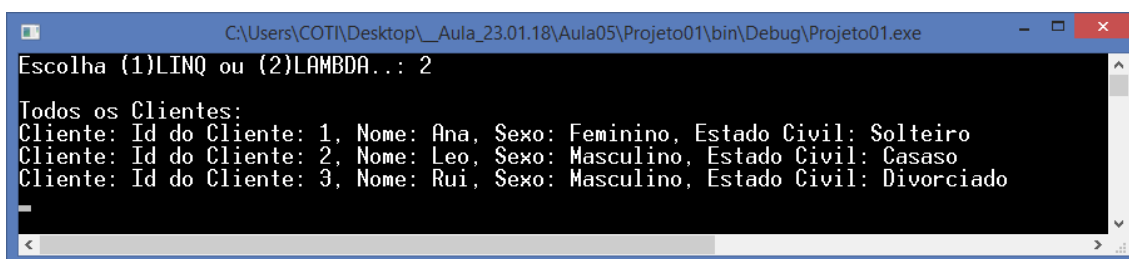
            Cliente c3 = new Cliente
                (3, "Rui", Sexo.Masculino, EstadoCivil.Divorciado);

            //testando..
            ct.Inserir(c1);
            ct.Inserir(c2);
            ct.Inserir(c3);

            //exibir os clientes..
            Console.WriteLine("\nTodos os Clientes:");
            foreach(Cliente c in ct.ConsultarTodos())
            {
                Console.WriteLine("Cliente: " + c.ToString());
            }

            Console.ReadKey();
        }
    }
}
```

Saida:



```
C:\Users\COTI\Desktop\_Aula_23.01.18\Aula05\Projeto01\bin\Debug\Projeto01.exe
Escolha (1)LINQ ou (2)LAMBDA..: 2
Todos os Clientes:
Cliente: Id do Cliente: 1, Nome: Ana, Sexo: Feminino, Estado Civil: Solteiro
Cliente: Id do Cliente: 2, Nome: Leo, Sexo: Masculino, Estado Civil: Casado
Cliente: Id do Cliente: 3, Nome: Rui, Sexo: Masculino, Estado Civil: Divorciado
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto01.Entidades;
using Projeto01.Entidades.Tipos;
using Projeto01.Contratos;
using Projeto01.Controles;

namespace Projeto01
{
    class Program
    {
        static void Main(string[] args)
        {
            IControleCliente ct = null;

            Console.WriteLine("Escolha (1)LINQ ou (2)LAMBDA..: ");
            int opcao = int.Parse(Console.ReadLine());

            switch(opcao)
            {
                case 1:
                    ct = new ControleClienteLinq(); //POLIMORFISMO..
                    break;

                case 2:
                    ct = new ControleClienteLambda(); //POLIMORFISMO..
                    break;
            }

            //criando clientes..
            Cliente c1 = new Cliente
                (1, "Ana", Sexo.Feminino, EstadoCivil.Solteiro);

            Cliente c2 = new Cliente
                (2, "Leo", Sexo.Masculino, EstadoCivil.Casado);

            Cliente c3 = new Cliente
                (3, "Rui", Sexo.Masculino, EstadoCivil.Divorciado);

            //testando..
            ct.Inserir(c1);
            ct.Inserir(c2);
            ct.Inserir(c3);

            //exibir os clientes..
            Console.WriteLine("\nTodos os Clientes:");
            foreach(Cliente c in ct.ConsultarTodos())
            {
                Console.WriteLine("Cliente: " + c.ToString());
            }

            //exibir os clientes do sexo masculino..
            Console.WriteLine("\nClientes do Sexo Masculino:");
            foreach(Cliente c in ct.ConsultarPorSexo(Sexo.Masculino))
            {
                Console.WriteLine("Cliente: " + c.ToString());
            }
        }
    }
}
```

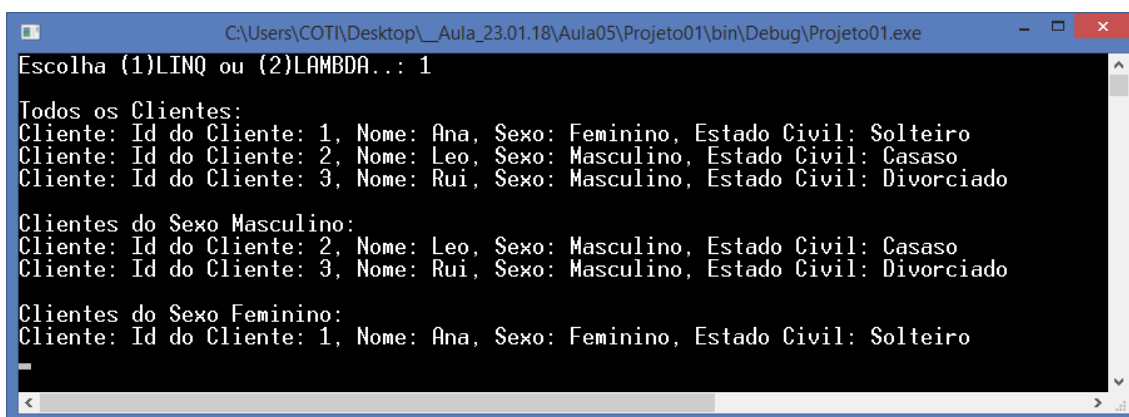
```

Console.WriteLine("\nClientes do Sexo Feminino:");
foreach (Cliente c in ct.ConsultarPorSexo(Sexo.Feminino))
{
    Console.WriteLine("Cliente: " + c.ToString());
}

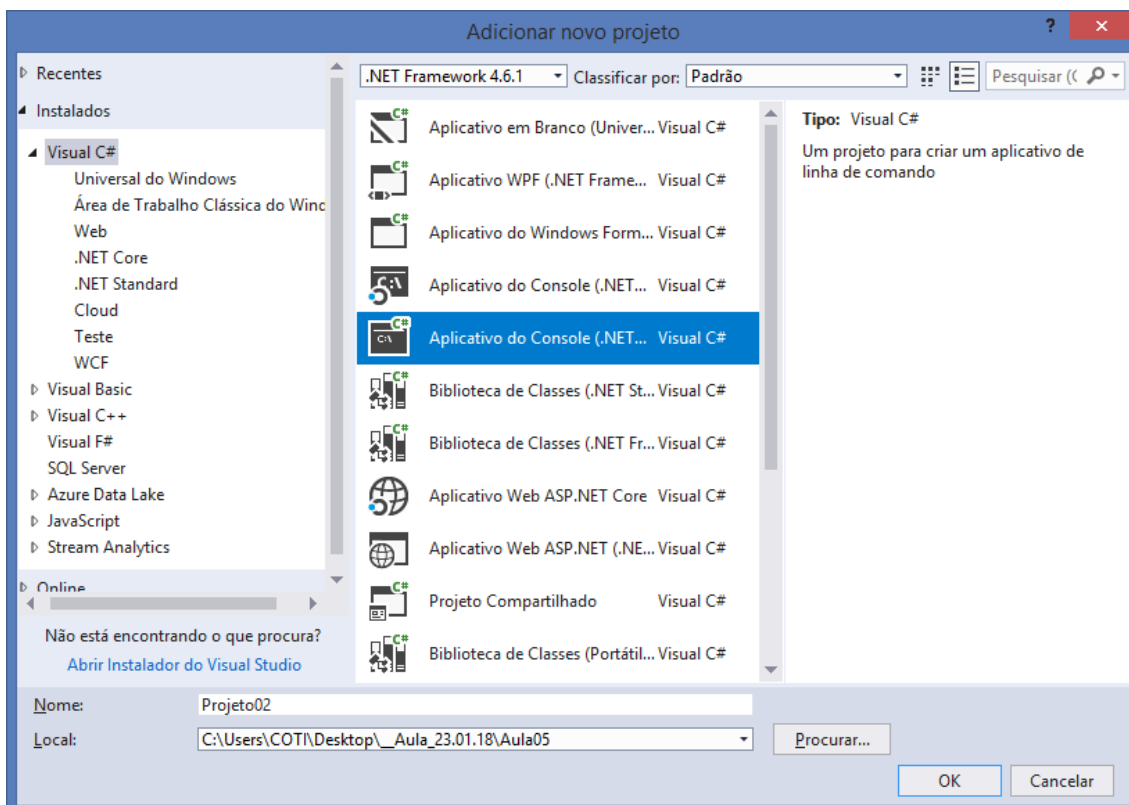
Console.ReadKey();
    }
}
}

```

Executando:



Novo projeto:



Classe de entidade:

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

namespace Projeto02.Entidades
{
    public class Cliente
    {
        //propriedades [prop] + 2x[tab]
        public int IdCliente { get; set; }
        public string Nome { get; set; }

        //construtor default [ctor] + 2x[tab]
        public Cliente()
        {
            //vazio..
        }

        //sobrecarga de construtores (overloading)
        public Cliente(int idCliente, string nome)
        {
            IdCliente = idCliente;
            Nome = nome;
        }

        //sobrescrita (override) do método ToString()
        public override string ToString()
        {
            return $"Id do Cliente: {IdCliente}, Nome: {Nome}";
        }
    }
}
```

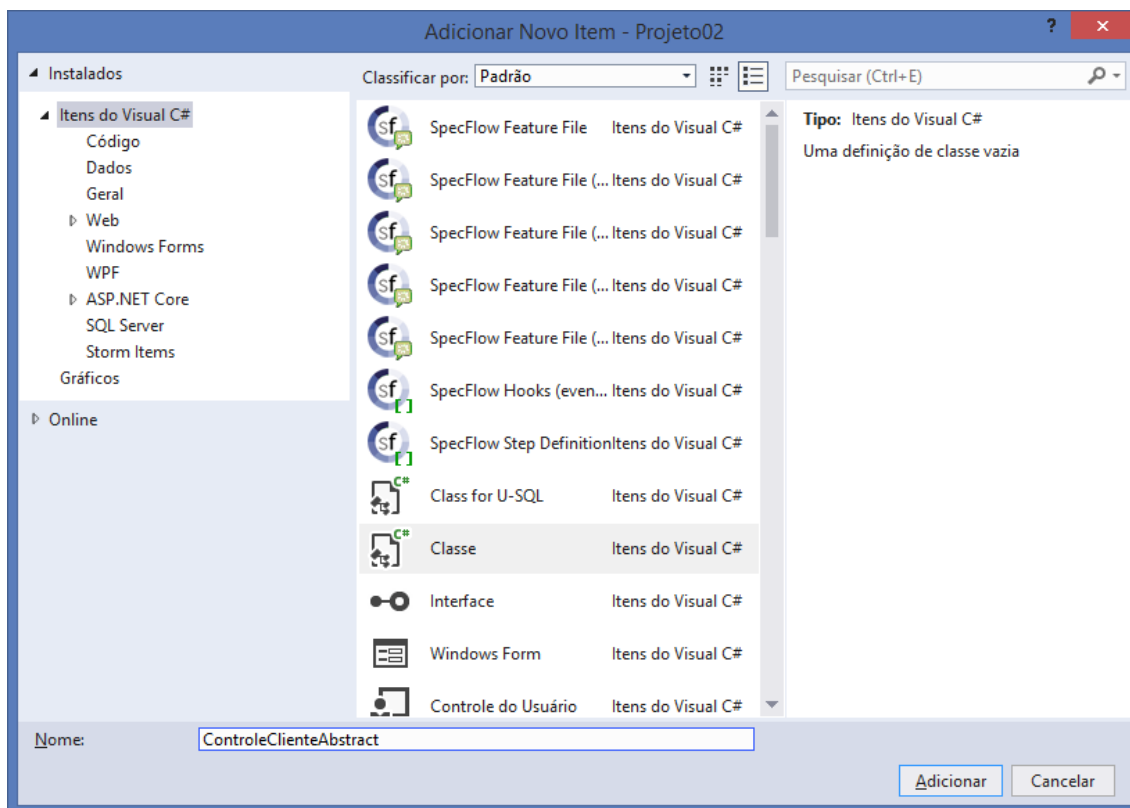
Classe Abstrata

É um tipo de classe que pode conter métodos abstratos, ou seja, métodos que deverão ser programados por classes que herdarem a classe abstrata.

Comparativo:

	Classe	Classe Abstrata	Interface
Atributos	Sim	Sim	Não
Construtores	Sim	Sim	Não
Métodos	Sim	Sim	Não
Métodos absrtratos	Não	Sim	Sim

Exemplo:

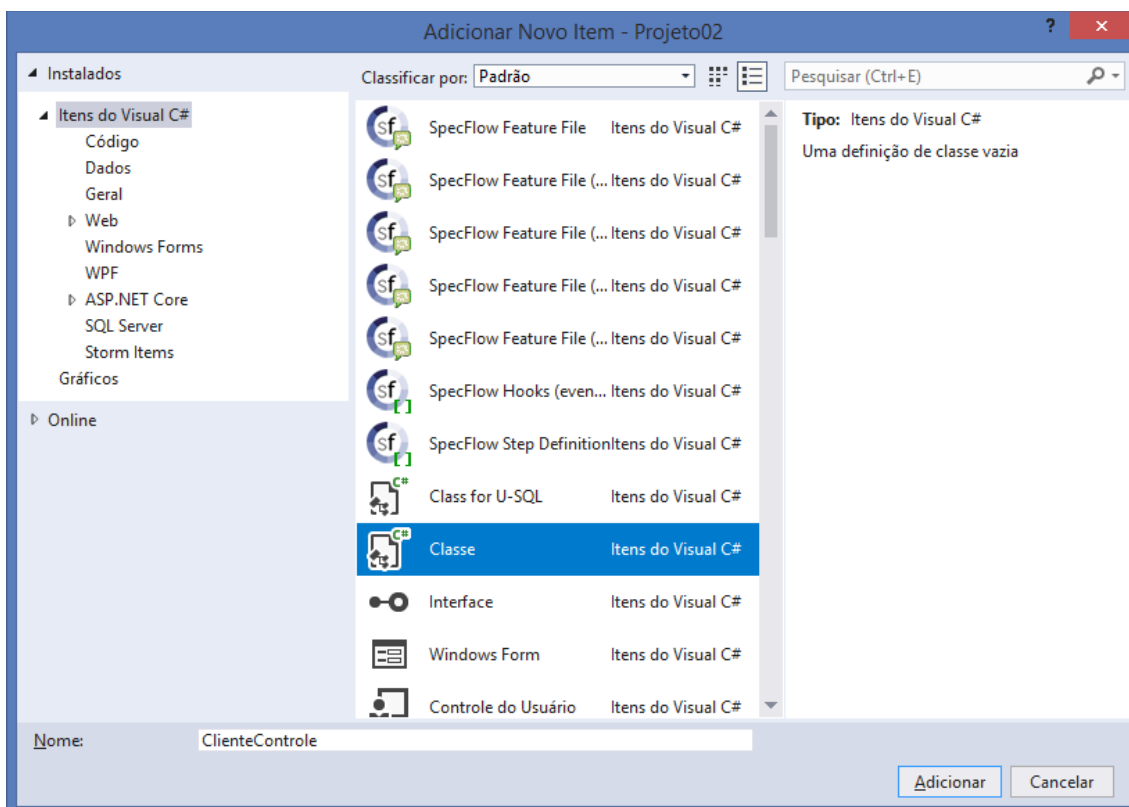


```
using Projeto02.Entidades;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

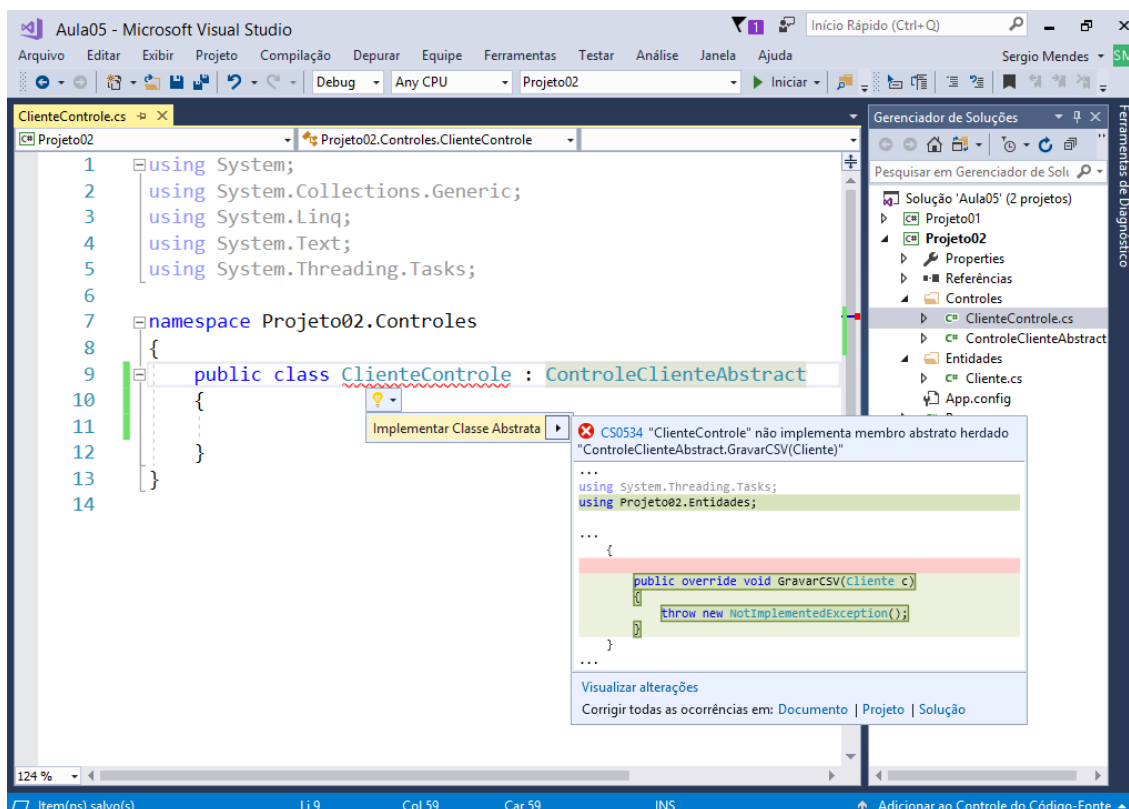
namespace Projeto02.Controles
{
    //classe abstrata..
    public abstract class ControleClienteAbstract
    {
        public void GravarTxt(Cliente c)
        {
            StreamWriter sw = new StreamWriter("c:\\temp\\clientes.txt", true);
            sw.WriteLine(c.ToString());
            sw.Close();
        }

        //método abstrato..
        public abstract void GravarCSV(Cliente c);
    }
}
```

** Sendo assim, a subclasse que herdar a classe abstrata deverá implementar o método GravarCSV



Implementando a classe abstrata:




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Projeto02.Entidades;
using System.IO;

namespace Projeto02.Controles
{
    public class ControleCliente : ControleClienteAbstract
    {
        public override void GravarCSV(Cliente c)
        {
            StreamWriter sw = new StreamWriter("c:\\temp\\clientes.csv", true);
            sw.WriteLine("{0};{1}", c.IdCliente, c.Nome);
            sw.Close();
        }
    }
}
```

Executando:

```
using Projeto02.Controles;
using Projeto02.Entidades;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Projeto02
{
    class Program
    {
        static void Main(string[] args)
        {
            Cliente c = new Cliente(1, "Sergio Mendes");

            ControleCliente cc = new ControleCliente();
            cc.GravarTxt(c);
            cc.GravarCSV(c);

            Console.WriteLine("Dados gravados.");
            Console.ReadKey();
        }
    }
}
```

