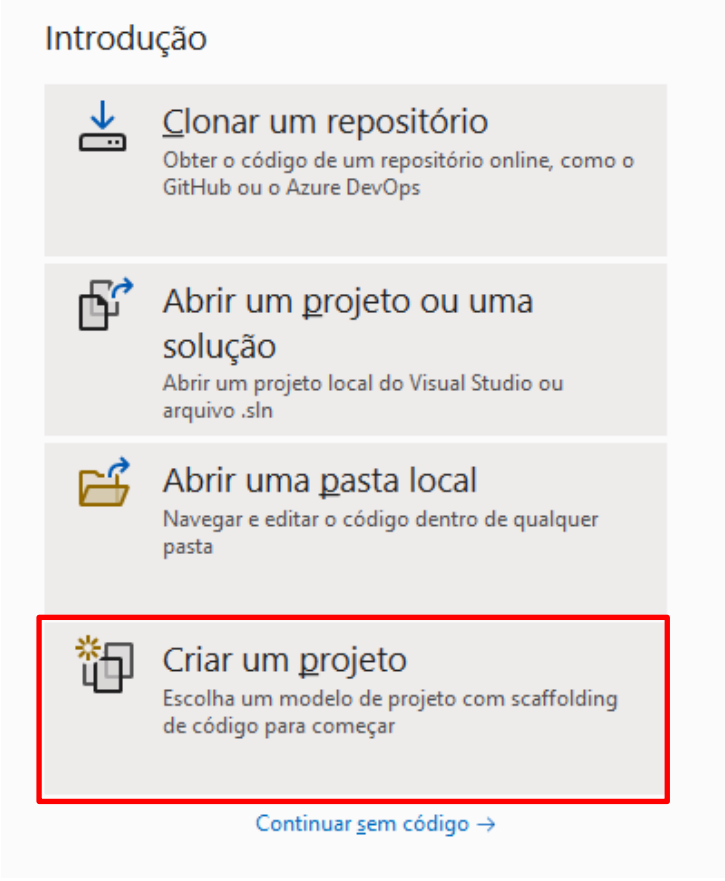
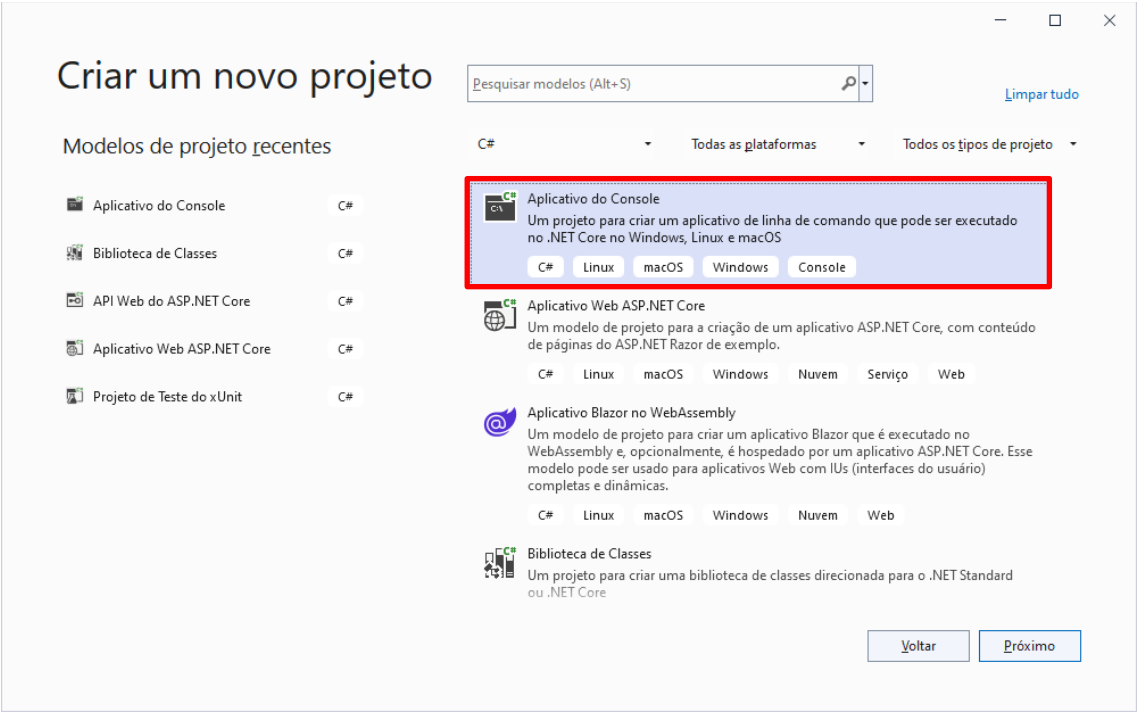
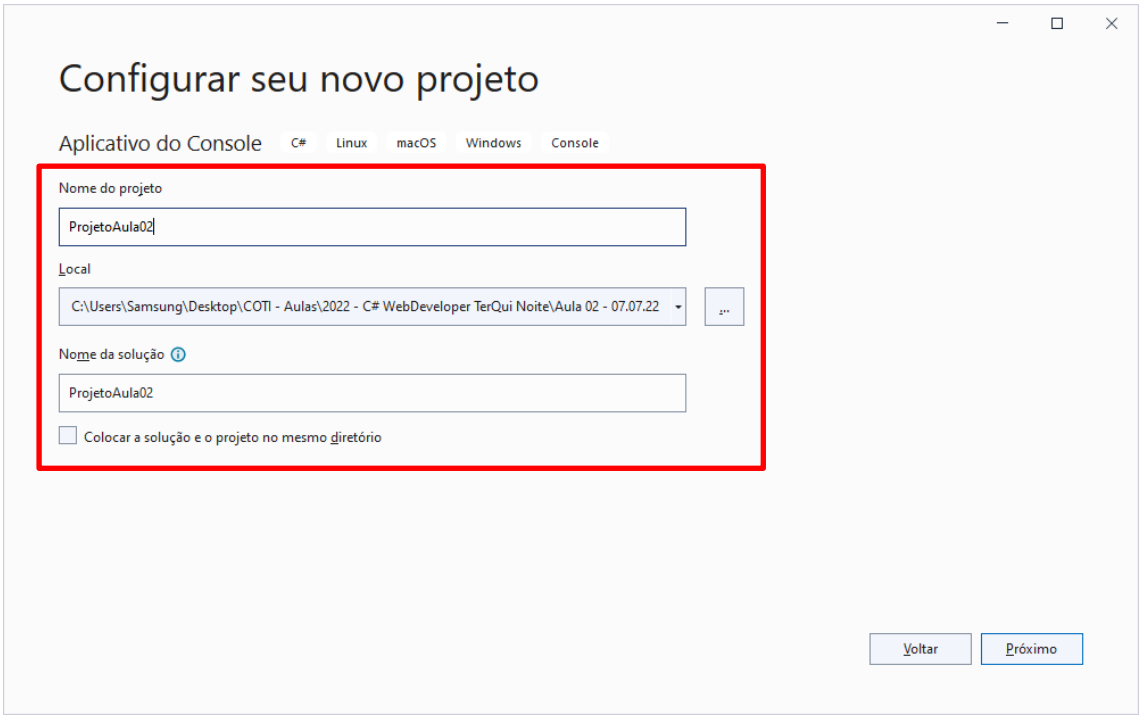
Novo projeto:



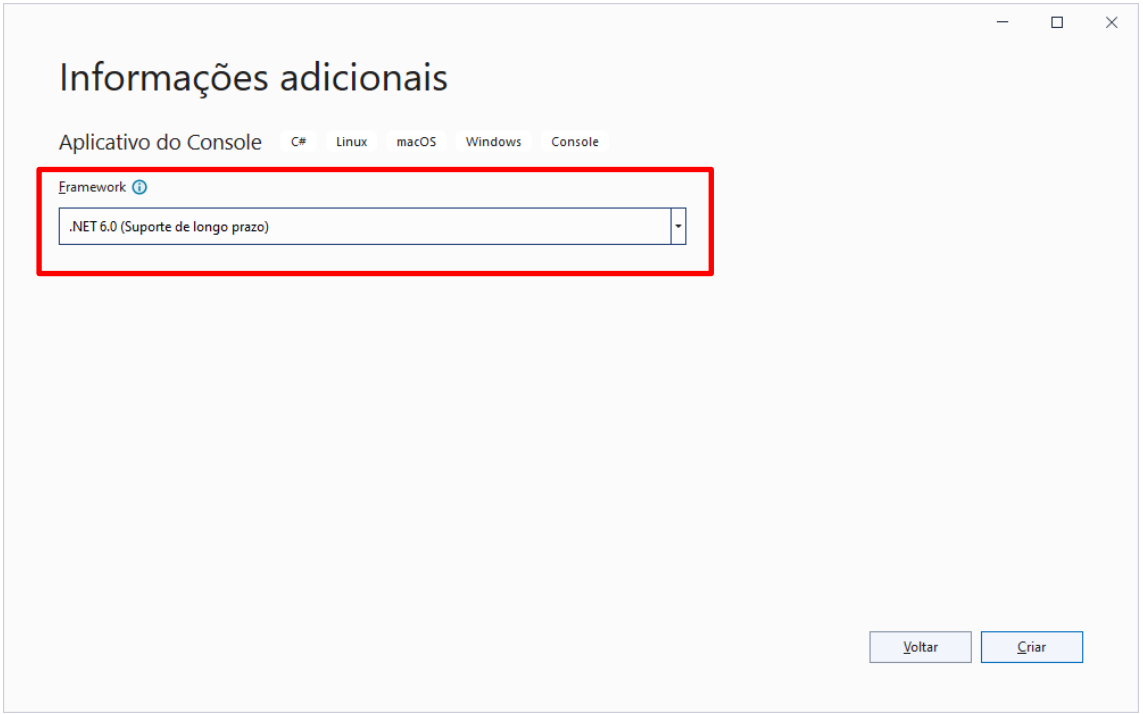
Selecione a opção: "**Aplicativo do Console**":



Nome: **ProjetoAula02**



Selecione a versão do Framework .NET
.NET (6.0)

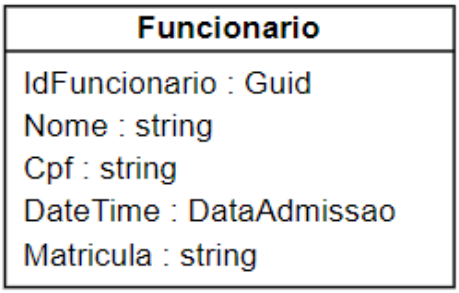


Escrevendo a classe Program.cs

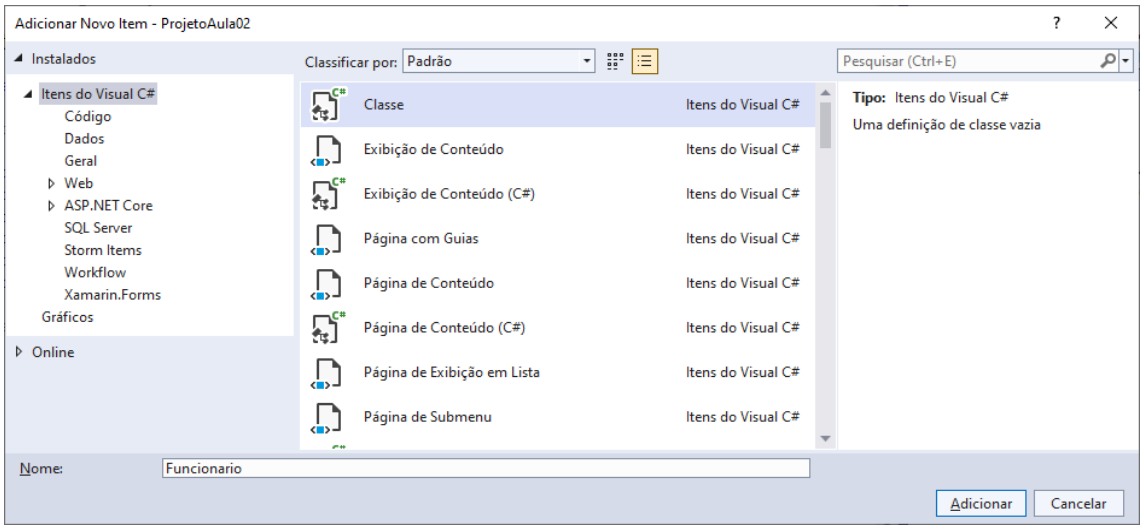
```
//localização da classe dentro do projeto
using ProjetoAula02.Controllers;

namespace ProjetoAula02
{
    //definição da classe
    public class Program
    {
        //método para executar o projeto (inicialização)
        public static void Main(string[] args)
        {
            Console.WriteLine("Aula 02 C# WebDeveloper");
        }
    }
}
```

Primeiro, vamos criar uma classe de Entidade para modelagem de dados de um funcionário:

/Entities



Em orientação a objetos, uma classe é uma descrição que abstrai um conjunto de objetos com características similares. Mais formalmente, é um conceito que encapsula abstrações de dados e procedimentos que descrevem o conteúdo e o comportamento de entidades do mundo real, representadas por objetos.



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

namespace ProjetoAula02.Entities
{
    public class Funcionario
    {
        #region Propriedades

        public Guid IdFuncionario { get; set; }
        public string Nome { get; set; }
        public string Cpf { get; set; }
        public DateTime DataAdmissao { get; set; }
        public string Matricula { get; set; }

        #endregion
    }
}
```

Em seguida, vamos criar uma classe de Entidade para modelagem de dados de uma empresa:

/Entities


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

namespace ProjetoAula02.Entities
{
    public class Empresa
    {
        #region Propriedades

        public Guid IdEmpresa { get; set; }
        public string NomeFantasia { get; set; }
        public string RazaoSocial { get; set; }
        public string Cnpj { get; set; }

        #endregion
    }
}
```

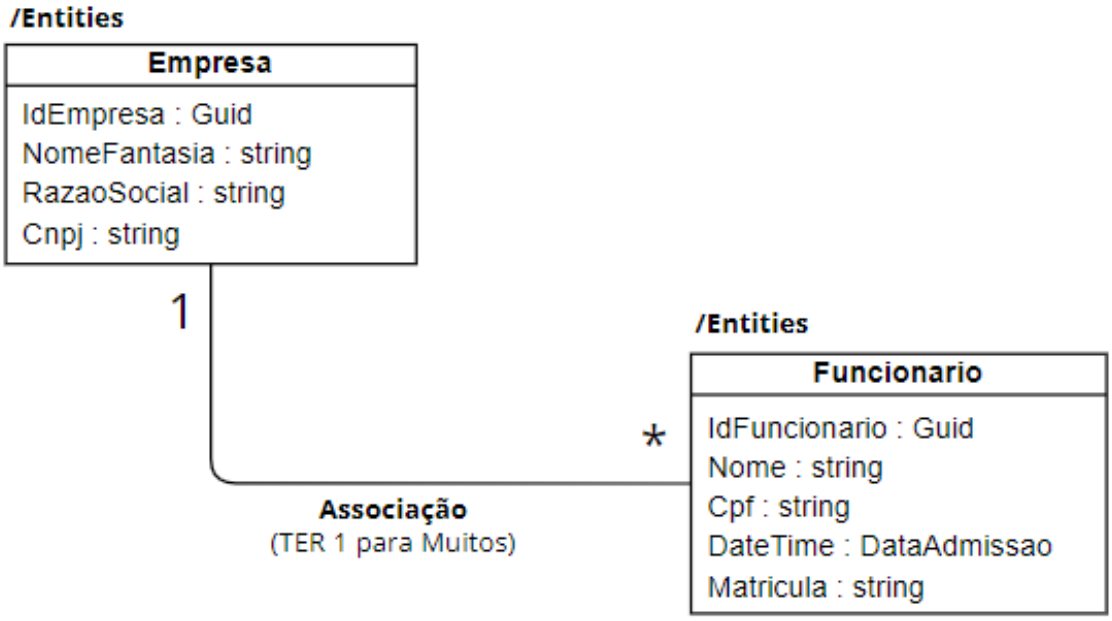
Relacionamento de associação entre classes (TER)

O relacionamento de associação define um vínculo entre classes do tipo "TER", que pode ser expresso como TER-1 ou TER-MUITOS.

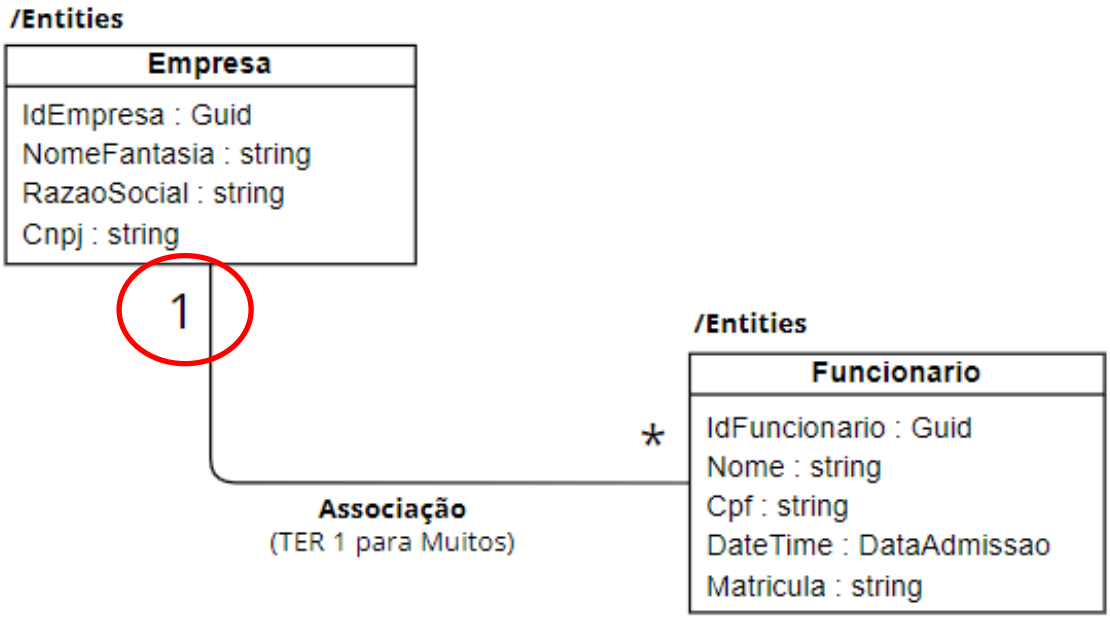
Por exemplo, vamos definir um relacionamento entre empresa e funcionário onde:

- Empresa **TEM MUITOS** Funcionários
- Funcionário **PERTENCE A 1** Empresa

Dessa forma teremos um relacionamento de multiplicidade **1 para muitos** conforme abaixo:

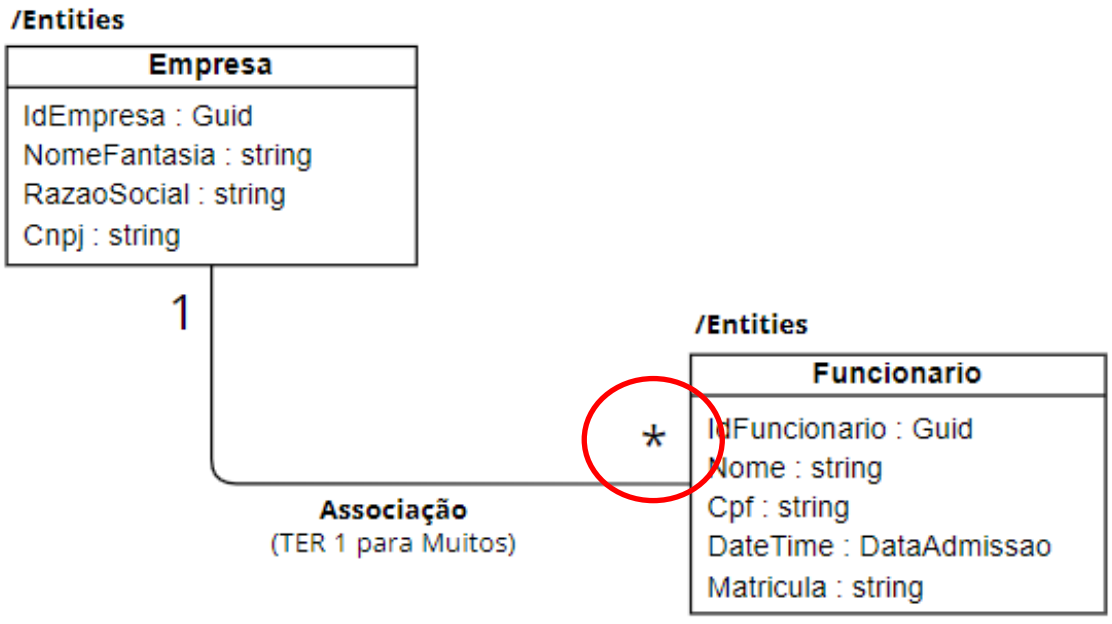


Primeiro, vamos criar o vínculo de Funcionario para Empresa **(Funcionário PERTENCE A 1 Empresa)**



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

namespace ProjetoAula02.Entities
{
    public class Funcionario
    {
        #region Propriedades

        public Guid IdFuncionario { get; set; }
        public string Nome { get; set; }
        public string Cpf { get; set; }
        public DateTime DataAdmissao { get; set; }
        public string Matricula { get; set; }
        public Empresa Empresa { get; set; }

        #endregion
    }
}
```

Em seguida, vamos criar o vínculo de Empresa para Funcionário
(Empresa POSSUI MUITOS Funcionários)




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

namespace ProjetoAula02.Entities
{
```

```
    public class Empresa
    {
        #region Propriedades

        public Guid IdEmpresa { get; set; }
        public string NomeFantasia { get; set; }
        public string RazaoSocial { get; set; }
        public string Cnpj { get; set; }
        public List<Funcionario> Funcionarios { get; set; }

        #endregion
    }
}
```

Para criarmos um relacionamento do tipo **TER-MUITOS** podemos usar uma biblioteca do .Net chamada **Collections**, que permite que declaremos coleções de objetos nas classes, tais como:

System.Collections.Generic

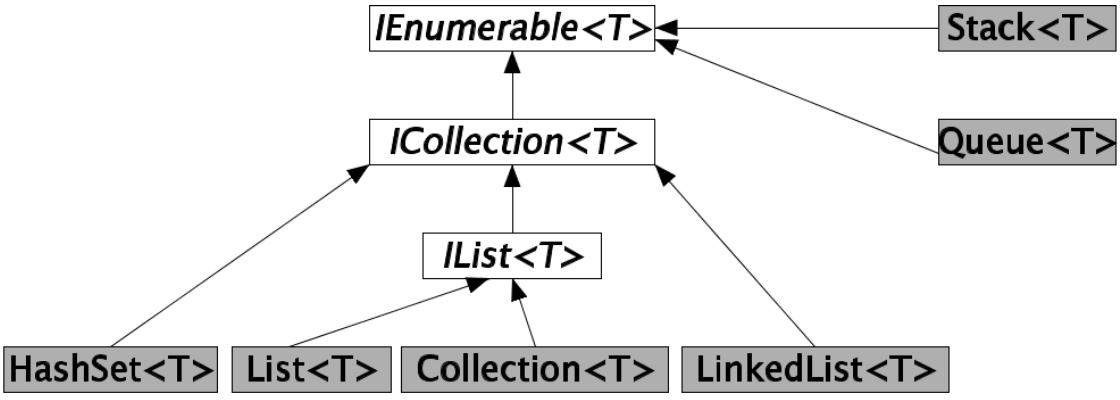
Biblioteca de coleções do .NET

Listas (List)
São o tipo mais comum e versátil de coleção, podendo adicionar, remover, ordenar elementos etc.

Filas (Queue)
Tipo de coleção de dados que utiliza o padrão FIFO – FIRST IN FIRST OUT (primeiro que entra, primeiro que sai).

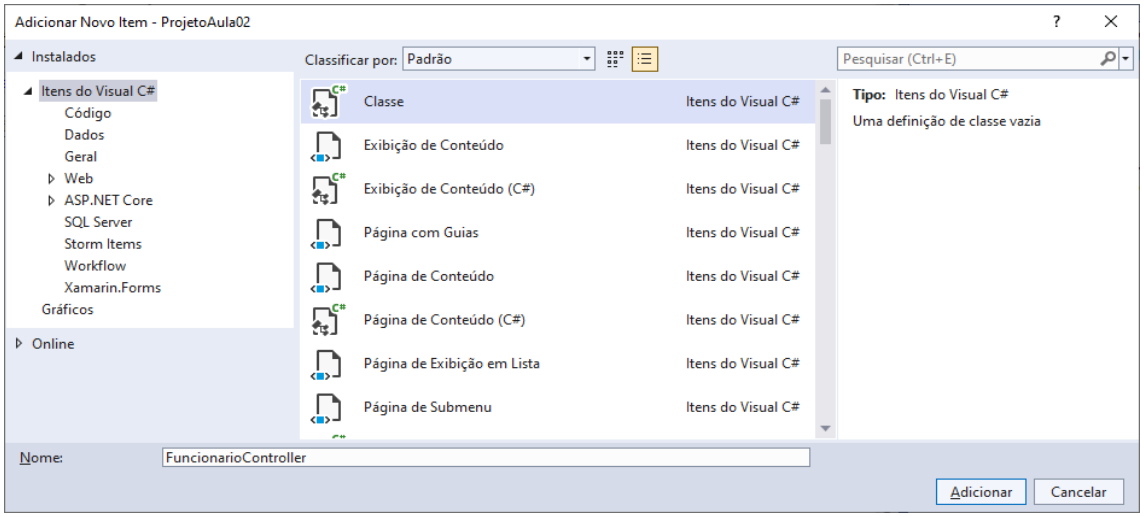
Pilhas (Stack)
Tipo de coleção de dados que utiliza o padrão LIFO – LAST IN FIRST OUT (último que entra, primeiro que sai)

Mapas (Dictionary)
Tipo de coleção que utiliza o padrão CHAVE / VALOR, onde seus elementos são endereçados através de algum tipo de chave única.



Vamos criar uma classe de controle para capturar os dados de um funcionário informado pelo usuário através do prompt do DOS:

/Controllers/**FuncionarioController.cs**



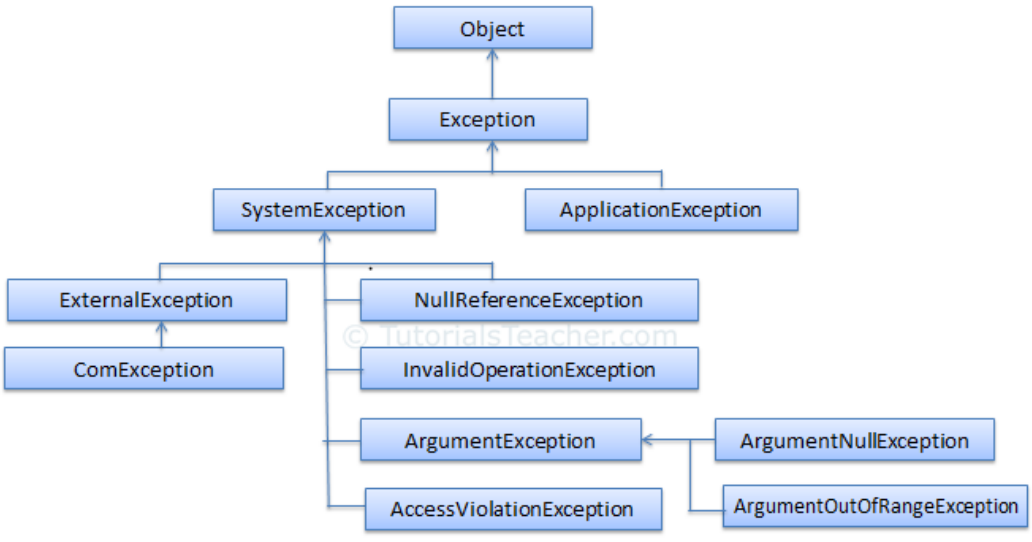
Exceptions

São exceções que um programa pode lançar em tempo de execução. Ou seja, erros em tempo de execução de uma aplicação. Em C#, a classe mais genérica para tratamento de exceções chama-se **Exception**.

Abaixo da classe Exception, temos uma árvore que possui dezenas de classes de exceção mais especificas que são utilizadas para capturar erros mais específicos.

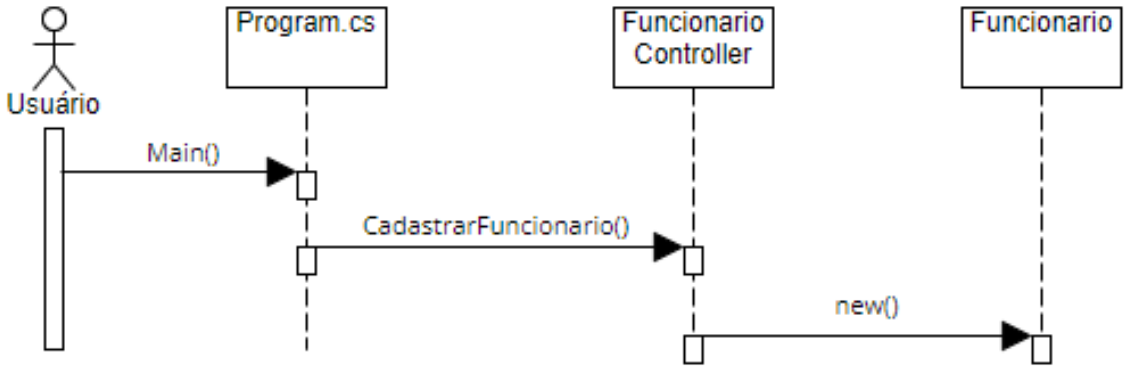
Exemplos:

- FormatException
- FileNotFoundException
- SqlException
- Etc

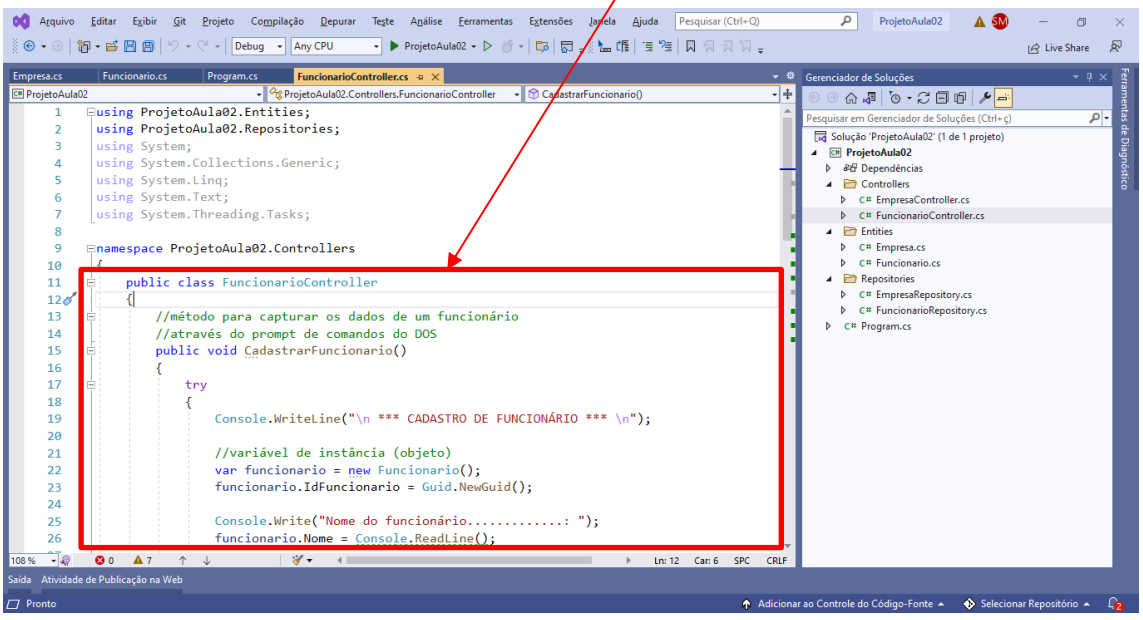


```
using ProjetoAula02.Entities;
using ProjetoAula02.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Controllers
{
    public class FuncionarioController
    {
        //método para capturar os dados de um funcionário
        //através do prompt de comandos do DOS
        public void CadastrarFuncionario()
        {
            try
            {
                Console.WriteLine("\n *** CADASTRO DE FUNCIONÁRIO *** \n");

                //variável de instância (objeto)
                var funcionario = new Funcionario();
                funcionario.IdFuncionario = Guid.NewGuid();

                Console.Write("Nome do funcionário.............: ");
                funcionario.Nome = Console.ReadLine();

                Console.Write("CPF do funcionário..............: ");
                funcionario.Cpf = Console.ReadLine();

                Console.Write("Matrícula do funcionário........: ");
                funcionario.Matricula = Console.ReadLine();

                Console.Write("Data de admissão do funcionário.: ");
                funcionario.DataAdmissao
                          = DateTime.Parse(Console.ReadLine());

                //inicializando a propriedade empresa da classe funcionario
                funcionario.Empresa = new Empresa();
                funcionario.Empresa.IdEmpresa = Guid.NewGuid();

                Console.Write("Nome fantasia da empresa........: ");
                funcionario.Empresa.NomeFantasia = Console.ReadLine();

                Console.Write("Razão Social da empresa.........: ");
                funcionario.Empresa.RazaoSocial = Console.ReadLine();

                Console.Write("CNPJ da empresa.................: ");
                funcionario.Empresa.Cnpj = Console.ReadLine();

            }
            catch (Exception e)
            {
                Console.WriteLine("\nFalha ao cadastrar funcionário!");
                Console.WriteLine($"Erro: {e.Message}");
            }
        }
    }
}
```

Agora, vamos instanciar a classe FuncionarioController no método Main da classe **Program.cs** de forma a executar a rotina de cadastro de funcionário:

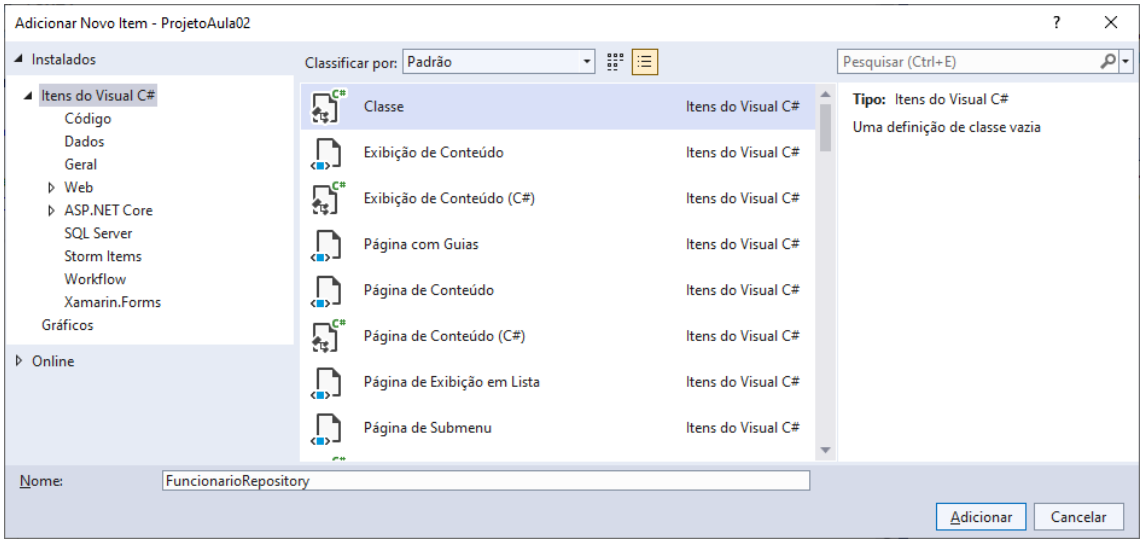



```
//localização da classe dentro do projeto
using ProjetoAula02.Controllers;

namespace ProjetoAula02
{
    //definição da classe
    public class Program
    {
        //método para executar o projeto (inicialização)
        public static void Main(string[] args)
        {
            var funcionarioController = new FuncionarioController();
            funcionarioController.CadastrarFuncionario();

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



Agora, vamos desenvolver uma classe de Repositório para gravarmos os dados do Funcionário em um arquivo de extensão .txt

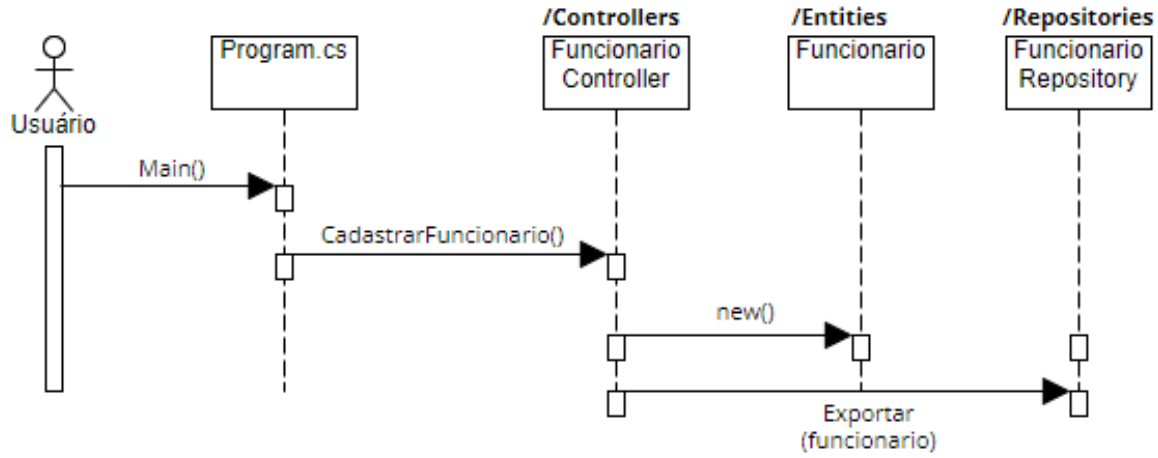
/Repositories/**FuncionarioRepository.cs**




```
using ProjetoAula02.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Repositories
{
    public class FuncionarioRepository
    {
        //método para exportar os dados do funcionário para arquivo
        public void Exportar(Funcionario funcionario)
        {
            //criando um arquivo
            using (var streamWriter = new StreamWriter
             ($"c:\\temp\\funcionario_{funcionario.IdFuncionario}.txt"))
            {
                //escrevendo os dados do funcionário no arquivo..
                streamWriter.WriteLine($"ID.....................:
                         {funcionario.IdFuncionario}");

                streamWriter.WriteLine($"NOME DO FUNCIONÁRIO..:
                         {funcionario.Nome}");

                streamWriter.WriteLine($"CPF....................:
                         {funcionario.Cpf}");

                streamWriter.WriteLine($"MATRICULA.............:
                         {funcionario.Matricula}");

                streamWriter.WriteLine($"DATA DE ADMISSÃO.....:
                         {funcionario.DataAdmissao}");
```

```
                streamWriter.WriteLine($"ID DA EMPRESA........:
                        {funcionario.Empresa.IdEmpresa}");

                streamWriter.WriteLine($"NOME FANTASIA........:
                        {funcionario.Empresa.NomeFantasia}");

                streamWriter.WriteLine($"RAZÃO SOCIAL.........:
                        {funcionario.Empresa.RazaoSocial}");

                streamWriter.WriteLine($"CNPJ.................:
                        {funcionario.Empresa.Cnpj}");
            }
        }
    }
}
```

Dessa forma iremos, dentro da classe **FuncionarioController**, instanciar e executar o método Exportar da classe **FuncionarioRepository**:

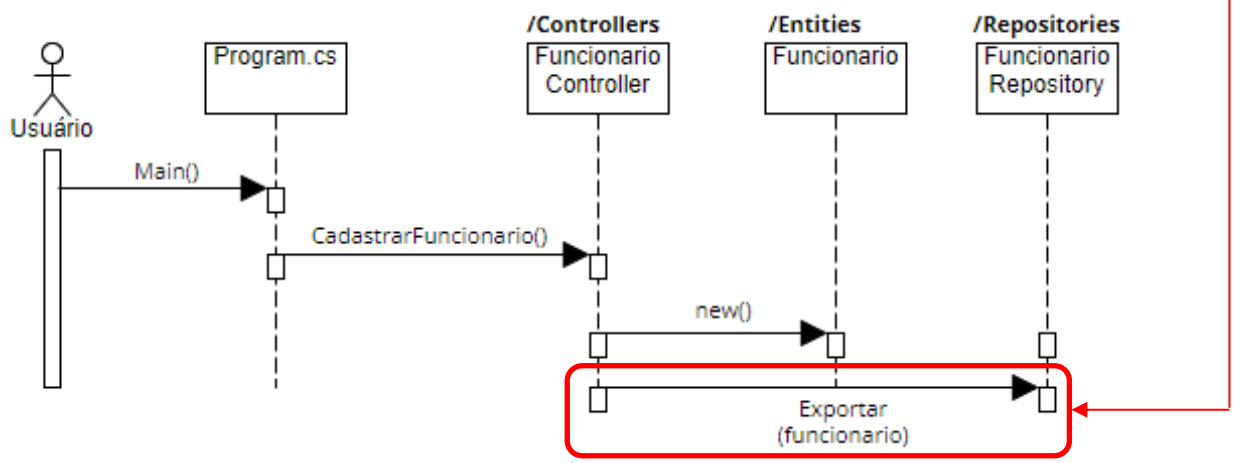


/Controllers/**FuncionarioController.cs**

Executando a exportação de dados do funcionário.

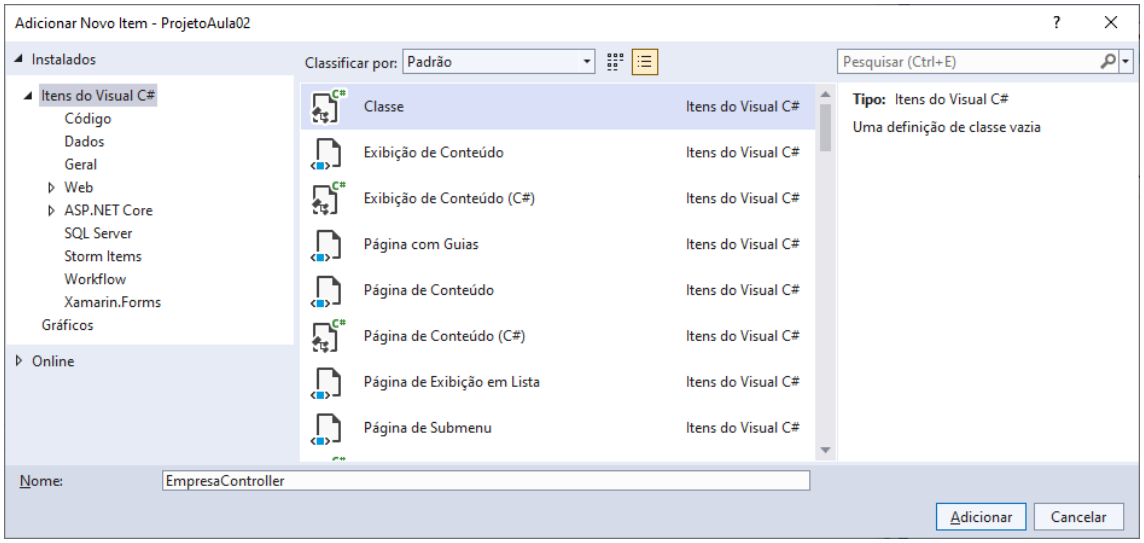
```
using ProjetoAula02.Entities;
using ProjetoAula02.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Controllers
{
    public class FuncionarioController
    {
        //método para capturar os dados de um funcionário
        //através do prompt de comandos do DOS
        public void CadastrarFuncionario()
        {
            try
            {
                Console.WriteLine("\n *** CADASTRO DE FUNCIONÁRIO *** \n");
```

```
                //variável de instância (objeto)
                var funcionario = new Funcionario();
                funcionario.IdFuncionario = Guid.NewGuid();

                Console.Write("Nome do funcionário.............: ");
                funcionario.Nome = Console.ReadLine();

                Console.Write("CPF do funcionário..............: ");
                funcionario.Cpf = Console.ReadLine();

                Console.Write("Matrícula do funcionário........: ");
                funcionario.Matricula = Console.ReadLine();

                Console.Write("Data de admissão do funcionário.: ");
                funcionario.DataAdmissao
                          = DateTime.Parse(Console.ReadLine());

                //inicializando a propriedade empresa da classe funcionario
                funcionario.Empresa = new Empresa();
                funcionario.Empresa.IdEmpresa = Guid.NewGuid();

                Console.Write("Nome fantasia da empresa........: ");
                funcionario.Empresa.NomeFantasia = Console.ReadLine();

                Console.Write("Razão Social da empresa.........: ");
                funcionario.Empresa.RazaoSocial = Console.ReadLine();

                Console.Write("CNPJ da empresa.................: ");
                funcionario.Empresa.Cnpj = Console.ReadLine();

                //instanciando a classe FuncionarioRepository
                var funcionarioRepository = new FuncionarioRepository();
                funcionarioRepository.Exportar(funcionario);

                Console.WriteLine("\nFUNCIONÁRIO CADASTRADO COM SUCESSO!");
            }
            catch (Exception e)
            {
                Console.WriteLine("\nFalha ao cadastrar funcionário!");
                Console.WriteLine($"Erro: {e.Message}");
            }
        }
    }
}
```



Vamos desenvolver um controlador que também realizar um fluxo de cadastro para Empresa, chamado: **EmpresaController**.

/Controllers/**EmpresaController.cs**

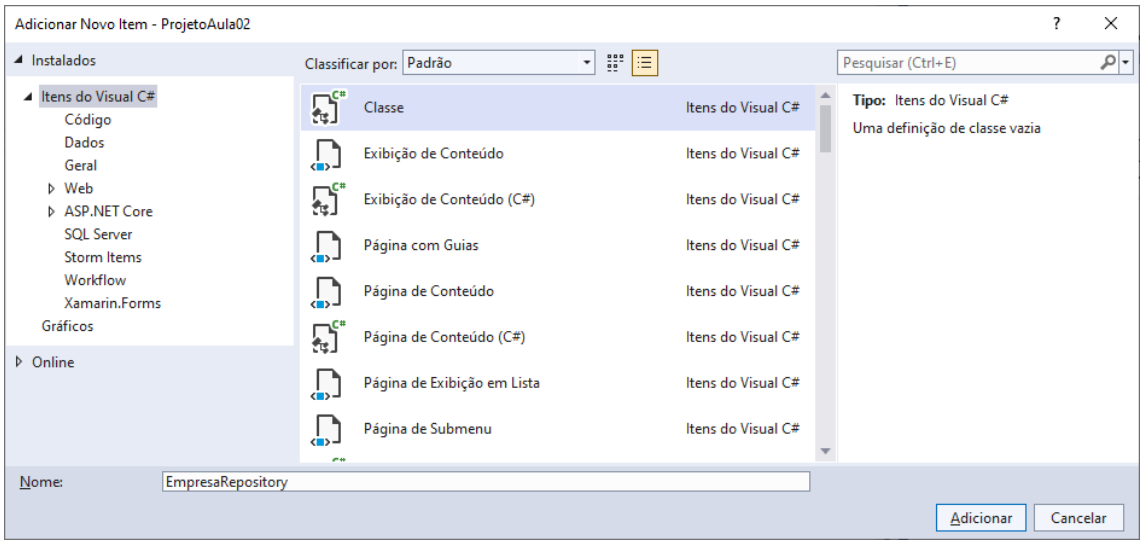



```
using ProjetoAula02.Entities;
using ProjetoAula02.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Controllers
{
    public class EmpresaController
    {
        public void CadastrarEmpresa()
        {
            try
            {
                Console.WriteLine("\n *** CADASTRO DE EMPRESA *** \n");

                var empresa = new Empresa();
                empresa.IdEmpresa = Guid.NewGuid();

                Console.Write("Nome fantasia da empresa.....: ");
                empresa.NomeFantasia = Console.ReadLine();

                Console.Write("Razão Social da empresa......: ");
                empresa.RazaoSocial = Console.ReadLine();

                Console.Write("CNPJ da empresa..............: ");
                empresa.Cnpj = Console.ReadLine();
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine("\nFalha ao cadastrar empresa!");
                Console.WriteLine($"Erro: {e.Message}");
            }
        }
    }
}
```

Agora, vamos adicionar uma classe chamada **EmpresaRepository** de forma que possamos gravar os dados da empresa em um arquivo de extensão .txt

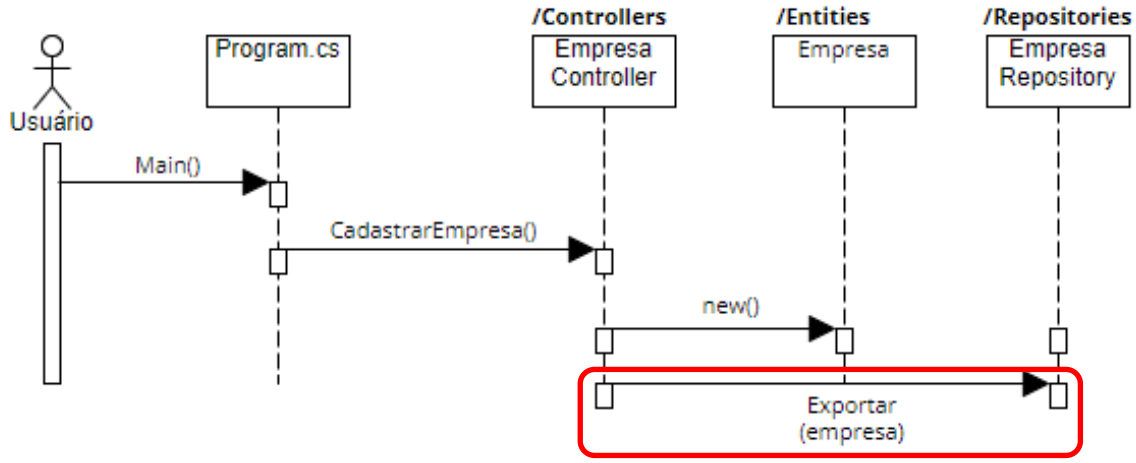
/Repositories/**EmpresaRepository.cs**




```
using ProjetoAula02.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Repositories
{
    public class EmpresaRepository
    {
        public void Exportar(Empresa empresa)
        {
            using (var streamWriter = new StreamWriter
                   ($"c:\\temp\\empresa_{empresa.IdEmpresa}.txt"))
            {
                streamWriter.WriteLine
                   ($"ID.............: {empresa.IdEmpresa}");

                streamWriter.WriteLine
                   ($"NOME FANTASIA..: {empresa.NomeFantasia}");
```

```
                streamWriter.WriteLine
                    ($"RAZÃO SOCIAL...: {empresa.RazaoSocial}");

                streamWriter.WriteLine
                    ($"CNPJ..........: {empresa.Cnpj}");
            }
        }
    }
}
```

Voltando na classe **EmpresaController** e executando a exportação de dados para arquivo .txt através da classe **EmpresaRepository**:




```
using ProjetoAula02.Entities;
using ProjetoAula02.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ProjetoAula02.Controllers
{
    public class EmpresaController
    {
        public void CadastrarEmpresa()
        {
            try
            {
                Console.WriteLine("\n *** CADASTRO DE EMPRESA *** \n");

                var empresa = new Empresa();
                empresa.IdEmpresa = Guid.NewGuid();

                Console.Write("Nome fantasia da empresa.....: ");
                empresa.NomeFantasia = Console.ReadLine();

                Console.Write("Razão Social da empresa......: ");
                empresa.RazaoSocial = Console.ReadLine();
```

```
                Console.Write("CNPJ da empresa.............: ");
                empresa.Cnpj = Console.ReadLine();

                var empresaRepository = new EmpresaRepository();
                empresaRepository.Exportar(empresa);

                Console.WriteLine("EMPRESA CADASTRADA COM SUCESSO!");
            }
            catch (Exception e)
            {
                Console.WriteLine("\nFalha ao cadastrar empresa!");
                Console.WriteLine($"Erro: {e.Message}");
            }
        }
    }
}
```

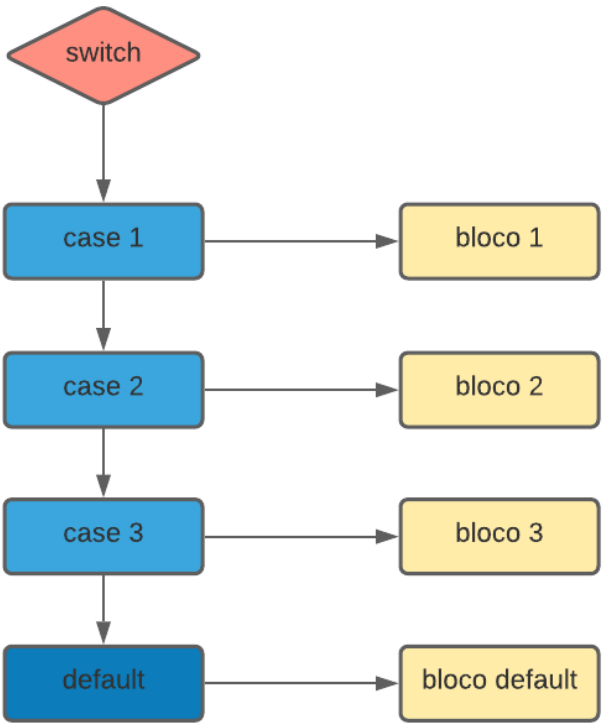
Por último, vamos modificar a classe Program.cs para executar os dois tipos de cadastro: Funcionários ou empresas dependendo da escolha do usuário:

/Program.cs

```
//localização da classe dentro do projeto
using ProjetoAula02.Controllers;

namespace ProjetoAula02
{
    //definição da classe
    public class Program
    {
        //método para executar o projeto (inicialização)
        public static void Main(string[] args)
        {
            try
            {
                //impressão de texto
                Console.WriteLine("(1) Cadastro de empresas");
                Console.WriteLine("(2) Cadastro de funcionários");

                Console.Write("\nInforme a opção desejada..: ");
                var opcao = int.Parse(Console.ReadLine());

                switch(opcao)
                {
                    case 1:
                    var empresaController = new EmpresaController();
                    empresaController.CadastrarEmpresa();
                    break;

                    case 2:
                    var funcionarioController = new FuncionarioController();
                    funcionarioController.CadastrarFuncionario();
                    break;
```

```
                    default:
                        Console.WriteLine("Opção inválida!");
                        break;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine($"\nFalha: {e.Message}");
            }

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

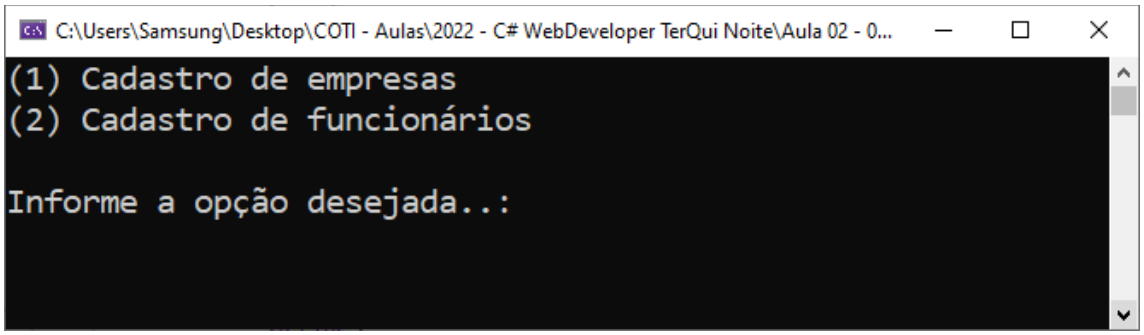
Note que utilizamos um bloco **switch / case** para verificar qual foi a opção informada pelo usuário e então decidir qual fluxo será executado pelo sistema (cadastro de empresa ou cadastro de funcionário):



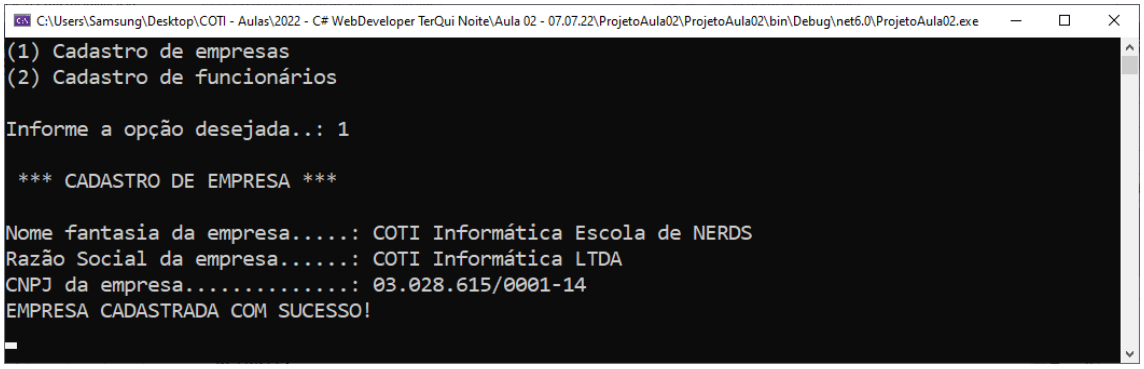

```
//localização da classe dentro do projeto
using ProjetoAula02.Controllers;

namespace ProjetoAula02
{
    //definição da classe
    public class Program
    {
        //método para executar o projeto (inicialização)
        public static void Main(string[] args)
        {
```

```
            try
            {
                //impressão de texto
                Console.WriteLine("(1) Cadastro de empresas");
                Console.WriteLine("(2) Cadastro de funcionários");

                Console.Write("\nInforme a opção desejada..: ");
                var opcao = int.Parse(Console.ReadLine());

                switch(opcao)
                {
                    case 1:
                    var empresaController = new EmpresaController();
                    empresaController.CadastrarEmpresa();
                    break;

                    case 2:
                    var funcionarioController = new FuncionarioController();
                    funcionarioController.CadastrarFuncionario();
                    break;

                    default:
                        Console.WriteLine("Opção inválida!");
                        break;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine($"\nFalha: {e.Message}");
            }

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

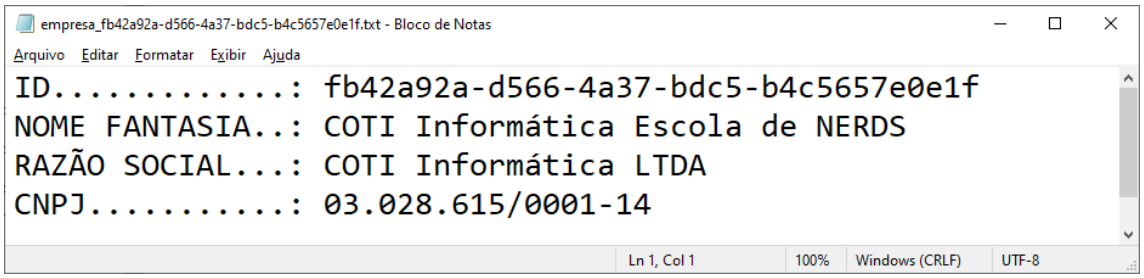
Executando:



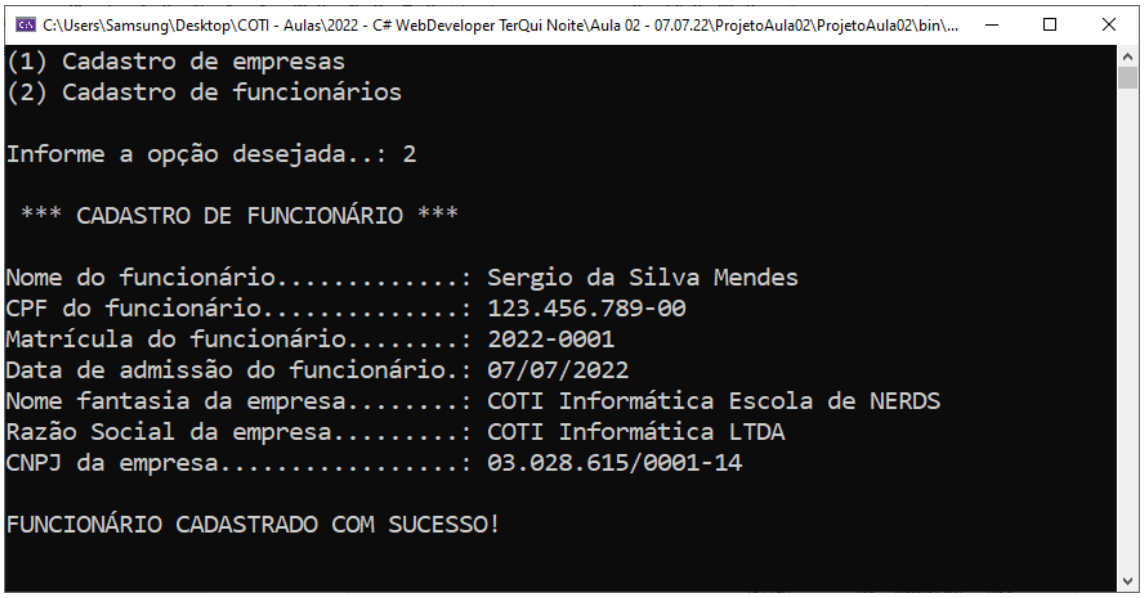
Cadastrando empresa:



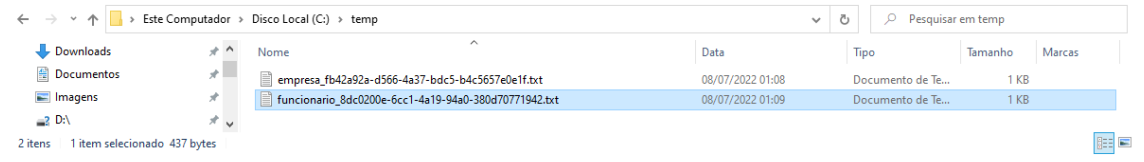
Arquivo gerado:

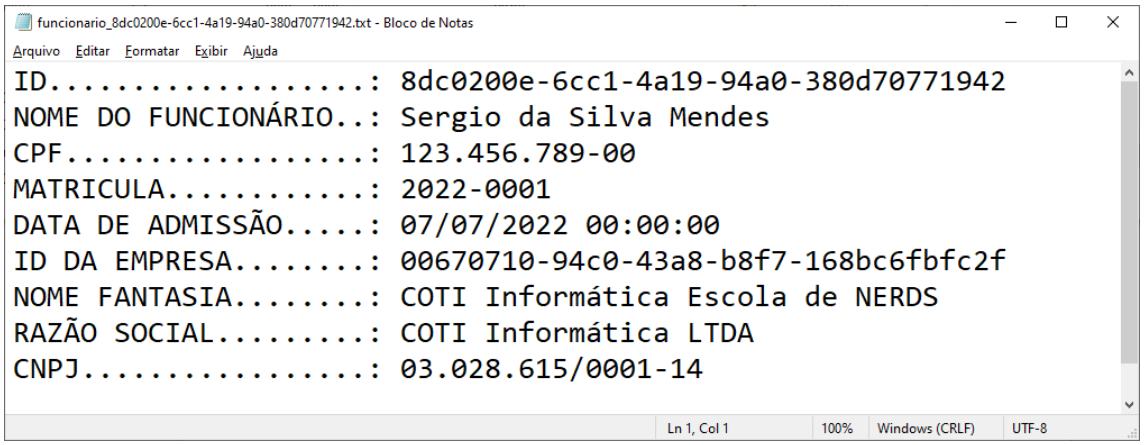


Cadastrando funcionário:

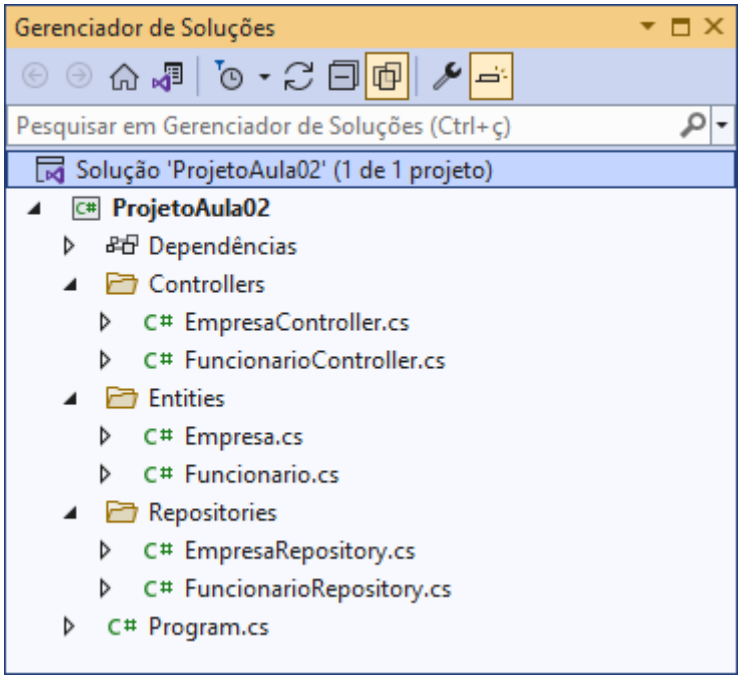


Arquivo gerado:





Estrutura do projeto e separação em camadas:



SRP – Princípio de responsabilidade única.

Define que cada classe em um projeto deve ter uma única responsabilidade, mantendo a coesão de forma que os métodos de uma classe sejam voltados para resolver apenas 1 problema específico.

Na programação, o Princípio da responsabilidade única declara que cada módulo ou classe deve ter responsabilidade sobre uma única parte da funcionalidade fornecida pelo software.

Você pode ter ouvido a citação: " Faça uma coisa e faça bem ".
Isso se refere ao princípio da responsabilidade única.
No artigo citado acima, Robert C. Martin define uma responsabilidade como um "motivo para mudar" e conclui que uma classe ou módulo deve ter um e apenas um motivo para ser alterado.

Como esse princípio nos ajuda a criar um software melhor? Vamos ver alguns dos seus benefícios:

- **Teste** — Uma classe com uma responsabilidade terá muito menos casos de teste
- **Menor acoplamento** — menos funcionalidade em uma única classe terá menos dependências
- **Organização** — Classes menores e bem-organizadas são mais fáceis de pesquisar do que as classes monolíticas