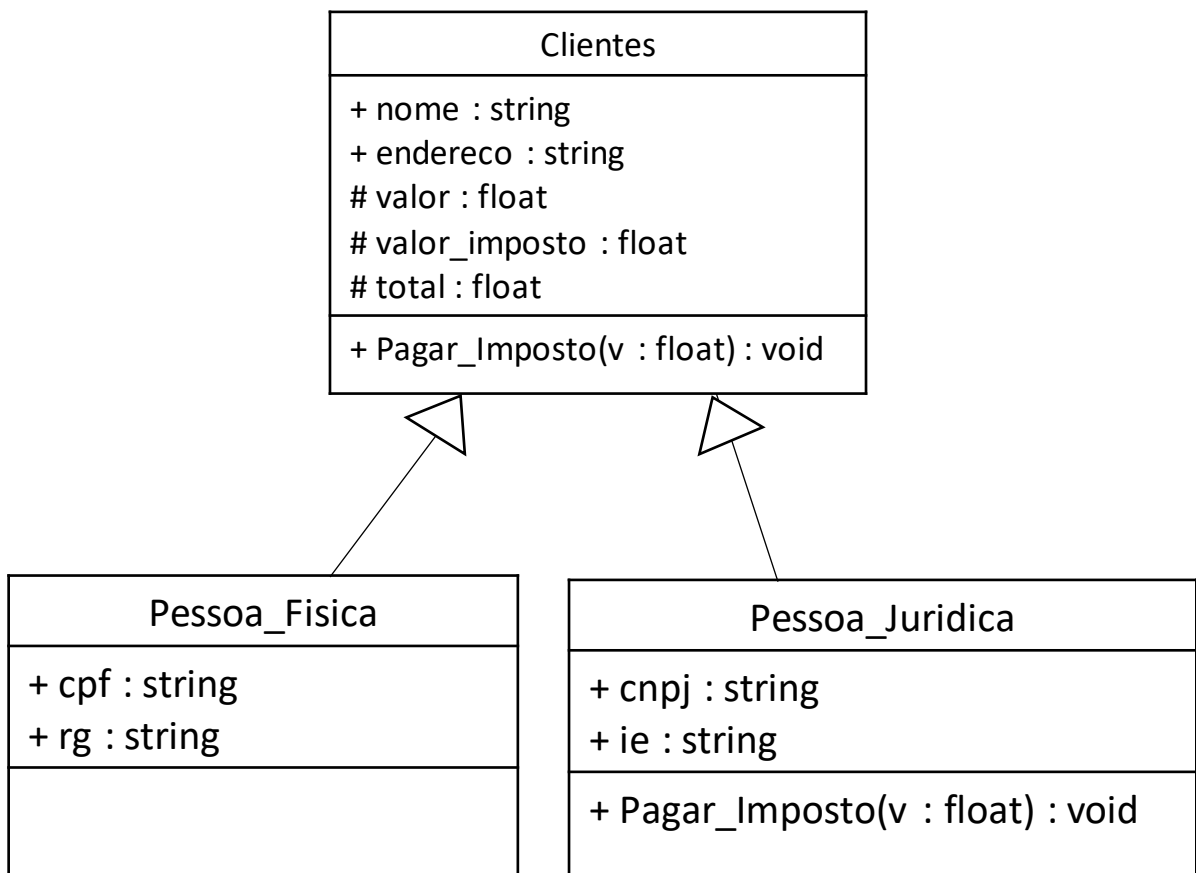


Atividade

Sistema de controle de clientes

Nesta atividade, vamos desenvolver um sistema de controle de clientes baseado no diagrama de classes a seguir.



Você deve analisar o diagrama de classes proposto e identificar as funcionalidades dele, para que o sistema atenda aos seguintes requisitos:

- a entrada de dados deverá ser efetuada via console e informada pelo usuário;
- o sistema deve ser baseado nos conceitos da Programação Orientada a Objetos (POO);
- os seguintes conceitos de POO devem ser implementados durante o desenvolvimento do sistema:
 - abstração;
 - herança;
 - polimorfismo;
 - encapsulamento;
- a cada execução, o sistema deve determinar se o cliente é pessoa física ou jurídica;
- o cálculo do imposto a pagar correspondente ao tipo de cliente deve ser realizado corretamente;
- os dados do cliente e resultados dos cálculos devem ser apresentados em tela.

Ferramentas

Para realizar esta atividade, você vai precisar de:

- uma pasta no seu disco rígido;
- Visual Studio Code;
- DotNet (SDK);
- extensão para C#.

Análise do diagrama

Os clientes podem ser pessoa física (com CPF e RG) ou pessoa jurídica (com CNPJ e Inscrição Estadual).

O diagrama de classes, então, mostra que a classe **Clientes** é a classe pai das subclasses **Pessoa_Fisica** e **Pessoa_Juridica**. Os atributos e métodos da classe-pai são herdados por ambas as classes-filhas.

A classe **Pessoa_Fisica** tem, ainda, os atributos específicos **CPF** e **RG**, e a classe **Pessoa_Juridica** tem os atributos específicos **CNPJ** e **IE**.

O imposto para pessoa física é de 10% sobre o valor e, para pessoa jurídica, é 20%; e 90% dos clientes são pessoas físicas.

O método **Pagar_Imposto** da classe-pai **Clientes** é reescrito (**override**) na classe **Pessoa_Juridica**, devido ao percentual diferente de imposto a ser pago.

Importante

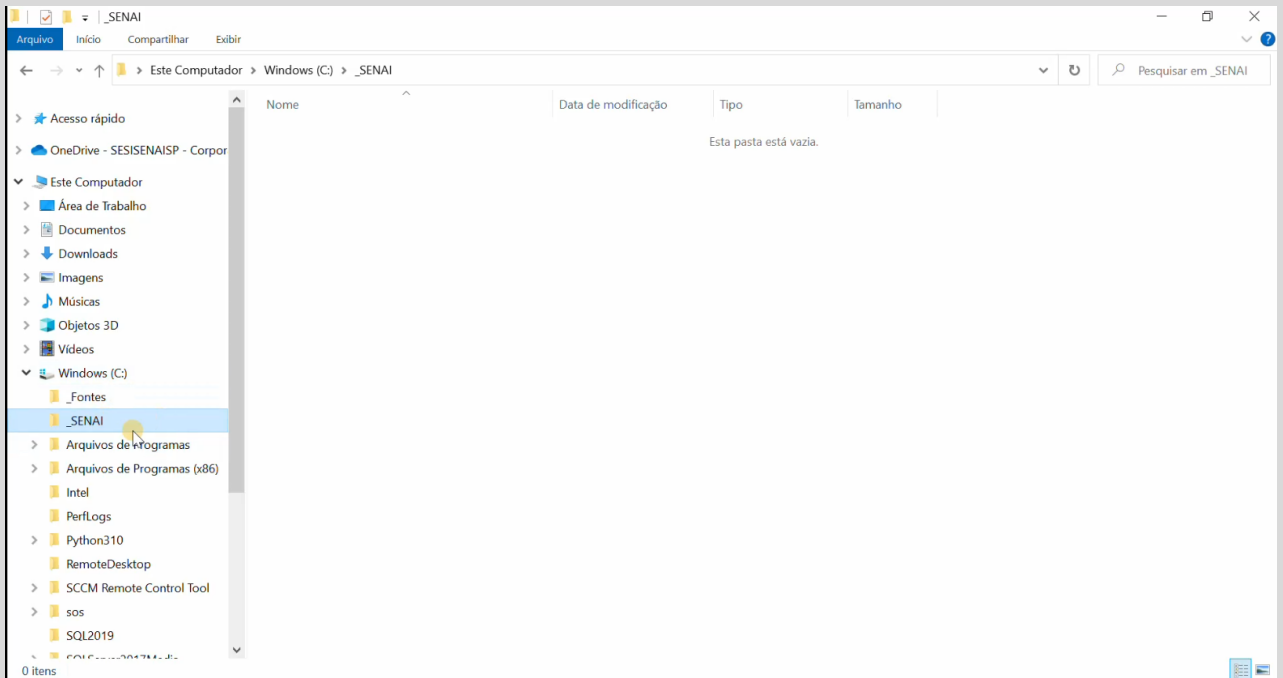
Lembre-se dos modificadores de acesso:

- + → público
- # → protected
- - → private

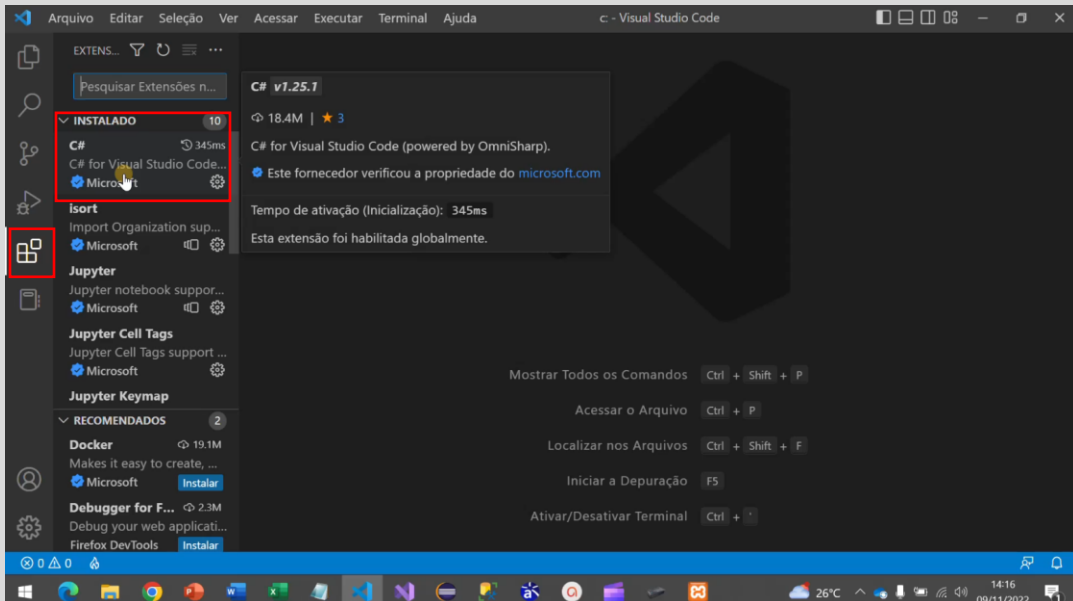


Preparação do sistema

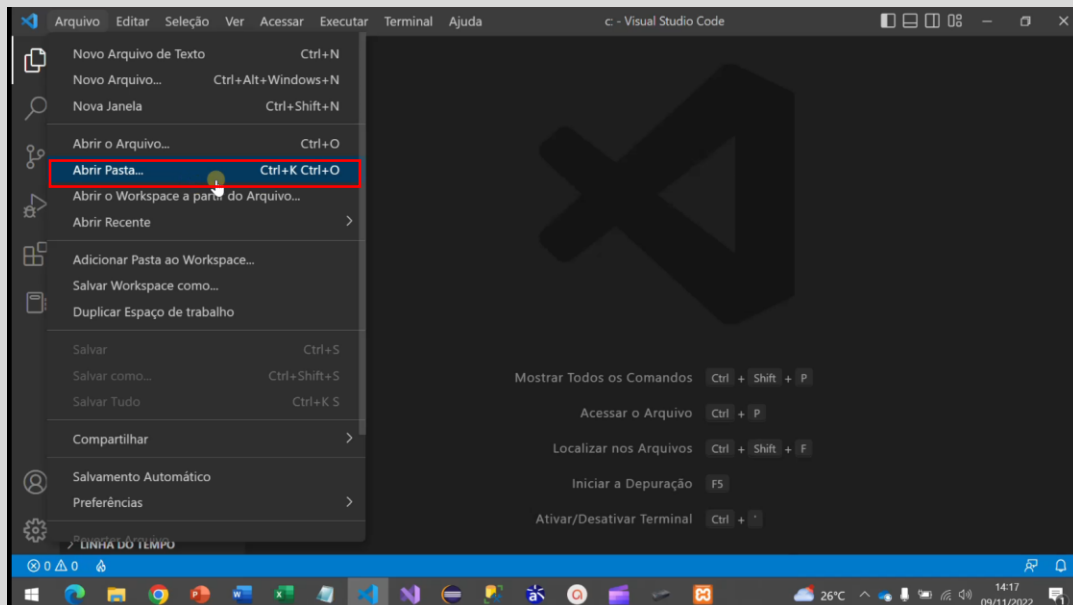
1. Crie uma pasta para salvar seu projeto. No nosso exemplo, a pasta é a **_SENAI**.



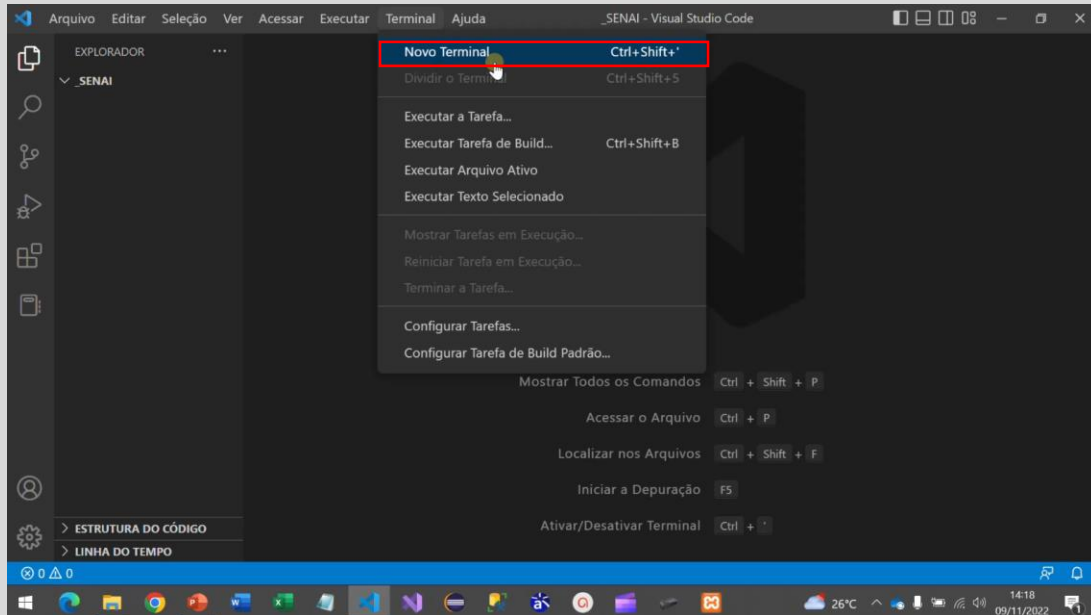
2. No VSC, verifique se a extensão para C# está instalada, clicando no **quinto ícone** no menu lateral esquerdo.



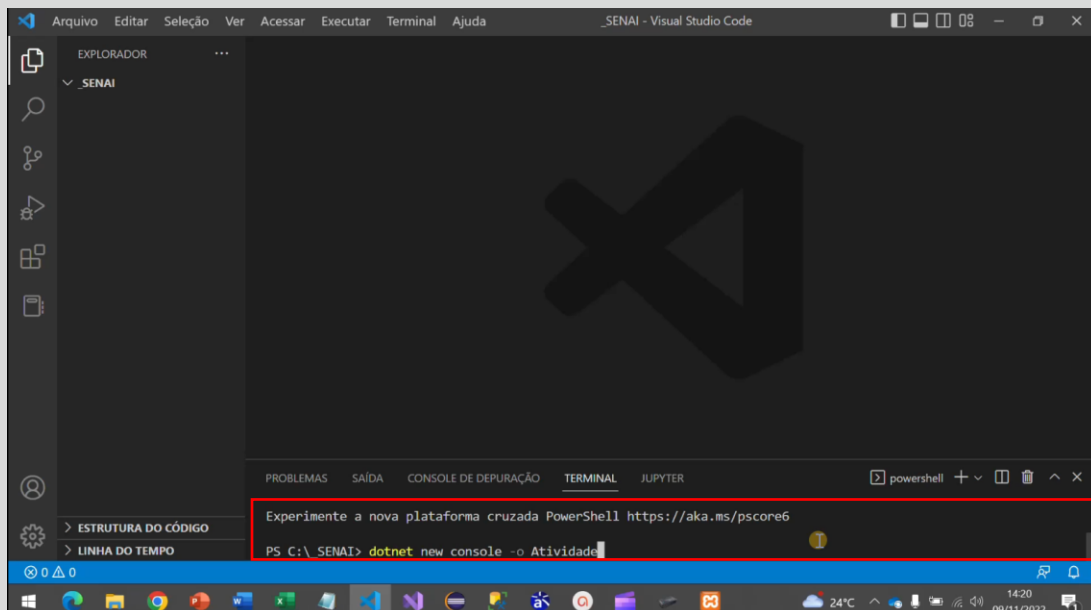
3. Abra a pasta do projeto, clicando em **Arquivo > Abrir Pasta...** no menu superior.



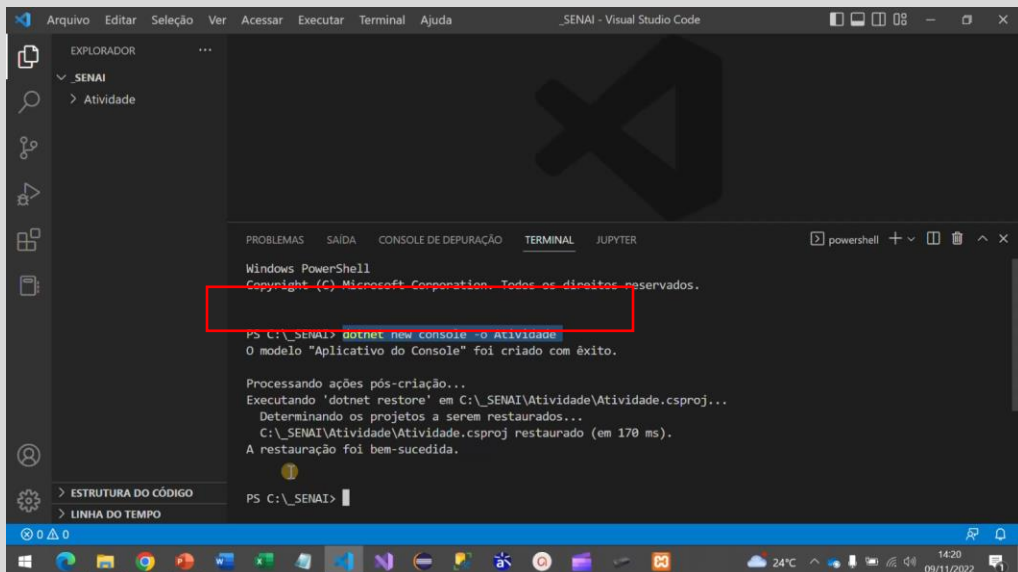
4. Com a pasta `_SENAI` aberta, abra um terminal no VSC.



5. No terminal aberto, crie seu projeto Atividade, digitando: **dotnet new console** –o **Atividade** e dê **Enter**.



6. Dentro do terminal, o sistema vai mostrar mensagem de êxito.



The screenshot shows the Visual Studio Code interface with the terminal pane active. The terminal output displays the command `PS C:_SENAI> dotnet new console -o Atividade` and its successful execution, including the creation of the project and the restoration of packages. A red rectangle highlights the command and its immediate output.

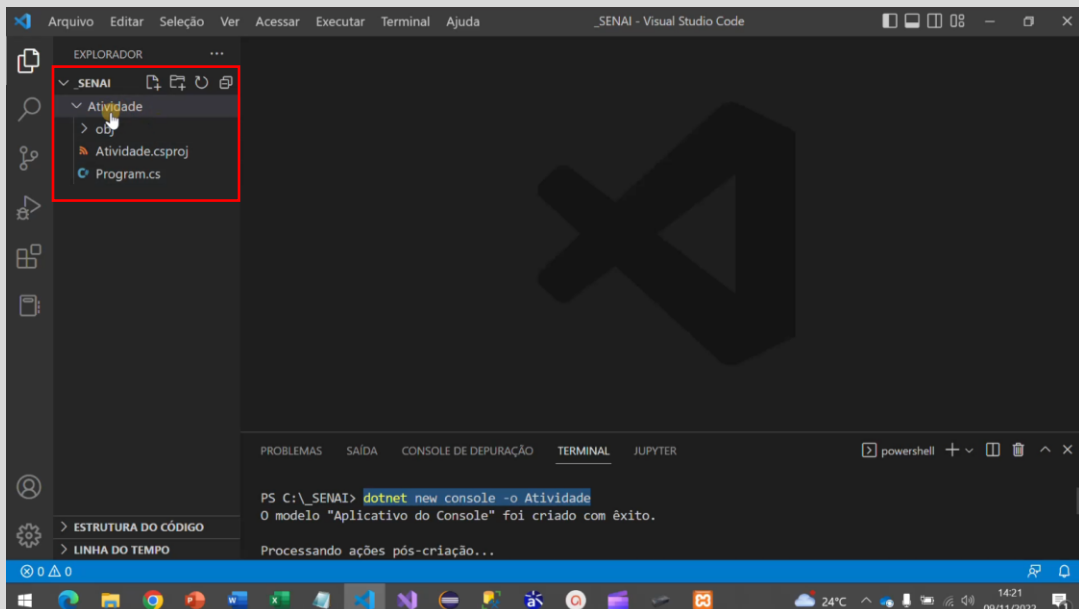
```
Windows PowerShell
Copyright (c) Microsoft Corporation. Todos os direitos reservados.

PS C:\_SENAI> dotnet new console -o Atividade
O modelo "Aplicativo do Console" foi criado com êxito.

Processando ações pós-criação...
Executando 'dotnet restore' em C:\_SENAI\Atividade\Atividade.csproj...
Determinando os projetos a serem restaurados...
C:\_SENAI\Atividade\Atividade.csproj restaurado (em 170 ms).
A restauração foi bem-sucedida.

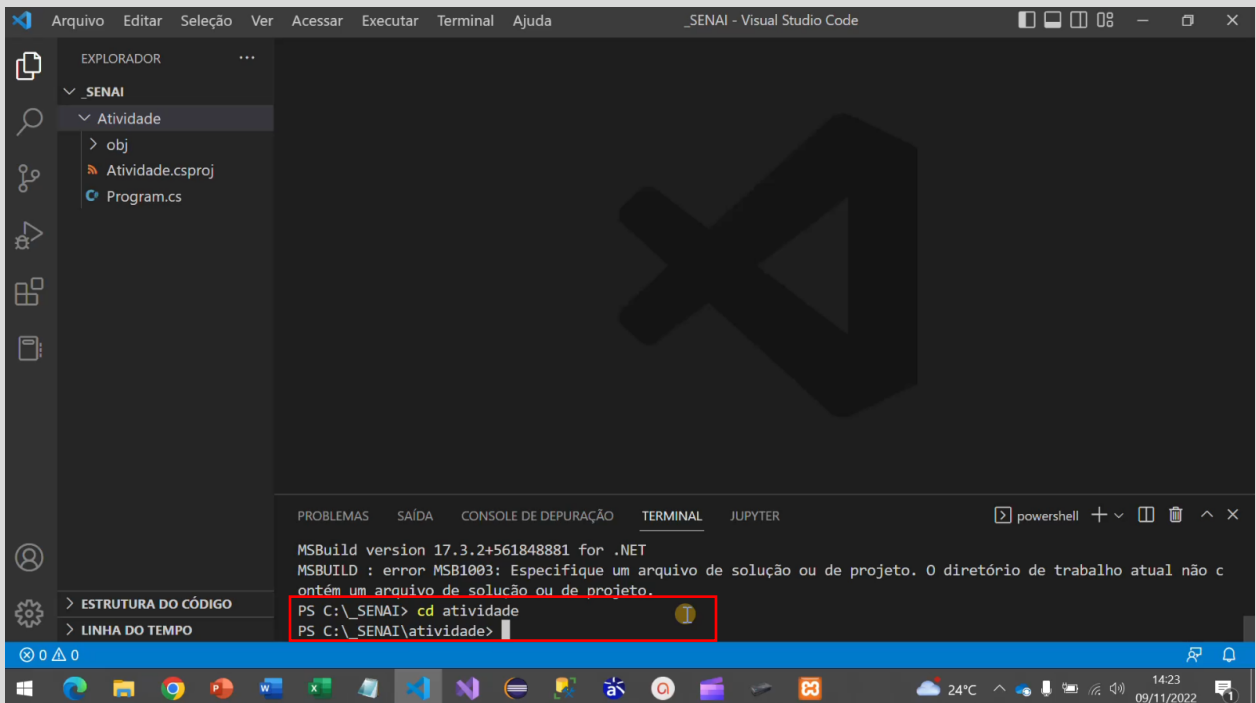
PS C:\_SENAI>
```

7. O sistema criará automaticamente a estrutura do projeto dentro de **_SENAI**: uma pasta **Atividade**, com uma subpasta **obj**, com os arquivos **Atividade.csproj** e **Program.cs**.



8. O próximo passo é criar os módulos de execução para compilação, por meio do comando **dotnet build**.

Antes de digitar o comando, você deve ir para a pasta **Atividade** (estamos na pasta **_SENAI**). Digite **cd atividade** e dê Enter.



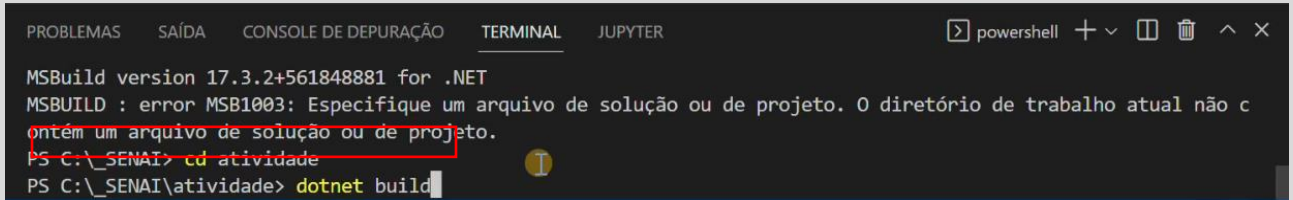
```
MSBuild version 17.3.2+561848881 for .NET
MSBUILD : error MSB1003: Especifique um arquivo de solução ou de projeto. O diretório de trabalho atual não contém um arquivo de solução ou de projeto.
PS C:\_SENAI> cd atividade
PS C:\_SENAI\atividade>
```

Você sabia?

O **cd** do comando **cd atividade** quer dizer “change directory” ou, em português, “mudar de diretório”.

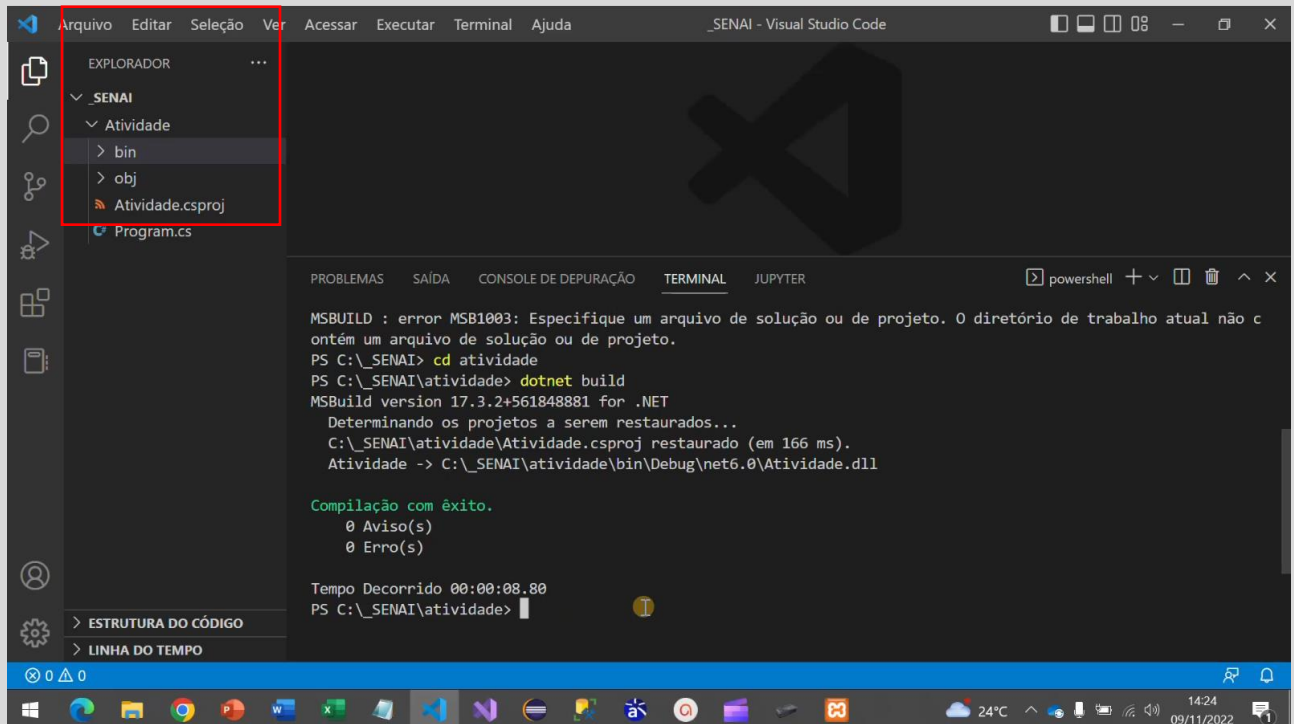


9. Em seguida, digite **dotnet build** e dê **Enter**.



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  JUPYTER
MSBuild version 17.3.2+561848881 for .NET
MSBUILD : error MSB1003: Especifique um arquivo de solução ou de projeto. O diretório de trabalho atual não contém um arquivo de solução ou de projeto.
PS C:\_SENAI> cd atividade
PS C:\_SENAI\atividade> dotnet build
```

O sistema vai criar automaticamente a pasta **bin** e a subpasta **Debug**, além de alguns elementos necessários para a execução e compilação do sistema.



```
Arquivo  Editar  Seleção  Ver  Acessar  Executar  Terminal  Ajuda  _SENAI - Visual Studio Code
EXPLORADOR
  _SENAI
    Atividade
      > bin
      > obj
      Atividade.csproj
      Program.cs

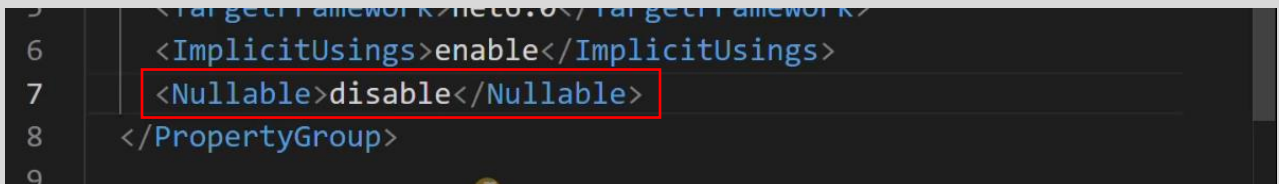
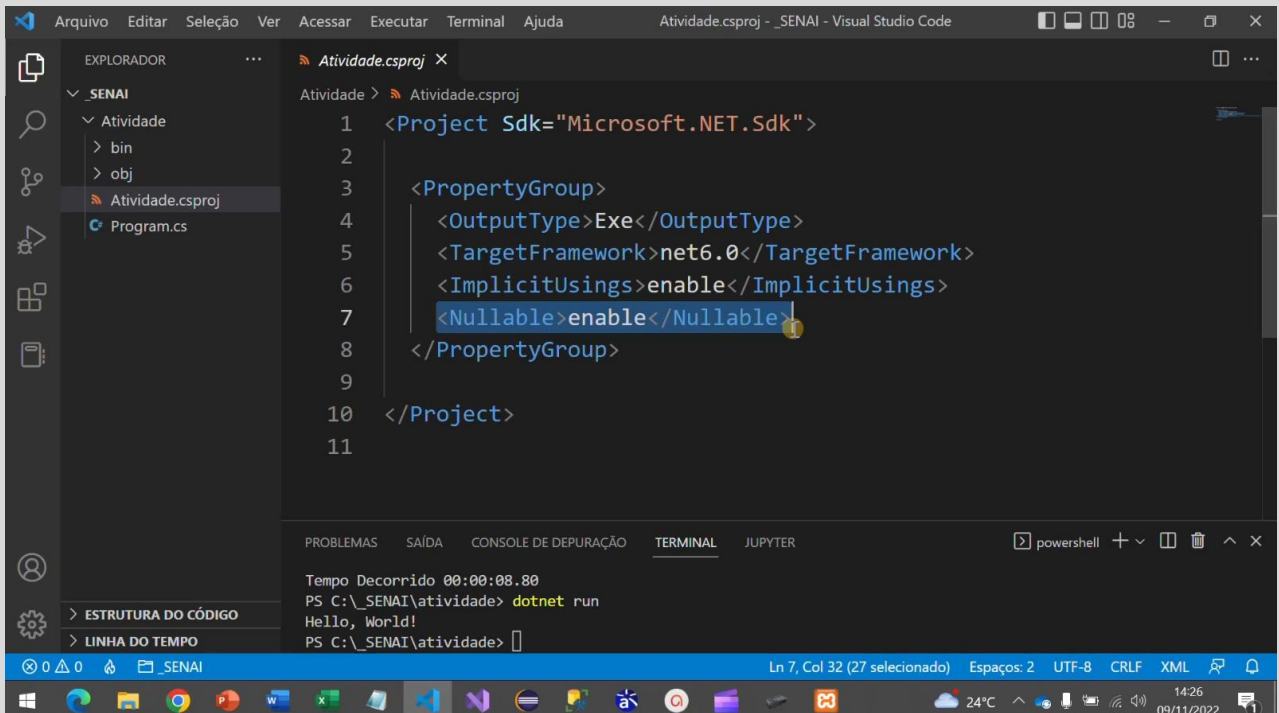
PROBLEMAS  SAÍDA  CONSOLE DE DEPUÇÃO  TERMINAL  JUPYTER
MSBUILD : error MSB1003: Especifique um arquivo de solução ou de projeto. O diretório de trabalho atual não contém um arquivo de solução ou de projeto.
PS C:\_SENAI> cd atividade
PS C:\_SENAI\atividade> dotnet build
MSBuild version 17.3.2+561848881 for .NET
Determinando os projetos a serem restaurados...
C:\_SENAI\atividade\Atividade.csproj restaurado (em 166 ms).
Atividade -> C:\_SENAI\atividade\bin\Debug\net6.0\Atividade.dll

Compilação com êxito.
  0 Aviso(s)
  0 Erro(s)

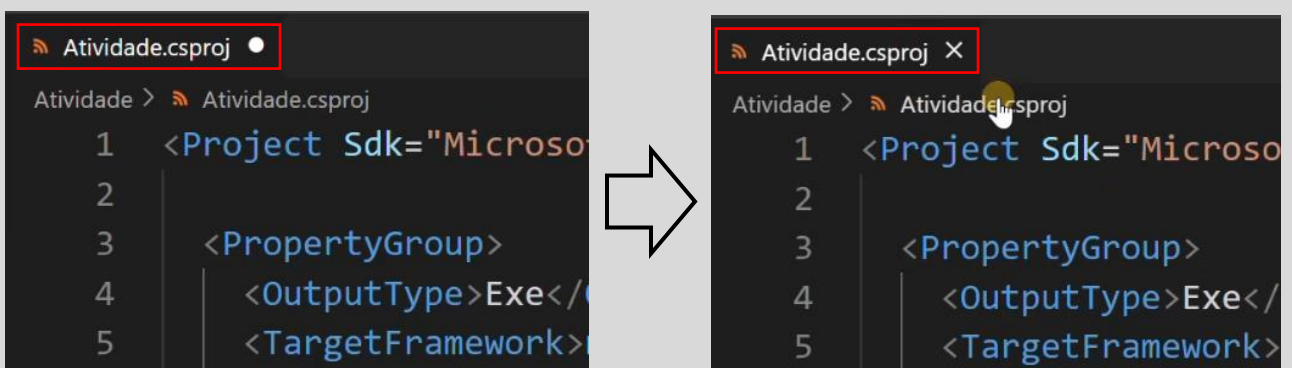
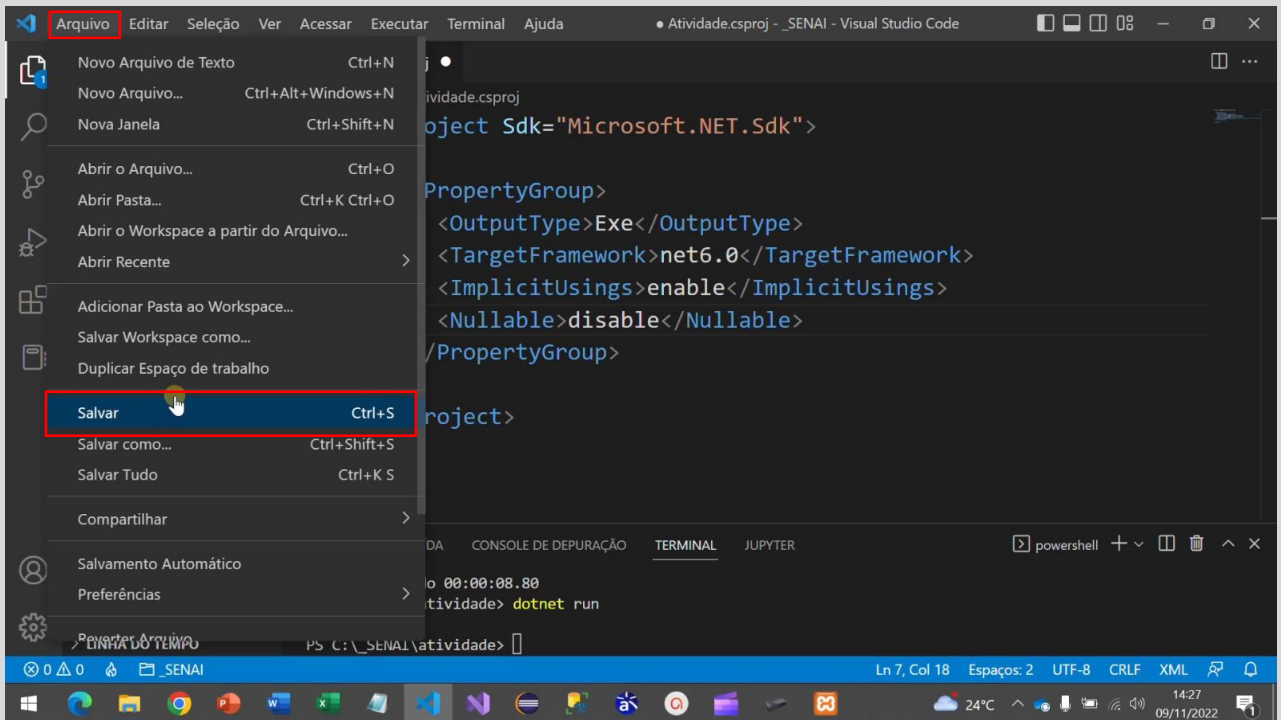
Tempo Decorrido 00:00:08.80
PS C:\_SENAI\atividade>
```

Implementação de classes

1. Selecione o arquivo **Atividade.csproj** e desabilite os alertas em amarelo da compilação, trocando o termo **enable** por **disable** em **Nullable**. Os alertas não influenciam na compilação.

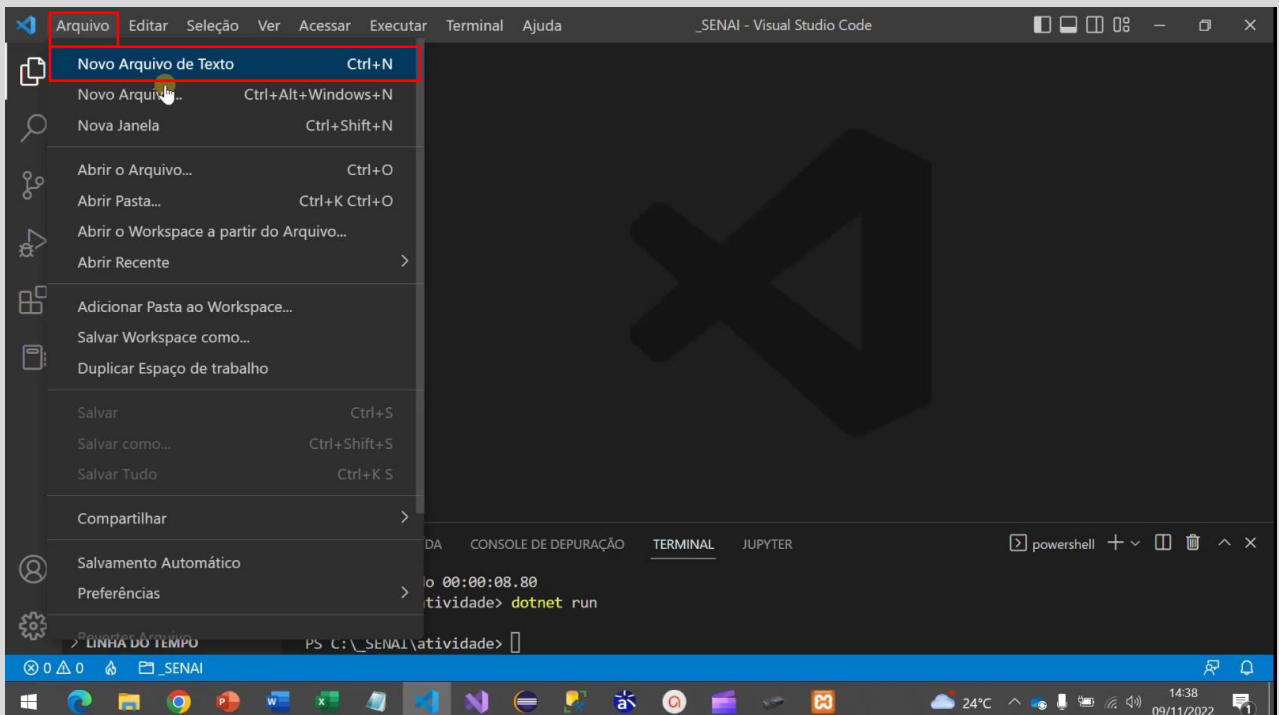


2. Salve a alteração, clicando em **Arquivo** e depois em **Salvar** no menu superior, ou usando o atalho **ctrl+s**. Após salvar, note que o círculo ao lado do nome do arquivo passará a ser um x e você já pode fechar o arquivo.

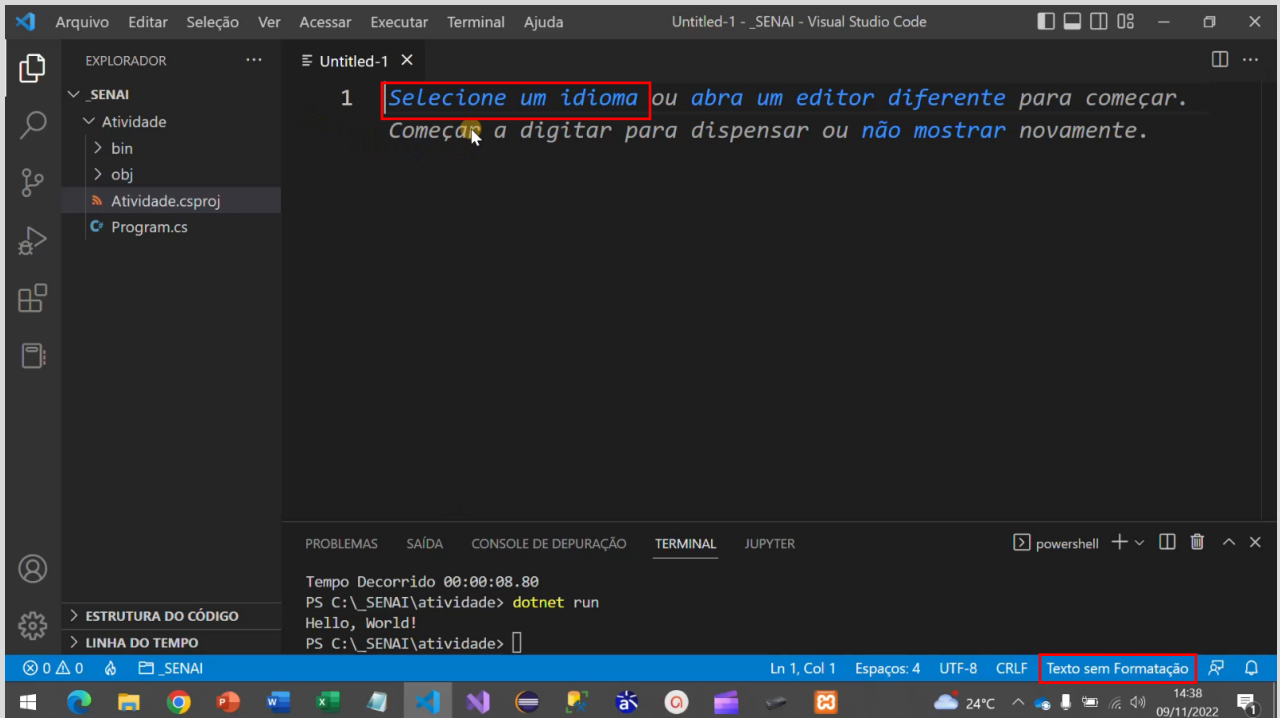


3. Vamos iniciar o desenvolvimento do sistema, propriamente dito, baseado no diagrama de classes. A primeira classe a ser implementada será a classe **Clientes**.

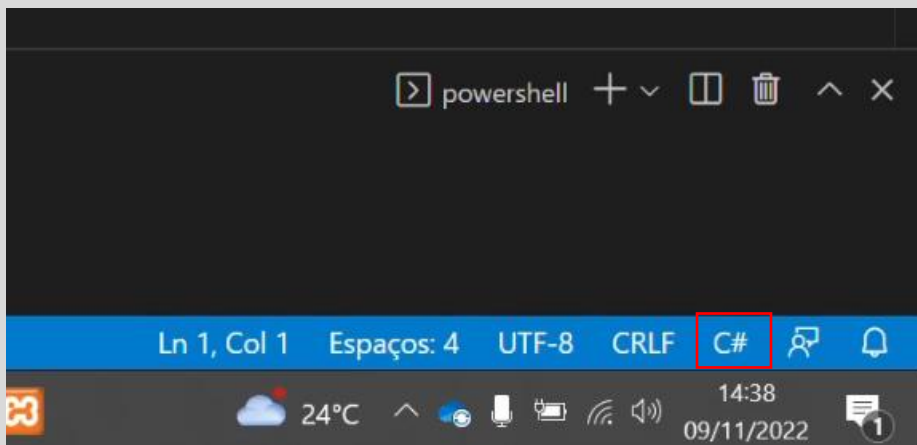
No menu superior, clique em **Arquivo** e depois em **Novo Arquivo de Texto**.



4. Clique sobre o texto destacado **Selecione um idioma** e selecione a linguagem **C#**.



Note, no canto inferior direito, onde aparecem informações do arquivo em uso, que o termo **Texto sem Formatação** passou a ser **C#**.



Dica!

Selecionar a linguagem otimiza o desenvolvimento, pois o VSC reconhece palavras-chaves e sugere a sintaxe pré-formatada, dando mais fluidez à implementação, minimizando erros de digitação e facilitando a organização do código.



Importante

Namespace é uma palavra-chave usada para organizar o código, separando-o por níveis.



Como boa prática, use sempre uma letra maiúscula para o nome das classes.

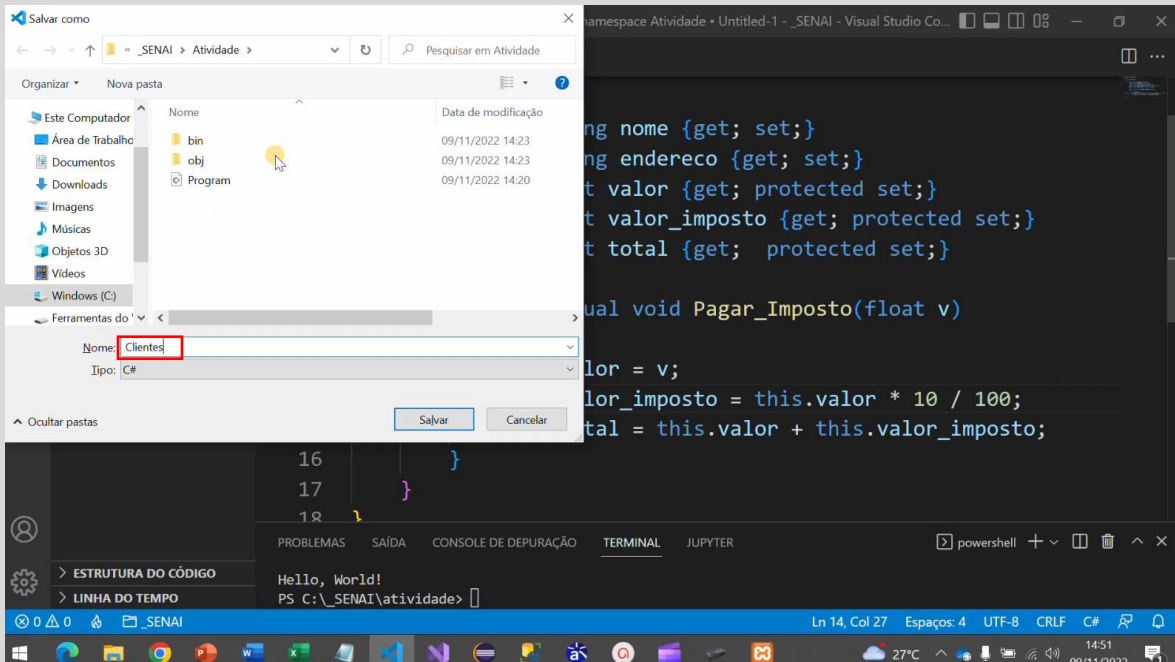
5. Para implementar a classe **Clientes** conforme o diagrama, digite o código a seguir no arquivo recém-criado.

```
namespace Atividade
{
    class Clientes
    {
        public string nome {get; set;}
        public string endereco {get; set;}
        public float valor {get; protected set;}
        public float valor_imposto {get; protected set;}
        public float total {get; protected set;}
        public virtual void Pagar_Imposto(float v)
        {
            this.valor = v;
            this.valor_imposto = this.valor * 10 / 100;
            this.total = this.valor + this.valor_imposto;
        }
    }
}
```

No caso dos atributos, todos são públicos. Os atributos **nome** e **endereco** são variáveis do tipo string, enquanto valor, valor_imposto e total são variáveis numéricas do tipo float. Em valor, valor_imposto e total, temos que proteger a alteração dos valores com **protected set**.

O método **Pagar_Imposto** é público com retorno indeterminado (void) e tem como argumento uma variável numérica do tipo float nomeada como **v**. O método precisa ter o modificador de acesso virtual, pois será reescrito na classe derivada. O cálculo do imposto a ser pago é a soma do valor **v** mais o **valor_imposto** (10% de **v**).

6. Salve seu arquivo com o mesmo nome da classe (**Clientes**), selecionando no meu superior **Arquivo** e depois **Salvar Como**, ou usando o atalho **ctrl+s**.



Importante

A codificação das classes deve começar com a classe-pai e depois deve passar para as classes derivadas, seguindo a sequência da herança. Caso contrário, o sistema apontará erro ao indicar uma classe-pai inexistente.



7. A próxima classe é a **Pessoa_Fisica**. Siga os mesmos passos da classe **Clientes**:

- abra um novo arquivo clicando em **Arquivo > Novo Arquivo** no menu superior;
- clique em **Selecione um idioma** e escolha a linguagem **C#**;
- salve o arquivo com o mesmo nome da classe (**Pessoa_Fisica**).

Em seguida, digite o seguinte bloco de código:

```
namespace Atividade
{
    class Pessoa_Fisica : Clientes
    {
        public string cpf {get; set;}
        public string rg {get; set;}
    }
}
```

O **namespace** é o mesmo (**Atividade**), pois as classes estão no mesmo projeto.

A classe **Pessoa_Fisica** tem dois atributos próprios e públicos: as variáveis string **cpf** e **rg**.

Para representar herança, usamos o símbolo **:** (dois pontos), indicando que a classe **Pessoa_Fisica** é derivada da classe **Clientes**, como mostra o recorte do código a seguir:

```
class Pessoa_Fisica : Clientes
```

Importante

Lembre-se de sempre de salvar os arquivos após inserir ou alterar os códigos.



7. A próxima classe é a **Pessoa_Juridica**, então:

- abra um novo arquivo clicando em **Arquivo > Novo Arquivo** no menu superior;
- clique em **Selecione um idioma** e escolha a linguagem **C#**;
- salve o arquivo com o mesmo nome da classe (**Pessoa_Juridica**).

Em seguida, digite o seguinte bloco de código:

```
namespace Atividade
{
    class Pessoa_Juridica : Clientes
    {
        public string cnpj {get; set;}
        public string ie {get; set;}
        public override void Pagar_Imposto(float v)
        {
            this.valor = v;
            this.valor_impuesto = this.valor * 20 / 100;
            this.total = this.valor + this.valor_impuesto;
        }
    }
}
```

O **namespace** é o mesmo (**Atividade**), pois as classes estão no mesmo projeto.

A classe **Pessoa_Juridica** é derivada da classe **Clientes**, então herda os atributos e métodos da classe-pai.

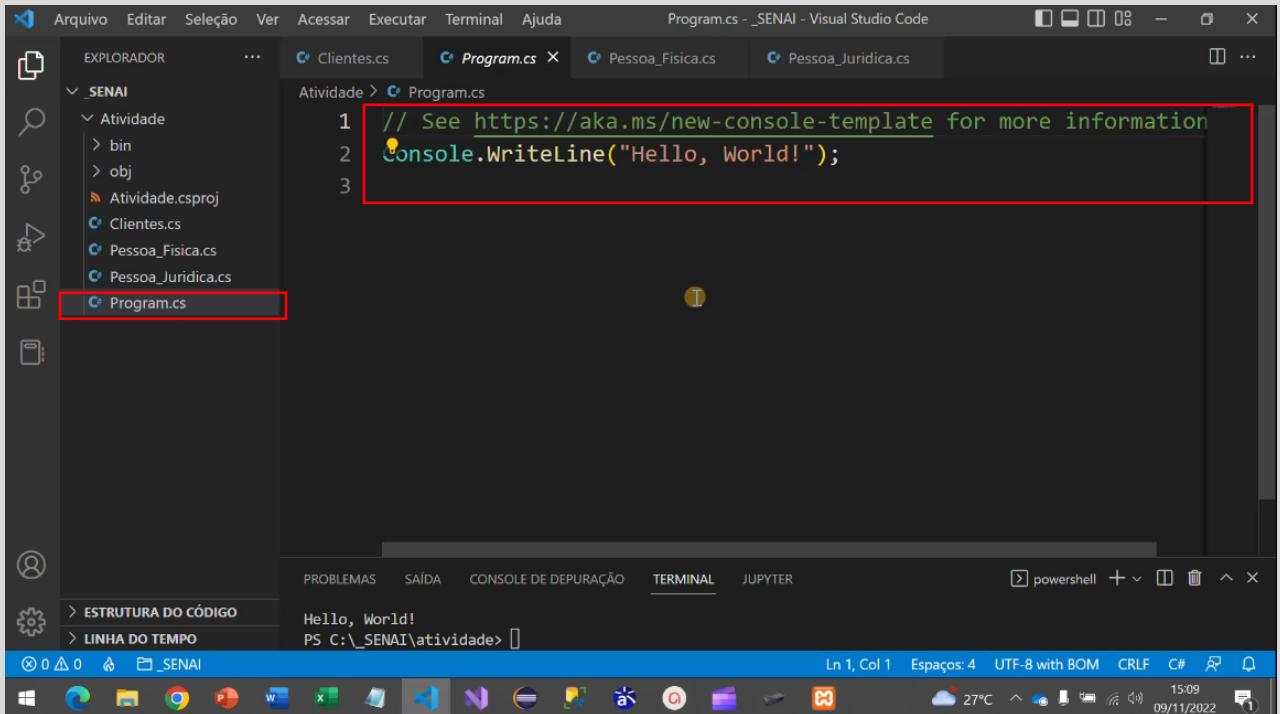
A classe **Pessoa_Juridica** tem dois atributos próprios e públicos: as variáveis string **cnnpj** e **ie**.

O método **Pagar_Imposto** é público, não tem retorno e tem o mesmo nome da classe-pai, mas com a palavra chave **override**, que indica que o método será reescrito. A diferença é que o percentual de imposto da pessoa jurídica é de 20%.

Terminada a implementação das classes, a próxima etapa é a codificação do programa em si.

Implementação do programa

1. Selecione o arquivo Program.cs no menu lateral esquerdo e apague o código teste que já está pronto.



2. Digite a seguinte estrutura básica para o Program.cs:

```
using System;
namespace Atividade
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

A primeira linha **using System;** significa que usaremos a biblioteca **System** do **C#**. O **namespace** é o mesmo (**Atividade**), pois as classes estão no mesmo projeto.

A classe desse arquivo é a **Program**. Dentro dessa classe, temos a função **Main**, dentro da qual estão todas as ações que o sistema irá executar. Essa é a estrutura básica de qualquer programa.

Dentro de **Main**, digite o bloco de código a seguir:

```
float val_pag;
    Console.WriteLine("Informar Nome");
    string var_nome = Console.ReadLine();
    Console.WriteLine("Informar Endereço");
    string var_endereco = Console.ReadLine();
    Console.WriteLine("Pessoa Física (f) ou Jurídica (j) ?");
    string var_tipo = Console.ReadLine();
    if(var_tipo == "f")
    {
        // --- Pessoa Física ----
        Pessoa_Fisica pf = new Pessoa_Fisica();
        pf.nome = var_nome;
        pf.endereco = var_endereco;
        Console.WriteLine("Informar CPF:");
        pf.cpf = Console.ReadLine();
        Console.WriteLine("Informar RG:");
        pf.rg = Console.ReadLine();
        Console.WriteLine("Informar Valor de Compra:");
        val_pag = float.Parse(Console.ReadLine());
        pf.Pagar_Imposto(val_pag);
        Console.WriteLine("----- Pessoa Física -----");
    };

    Console.WriteLine("Nome .....: " + pf.nome);
    Console.WriteLine("Endereço .....: " + pf.endereco);
    Console.WriteLine("CPF .....: " + pf.cpf);
    Console.WriteLine("RG .....: " + pf.rg);
    Console.WriteLine("Valor de Compra: " +
pf.valor.ToString("C"));
    Console.WriteLine("Imposto .....: " +
pf.valor_imposto.ToString("C"));
    Console.WriteLine("Total a Pagar : " +
pf.total.ToString("C"));
    }
    if(var_tipo == "j")
    {
```

```

// Pessoa Jurídica
    Pessoa_Juridica pj = new Pessoa_Juridica();
    pj.nome = var_nome;
    pj.endereco = var_endereco;
    Console.WriteLine("Informar CNPJ:");
    pj.cnpj = Console.ReadLine();
    Console.WriteLine("Informar IE:");
    pj.ie = Console.ReadLine();
    Console.WriteLine("Informar Valor de Compra:");
    val_pag = float.Parse(Console.ReadLine());
    pj.Pagar_Imposto(val_pag);
    Console.WriteLine("----- Pessoa Jurídica -----
");
        Console.WriteLine("Nome .....: " + pj.nome);
        Console.WriteLine("Endereço .....: " + pj.endereco);
        Console.WriteLine("CNPJ .....: " + pj.cnpj);
        Console.WriteLine("IE .....: " + pj.ie);
        Console.WriteLine("Valor de Compra: " +
pj.valor.ToString("C"));
        Console.WriteLine("Imposto .....: " +
pj.valor_imposto.ToString("C"));
        Console.WriteLine("Total a Pagar : " +
pj.total.ToString("C"));
    }
}
}
}

```

O método **Console.WriteLine** é usado para mostrar uma mensagem no console (ou terminal) do VSC. O método **Console.ReadLine** é usado para permitir entrada de dados, sempre do tipo string.

O C# é uma linguagem altamente tipada, ou seja, todas as variáveis devem ser declaradas.

Então, no primeiro bloco de código, o sistema vai apresentar a mensagem **Informar Nome** e permitir a entrada de dados, que serão armazenados na variável **var_nome**. Em seguida, aparecerá a mensagem **Informar Endereço** e a entrada de dados, que serão armazenados na variável **var_endereço**. Depois, virá o texto **Pessoa Física (f) ou Jurídica (j)**, para que o usuário indique f ou j.

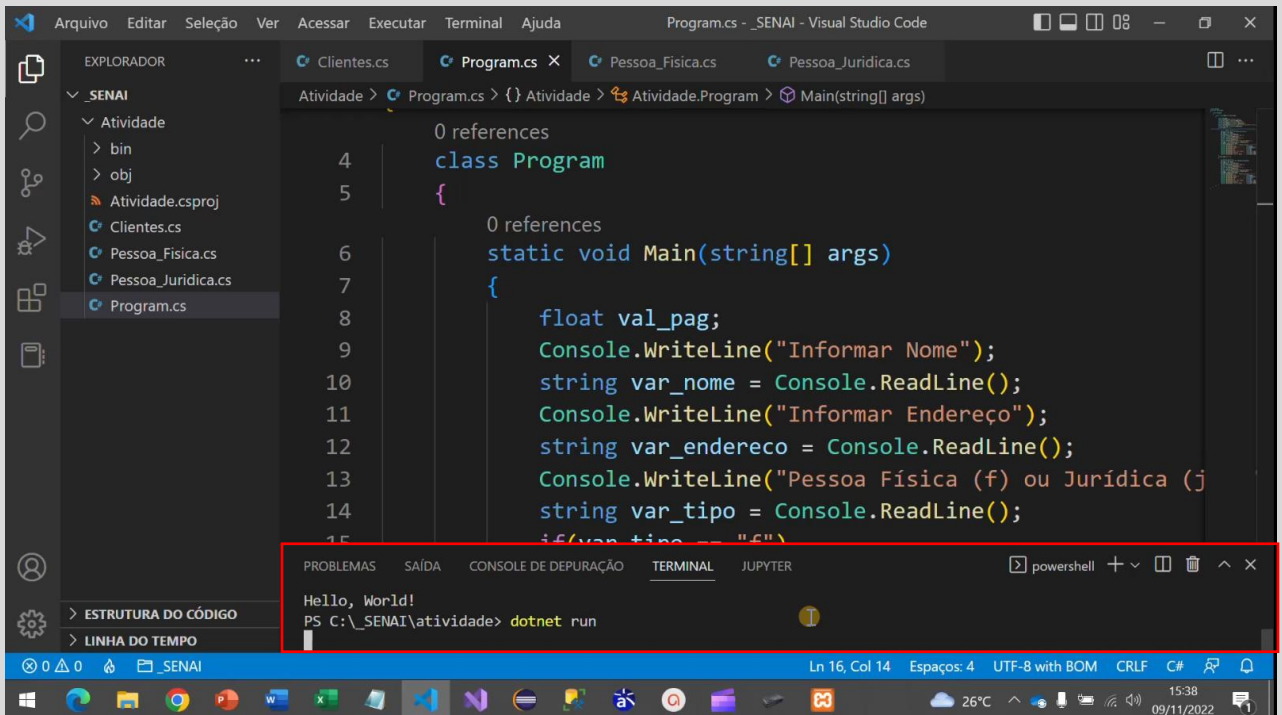
O sistema, então, fará uma verificação: se for pessoa física, o sistema pedirá cpf e rg, e o imposto cobrado será 10% em cima do valor da compra. Se for jurídica, o sistema pedirá cnpj e ie, e o imposto cobrado será de 20%.

Saiba mais

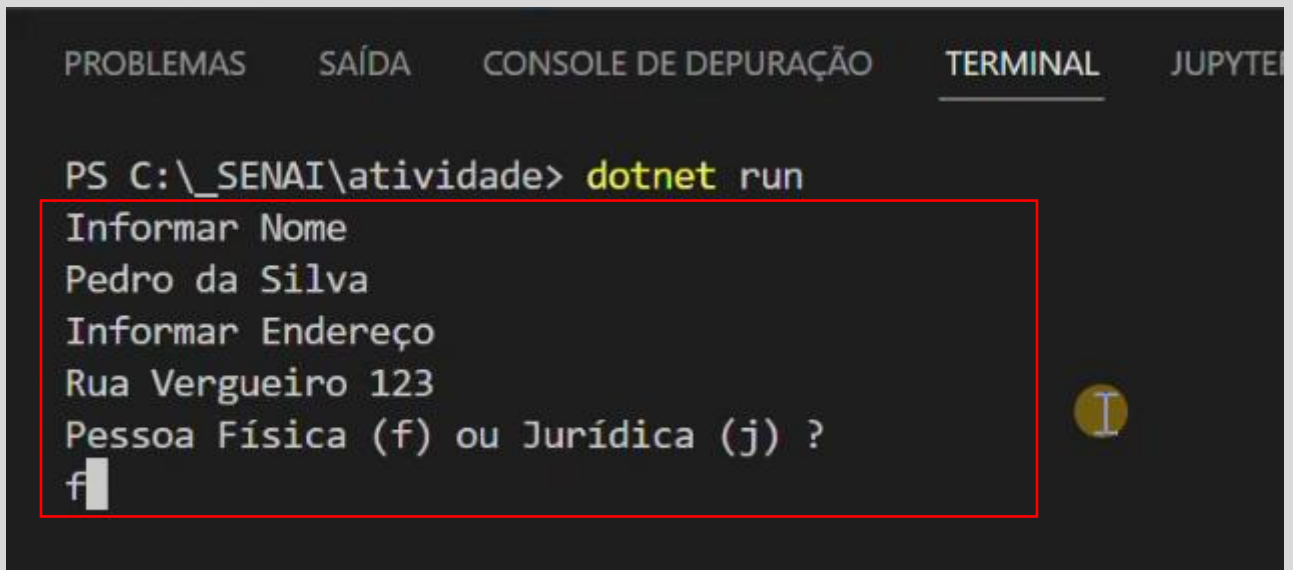
O método **float.Parse** transforma uma string em variável numérica do tipo float. Há métodos que fazem o caminho inverso, como o **toString**, que converte o valor da variável em caracteres.



3. Execute o programa, digitando **dotnet run** no terminal.



4. Digite os dados pedidos.



5. Nesse exemplo, a opção escolhida foi f de pessoa física e, portanto, o sistema vai pedir cpf e rg. Digite os dados pedidos e dê **Enter**.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  JUPYTER

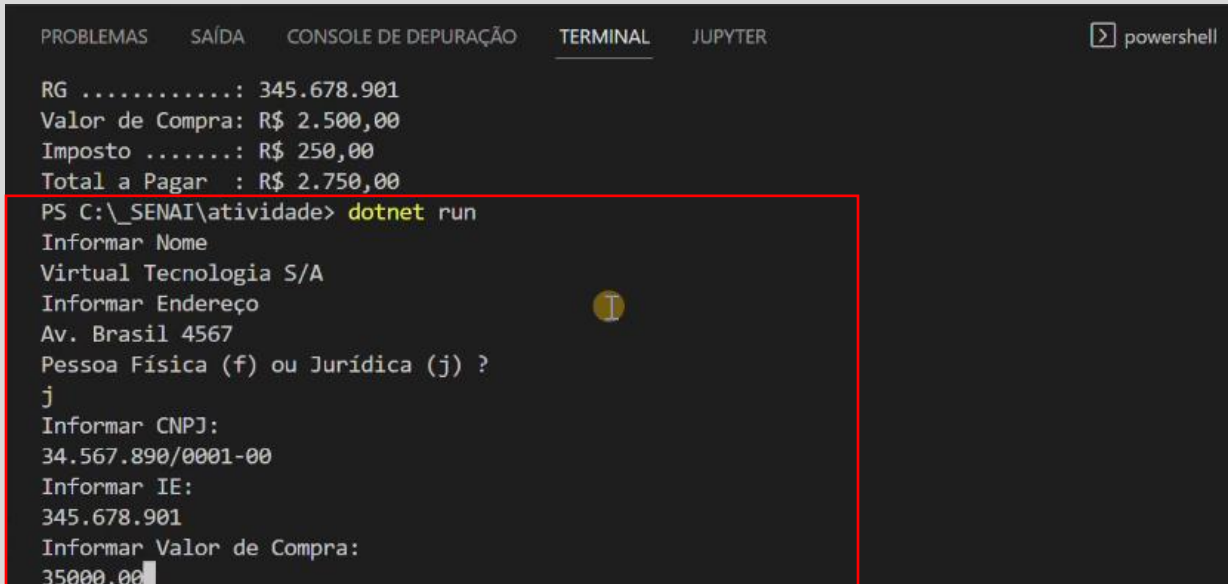
PS C:\_SENAI\atividade> dotnet run
Informar Nome
Pedro da Silva
Informar Endereço
Rua Vergueiro 123
Pessoa Física (f) ou Jurídica (j) ?
f
Informar CPF:
123.456.789-00
Informar RG:
345.678.901
Informar Valor de Compra:
2500,00
```

6. O sistema vai mostrar todos os dados e calcular corretamente o imposto (10% para pessoa física).

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  JUPYTER  powershell + v

Informar CPF:
123.456.789-00
Informar RG:
345.678.901
Informar Valor de Compra:
2500,00
----- Pessoa Física -----
Nome .....: Pedro da Silva
Endereço .....: Rua Vergueiro 123
CPF .....: 123.456.789-00
RG .....: 345.678.901
Valor de Compra: R$ 2.500,00
Imposto .....: R$ 250,00
Total a Pagar : R$ 2.750,00
PS C:\_SENAI\atividade>
```

7. Para testar a opção pessoa jurídica, digite **dotnet run** novamente e entre com os **dados pedidos**.

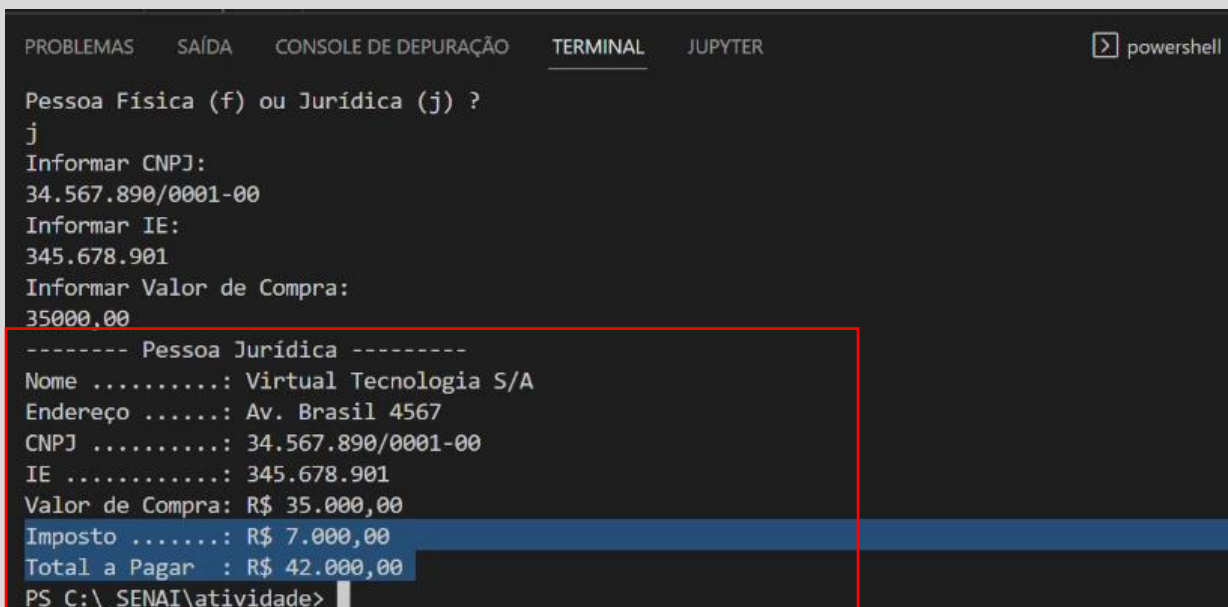


```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  JUPYTER  powershell

RG .....: 345.678.901
Valor de Compra: R$ 2.500,00
Imposto .....: R$ 250,00
Total a Pagar : R$ 2.750,00

PS C:\_SENAI\atividade> dotnet run
Informar Nome
Virtual Tecnologia S/A
Informar Endereço
Av. Brasil 4567
Pessoa Física (f) ou Jurídica (j) ?
j
Informar CNPJ:
34.567.890/0001-00
Informar IE:
345.678.901
Informar Valor de Compra:
35000,00
```

8. O sistema vai mostrar todos os dados e calcular corretamente o imposto (20% para pessoa jurídica).



```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURAÇÃO  TERMINAL  JUPYTER  powershell

Pessoa Física (f) ou Jurídica (j) ?
j
Informar CNPJ:
34.567.890/0001-00
Informar IE:
345.678.901
Informar Valor de Compra:
35000,00

----- Pessoa Jurídica -----
Nome .....: Virtual Tecnologia S/A
Endereço .....: Av. Brasil 4567
CNPJ .....: 34.567.890/0001-00
IE .....: 345.678.901
Valor de Compra: R$ 35.000,00
Imposto .....: R$ 7.000,00
Total a Pagar : R$ 42.000,00
PS C:\_SENAI\atividade>
```

9. Para testar a opção pessoa jurídica, digite **dotnet run** novamente e entre com os **dados pedidos**.

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  JUPYTER  powershell

RG .....: 345.678.901
Valor de Compra: R$ 2.500,00
Imposto .....: R$ 250,00
Total a Pagar : R$ 2.750,00

PS C:\_SENAI\atividade> dotnet run
Informar Nome
Virtual Tecnologia S/A
Informar Endereço
Av. Brasil 4567
Pessoa Física (f) ou Jurídica (j) ?
j
Informar CNPJ:
34.567.890/0001-00
Informar IE:
345.678.901
Informar Valor de Compra:
35000,00
```

10. O sistema vai mostrar todos os dados e calcular corretamente o imposto (20% para pessoa jurídica).

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  JUPYTER  powershell

Pessoa Física (f) ou Jurídica (j) ?
j
Informar CNPJ:
34.567.890/0001-00
Informar IE:
345.678.901
Informar Valor de Compra:
35000,00

----- Pessoa Jurídica -----
Nome .....: Virtual Tecnologia S/A
Endereço .....: Av. Brasil 4567
CNPJ .....: 34.567.890/0001-00
IE .....: 345.678.901
Valor de Compra: R$ 35.000,00
Imposto .....: R$ 7.000,00
Total a Pagar : R$ 42.000,00
PS C:\_SENAI\atividade>
```

Nesta atividade, usamos todos os conceitos básicos de POO:

- abstração: ao transformar o mundo real em classes;
- herança: ao usar classe pai e classes derivadas;
- Polimorfismo: ao usar o **override** no método **Pagar_Imposto**;
- Encapsulamento: ao usar o modificador de acesso **protected** nos métodos **set** da classe-pai.

Revisamos, também, os conceitos de variáveis, estruturas de decisão e sintaxe de C#.