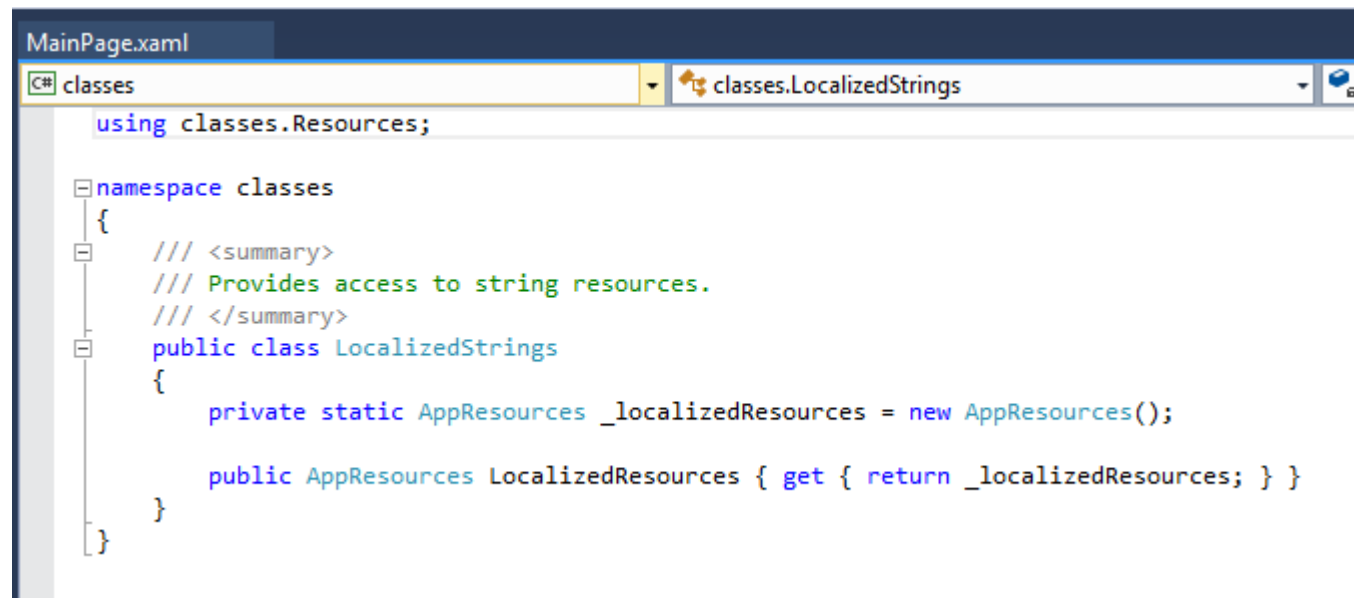


# Aplicações com Windows Phone

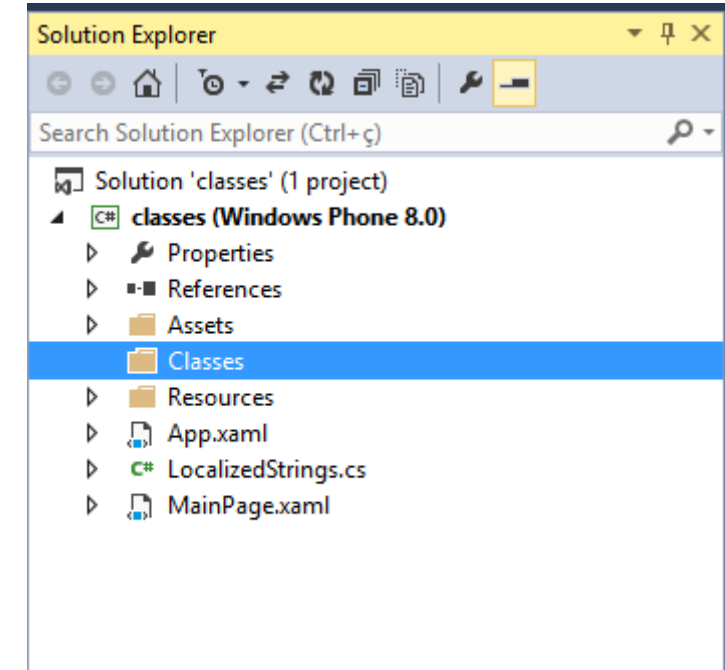
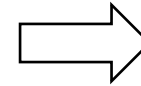
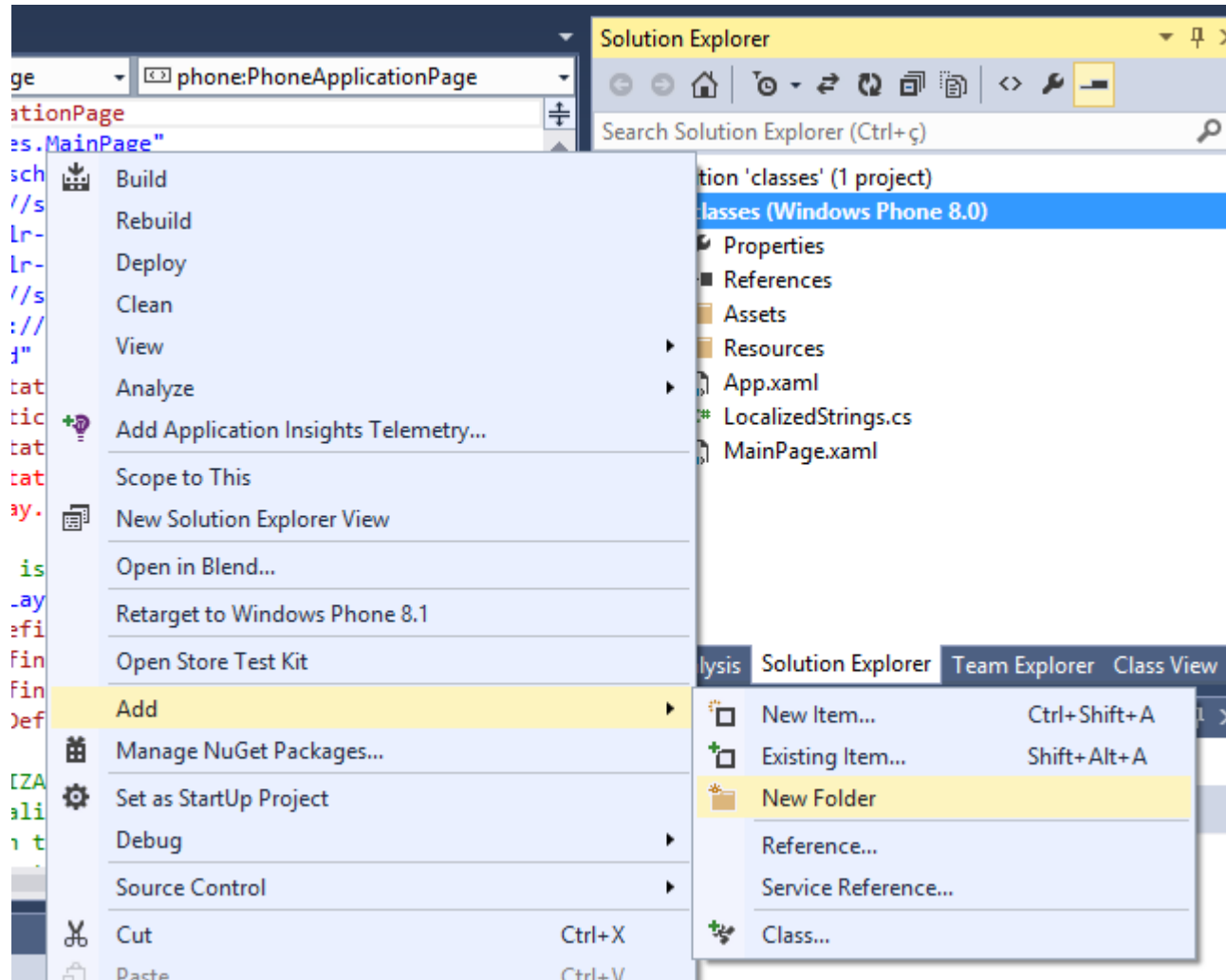
Orientação a Objetos

## Criação de novo projeto: classes

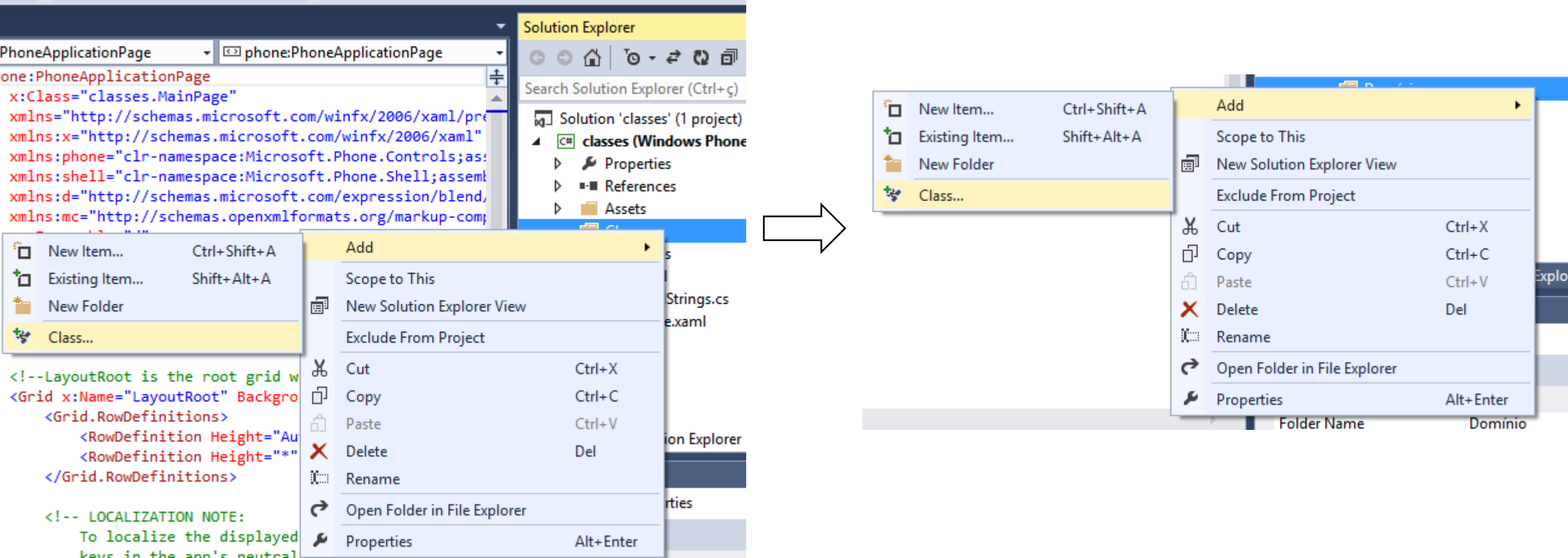


MainPage.xaml do projeto

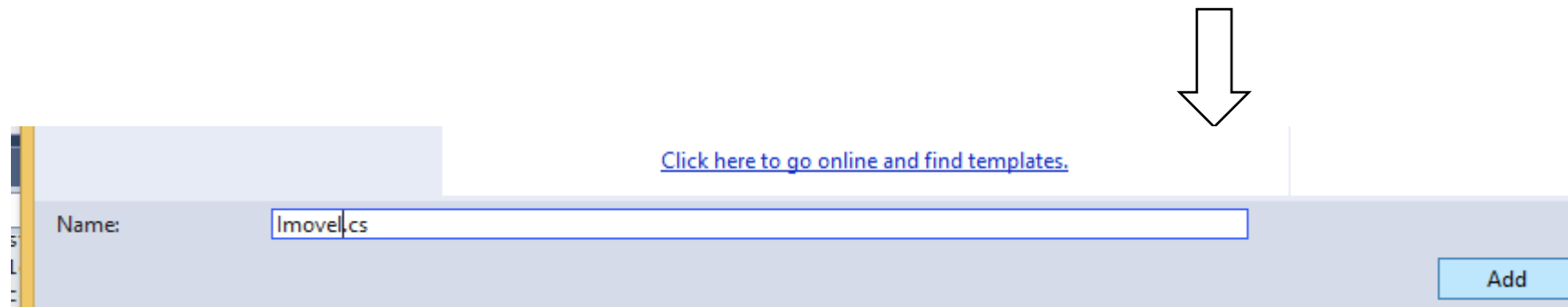
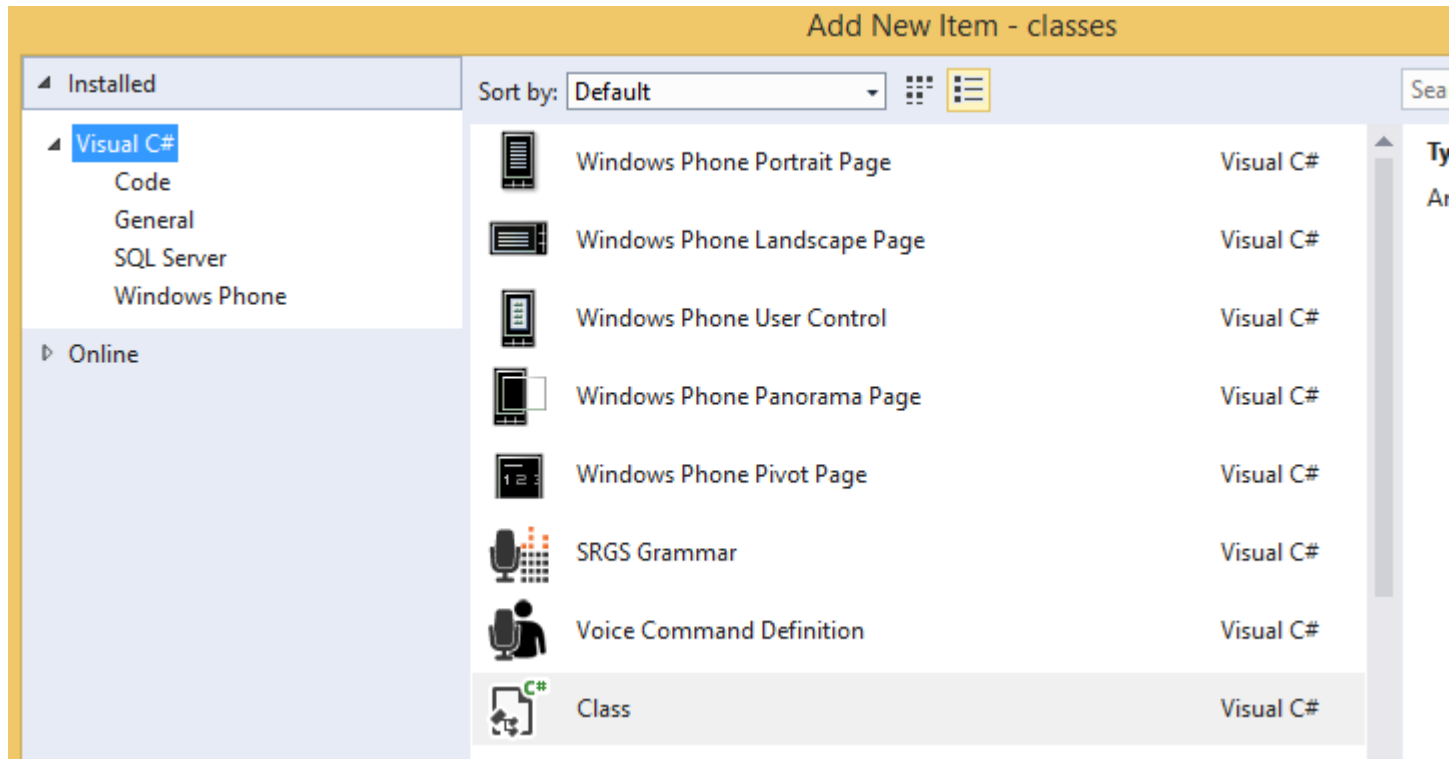
## Adicionando uma pasta ao projeto:



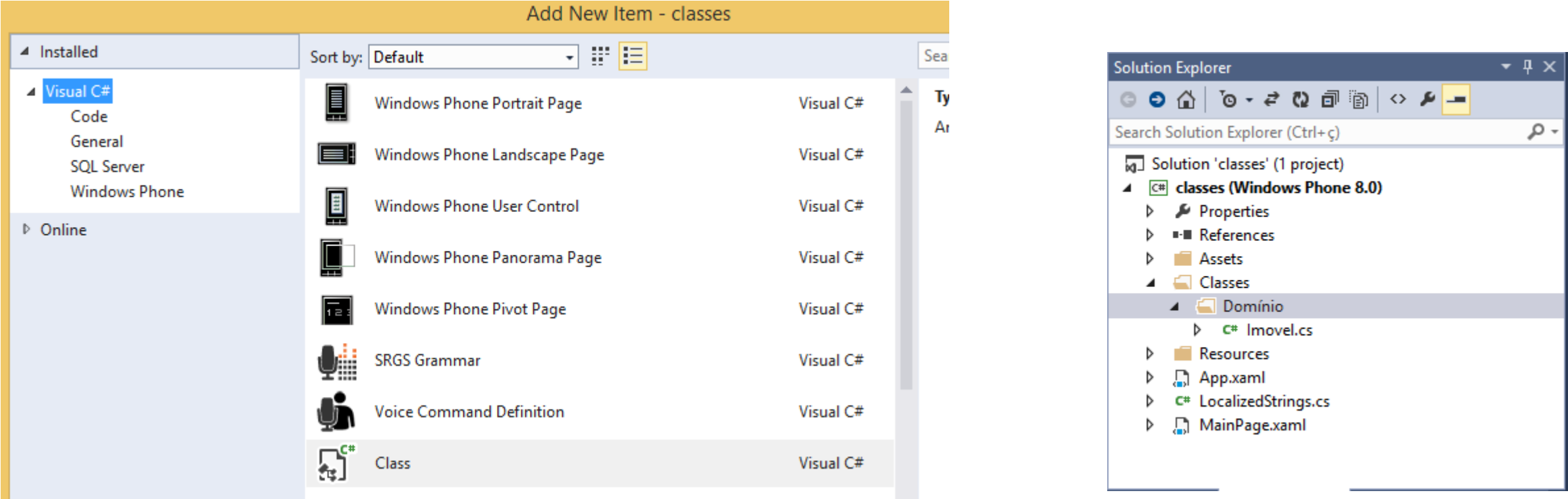
Adicionando uma subpasta ao diretório criado Classes (\Domínio) e criando uma classe Visual C# na pasta:



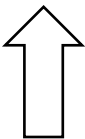
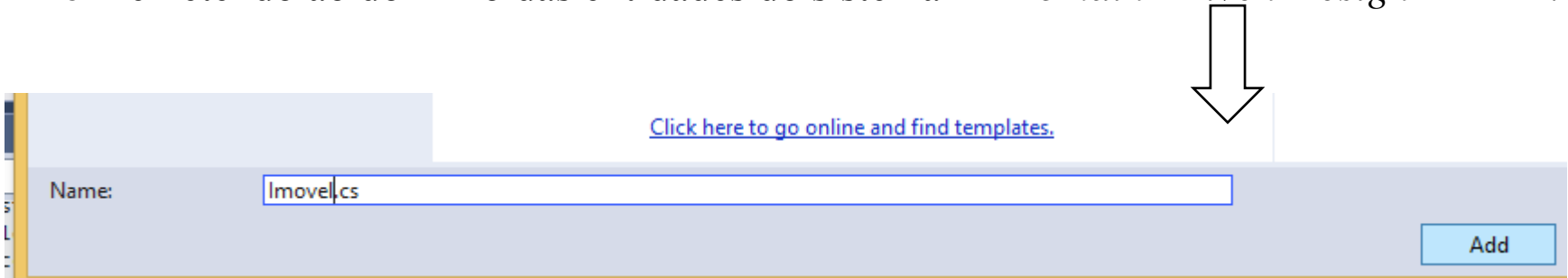
Adicionando uma subpasta ao diretório criado Classes (\Domínio) e criando uma classe Visual C# na pasta: Classe Imovel.cs



Adicionando uma subpasta ao diretório criado Classes (\Domínio) e criando uma classe Visual C# na pasta: Classe Imovel.cs.

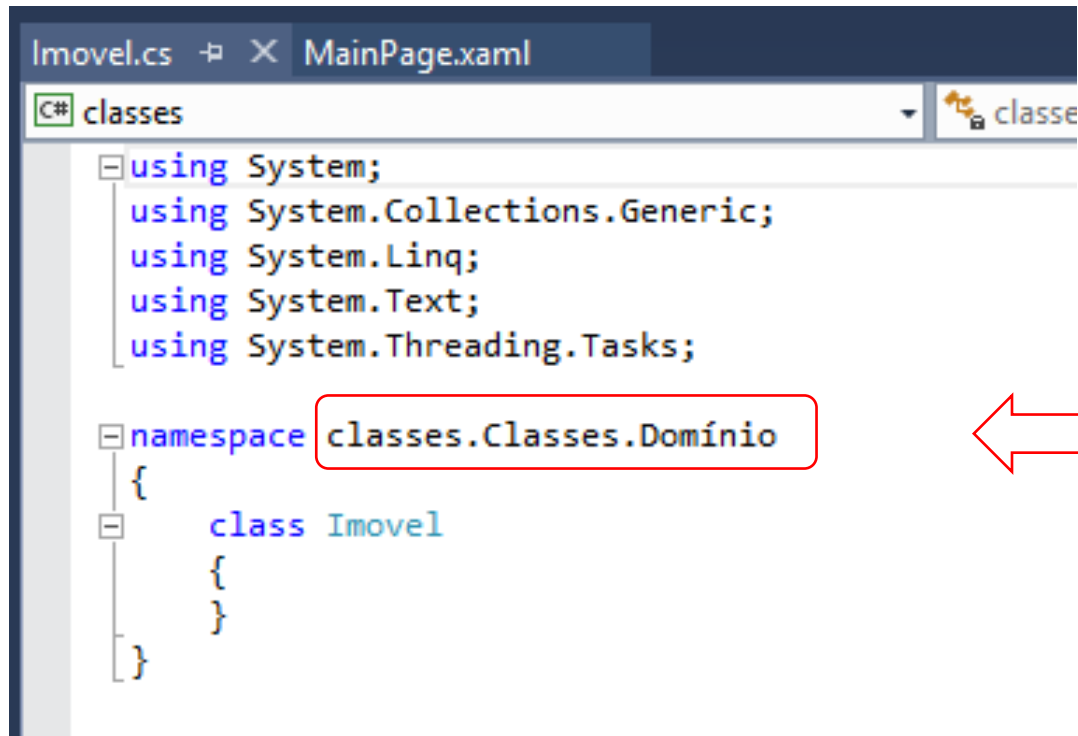


Pasta “Domínio” remetendo ao domínio das entidades do sistema → *Domain Driven Design* – DDD.



(Lecheta, 2014)

## Visualização da classe Imovel.cs.

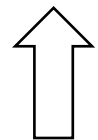


```
Imovel.cs X MainPage.xaml
C# classes
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

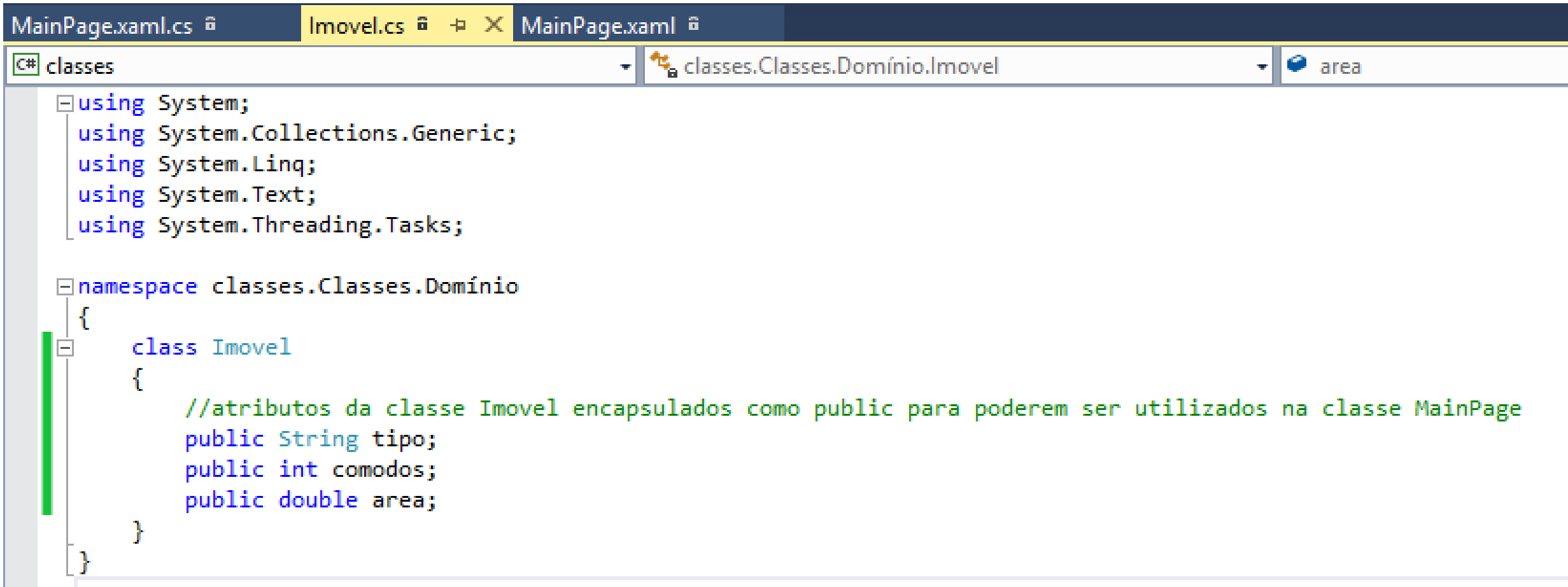
namespace classes.Classes.Domínio
{
    class Imovel
    {
    }
}
```

← namespace que reflete a estrutura física das pastas onde está situada a classe Imovel.

Classes da linguagem C# herdam diretamente de System.Object.



## Classe Imovel.cs.



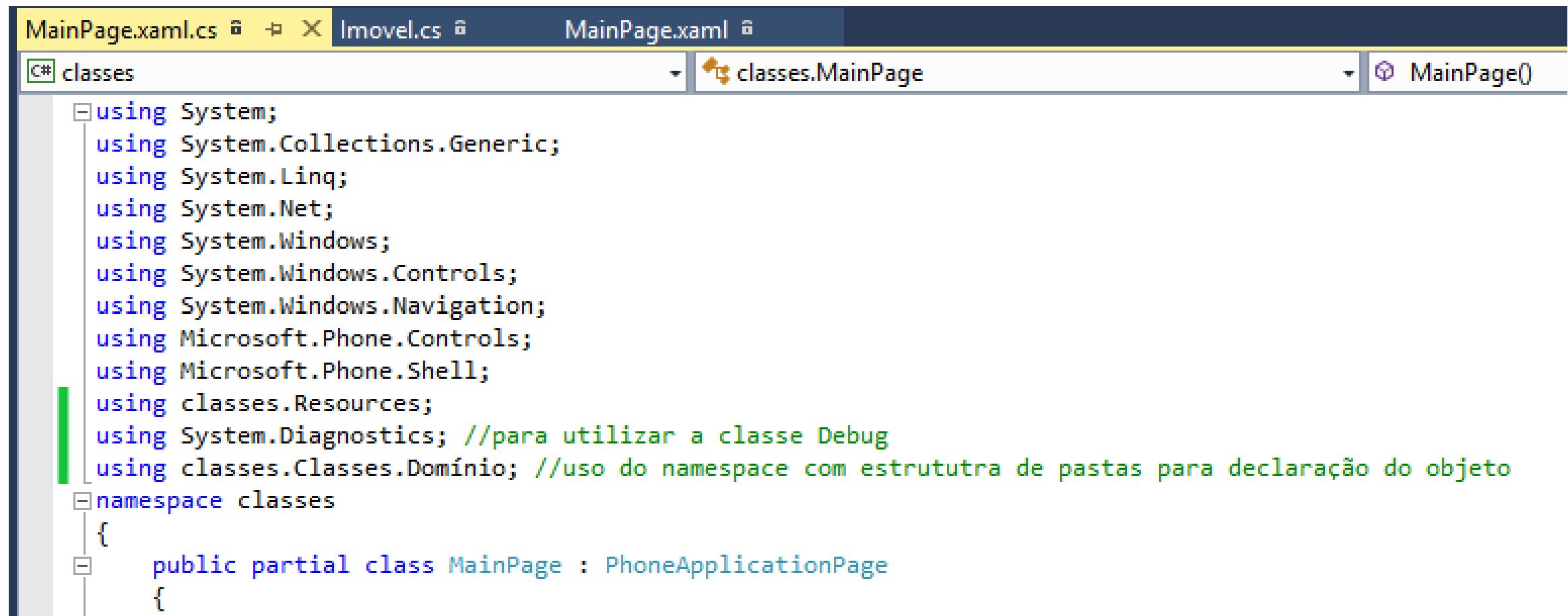
The image shows a Visual Studio code editor window with the following tabs: MainPage.xaml.cs, Imovel.cs (selected), and MainPage.xaml. The file explorer on the left shows the project structure: C# classes, classes.Classes.Domínio, and area. The code in Imovel.cs is as follows:

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

namespace classes.Classes.Domínio
{
    class Imovel
    {
        //atributos da classe Imovel encapsulados como public para poderem ser utilizados na classe MainPage
        public String tipo;
        public int comodors;
        public double area;
    }
}
```



# MainPage.xaml.cs.



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Navigation;
using Microsoft.Phone.Controls;
using Microsoft.Phone.Shell;
using classes.Resources;
using System.Diagnostics; //para utilizar a classe Debug
using classes.Classes.Domínio; //uso do namespace com estrutura de pastas para declaração do objeto

namespace classes
{
    public partial class MainPage : PhoneApplicationPage
    {
    }
```

## MainPage.xaml.cs.

Declaração de objeto Imovel, inserção de valores aos atributos e impressão de dados.

```
// Constructor
public MainPage()
{
    InitializeComponent();

    //imprime mensagem no console
    Debug.WriteLine("Exemplo de classes e objetos.");

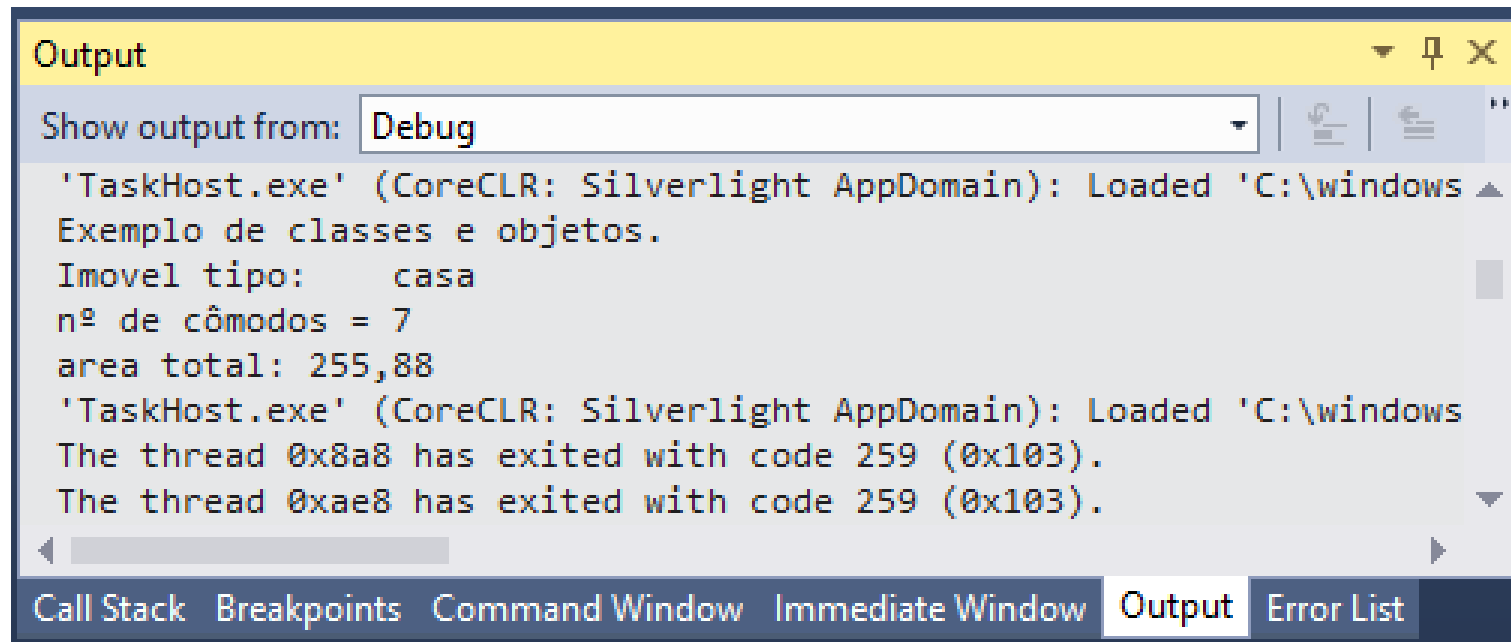
    //declaração de objeto da classe Imovel:
    Imovel v = new Imovel();

    //atribuição de valores aos atributos
    v.tipo = "casa";
    v.comodos = 7;
    v.area = 255.88;

    //imprime informações do imóvel no console
    Debug.WriteLine("Imovel tipo:\t" + v.tipo + "\n" + "nº de cômodos =\t" + v.comodos + "\n" + "area total:\t" + v.area);
}
}
```

## MainPage.xaml.cs e classe Imovel.

Saída do projeto.



The screenshot shows the 'Output' window in Visual Studio. The 'Show output from:' dropdown is set to 'Debug'. The output text is as follows:

```
'TaskHost.exe' (CoreCLR: Silverlight AppDomain): Loaded 'C:\windows  
Exemplo de classes e objetos.  
Imovel tipo:      casa  
nº de cômodos = 7  
area total: 255,88  
'TaskHost.exe' (CoreCLR: Silverlight AppDomain): Loaded 'C:\windows  
The thread 0x8a8 has exited with code 259 (0x103).  
The thread 0xae8 has exited with code 259 (0x103).
```

At the bottom of the window, there are tabs for 'Call Stack', 'Breakpoints', 'Command Window', 'Immediate Window', 'Output' (which is selected), and 'Error List'.

Classe Imovel\_1.cs (pasta Domínio) com atributos encapsulados como private + métodos set( ) e get( ).

```
Imovel_1.cs MainPage.xaml.cs Imovel.cs MainPage.xaml
C# classes classes.Classes.Domínio.Imovel_1 settipo(String tipo)
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace classes.Classes.Domínio
{
    class Imovel_1
    {
        //atributos da classe Imovel_1 encapsulados como public para poderem ser utilizados na classe MainPage
        private String tipo;
        private int comodos;
        private double area;

        //métodos set
        public void settipo(String tipo)
        {
            this.tipo = tipo;
        }
    }
}
```

```
public void setcomodos(int comodos)
{
    this.comodos = comodos;
}

public void setarea(double area)
{
    this.area = area;
}

//métodos get
public String gettipo( )
{
    return tipo;
}

public int getcomodos()
{
    return comodos;
}

public double getarea()
{
    return area;
}
}
```

Trecho de MainPage.xaml.cs com atribuição de valores através dos métodos set( ) e consulta de dados com os métodos get( ).

```
//imprime informações do imóvel no console
Debug.WriteLine("Imovel tipo:\t" + v.tipo + "\n" + "nº de cômodos =\t" + v.comodos + "\n" + "area total:\t" + v.area);

//declaração de objeto da classe Imovel_1:
Imovel_1 y = new Imovel_1();

//atribuição de valores aos atributos
y.settipo("apartamento");
y.setcomodos(5);
y.setarea(72.2);

//imprime informações do imóvel no console
Debug.WriteLine("Imovel tipo:\t" + y.gettipo( ) + "\n" + "nº de cômodos =\t" + y.getcomodos( ) + "\n" + "area total:\t" + y.getarea( ));

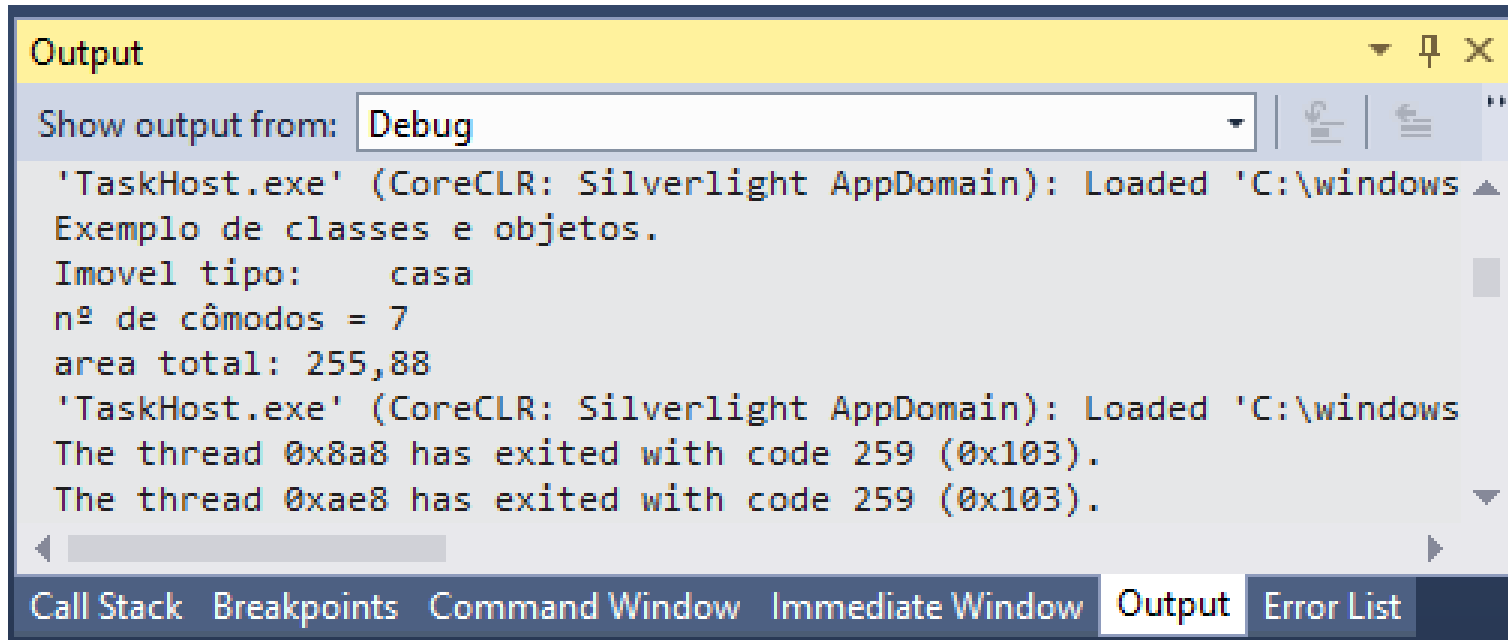
}

}

}
```

MainPage.xaml.cs e classes Imovel + Imovel\_1.

Saída do projeto com as classes Imovel e Imovel\_1.



The screenshot shows the 'Output' window in Visual Studio. The 'Show output from:' dropdown is set to 'Debug'. The output text is as follows:

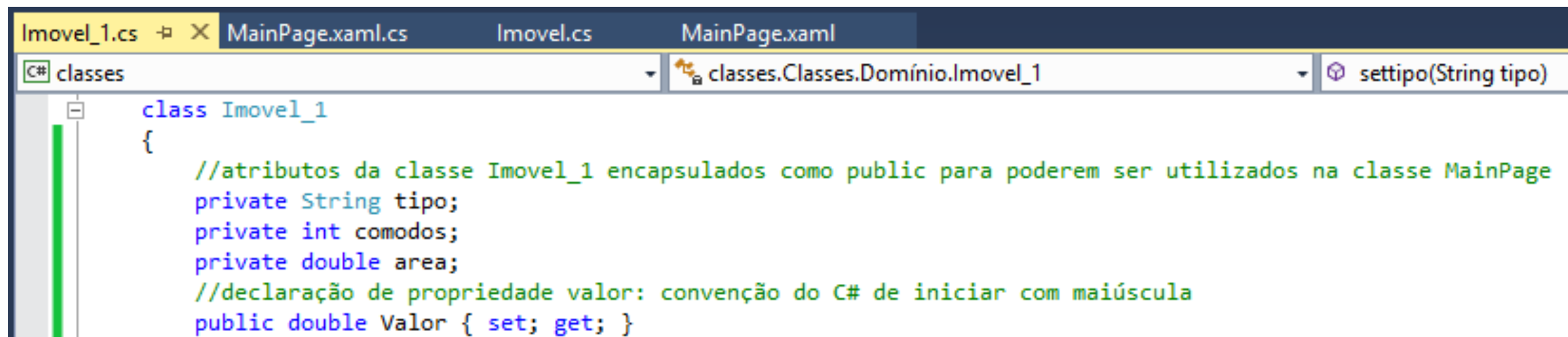
```
'TaskHost.exe' (CoreCLR: Silverlight AppDomain): Loaded 'C:\windows  
Exemplo de classes e objetos.  
Imovel tipo:      casa  
nº de cômodos = 7  
area total: 255,88  
'TaskHost.exe' (CoreCLR: Silverlight AppDomain): Loaded 'C:\windows  
The thread 0x8a8 has exited with code 259 (0x103).  
The thread 0xae8 has exited with code 259 (0x103).
```

At the bottom of the window, there are tabs for 'Call Stack', 'Breakpoints', 'Command Window', 'Immediate Window', 'Output' (which is selected), and 'Error List'.

## Transformação de atributo em propriedade para modificar o modo de acesso:

```
//declaração de propriedade valor: convenção do C# de iniciar com maiúscula  
public double Valor { set; get; }
```

A propriedade Valor pode ser utilizada como uma variável pública. Não se utilizam os métodos get( ) e set( ).



The screenshot shows a Visual Studio code editor with the following elements:

- File Explorer:** Shows the project structure with files `Imovel_1.cs`, `MainPage.xaml.cs`, `Imovel.cs`, and `MainPage.xaml`.
- Search Bar:** Displays `C# classes` and the current file path `classes.Classes.Domínio.Imovel_1`.
- Code Editor:** Contains the following C# code:

```
class Imovel_1  
{  
    //atributos da classe Imovel_1 encapsulados como public para poderem ser utilizados na classe MainPage  
    private String tipo;  
    private int comodoss;  
    private double area;  
    //declaração de propriedade valor: convenção do C# de iniciar com maiúscula  
    public double Valor { set; get; }  
}
```

## Transformação de atributo em propriedade para modificar o modo de acesso – classe MainPage.xaml.cs:

```
//imprime informações do imóvel no console
Debug.WriteLine("Imovel tipo:\t" + y.gettipo( ) + "\n" + "nº de cômodos =\t" + y.getcomodos( ) + "\n"

//acesso à propriedade Valor:
y.Valor = 376987.00;
Debug.WriteLine("Valor do imovel: R$\t" + y.Valor);
```

### Output

Show output from:

```
area total: 255,88
Imovel tipo:      apartamento
nº de cômodos = 5
area total: 72,2
Valor do imovel: R$ 376987
```



## Propriedade: controle do modo de acesso – classe Imovel\_1.cs:

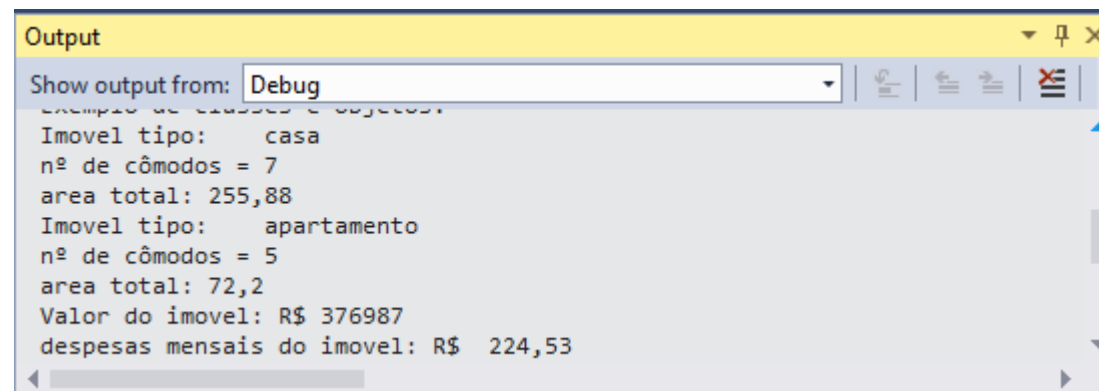
```
//propriedade Despesas
public double Despesas { get; private set; }

//método calcula_Despesas
public double calcula_Despesas(double conta_agua, double conta_luz)
{
    Despesas += conta_agua + conta_luz;
    return Despesas;
}
```

## Acesso a propriedade encapsulada como private – classe MainPage.xaml.cs:

```
//chamada ao método para calcular despesas: acesso à propriedade private Despesas
y.calcula_Despesas(100.65, 123.88);
Debug.WriteLine("despesas mensais do imovel: R$\t" + y.Despesas);
```

## Resultado: Output



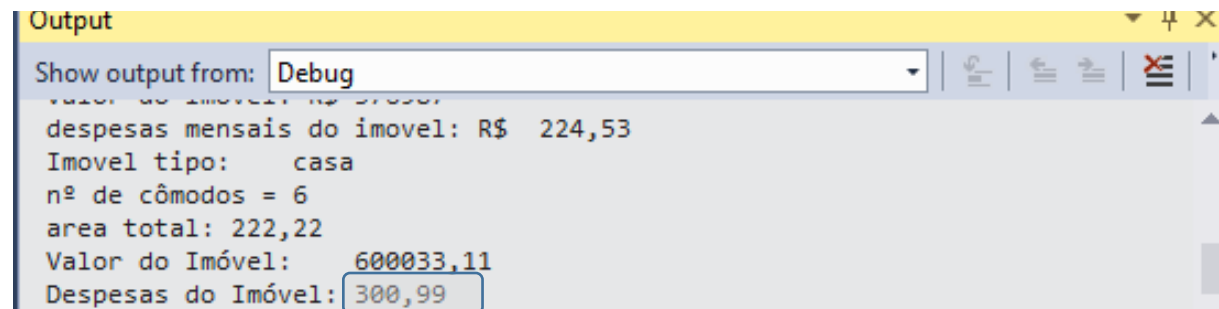
## Criação de método construtor na classe Imovel\_1.cs:

```
//método construtor
public Imovel_1 (String tipo, int comodos, double area, double Valor, double Despesas)
{
    this.tipo = tipo;
    this.comodos = comodos;
    this.area = area;
    this.Valor = Valor;
    this.Despesas = Despesas;
}
```

## Chamada ao método construtor – classe MainPage.xaml.cs:

```
//chamada ao método construtor com valores iniciais
Imovel_1 construtor = new Imovel_1("casa", 6, 222.22, 600033.11, 300.99);
Debug.WriteLine("Imovel tipo:\t" + construtor.gettipo() + "\n" + "nº de cômodos =\t" + construtor.getcomodos() + "\n" + "area total:\t"
+ construtor.getarea() + "\n" + "Valor do Imóvel:\t" + construtor.Valor + "\n" + "Despesas do Imóvel:\t" + construtor.Despesas);
```

## Resultado: Output



## Criação de classe herdeira da classe Imovel\_1.cs: classe Contrato\_Imovel.cs

```
Contrato_Imovel.cs X Imovel_1.cs MainPage.xaml.cs* Imovel.cs MainPage.xaml
[C#] classes classes.Classes.Domínio.Contrato_Imovel
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace classes.Classes.Domínio
{
    /*declaração da classe Contrato_Imovel com relação de herança
    com a classe Imovel_1 = superclasse ou classe mãe*/
    class Contrato_Imovel:Imovel_1
    {
        private String tipo_contrato;
        public Contrato_Imovel() : base() { }

        /*modificador base é utilizado para acessar métodos e atributos da superclasse;
        é similar ao método "super( )" */

        //método set
        public void settipo_contrato(String tipo_contrato) { this.tipo_contrato = tipo_contrato; }

        //método get
        public String gettipo_contrato() { return tipo_contrato; }
    }
}
```

## Chamada à classe herdeira – classe MainPage.xaml.cs:

```
//declaração de objeto da classe herdeira Contrato_Imovel
Contrato_Imovel c = new Contrato_Imovel();

//chamada ao método set
c.settipo_contrato("locação");

//chamada ao método de impressão
imprime(c);

} //fim do MainPage( )
//método imprime( ) recebe um objeto "d" do tipo da classe Contrato_Imovel
void imprime(Contrato_Imovel d) { Debug.WriteLine("Contrado do Imovel tipo:\t" + d.gettipo_contrato()); }
}
```

## Resultado: Output

Show output from: Debug

```
valor do imovel: R$ 222,22
despesas mensais do imovel: R$ 224,53
Imovel tipo: casa
nº de cômodos = 6
area total: 222,22
Valor do Imóvel: 600033,11
Despesas do Imóvel: 300,99
Contrado do Imovel tipo: locação
```

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

namespace classes.Classes.Domínio
{
    //declaração de classe abstrata
    abstract class Imovel_2
    {
        //atributos
        public String tipo_imovel { get; set; }
        public double valor_imovel { get; set; }

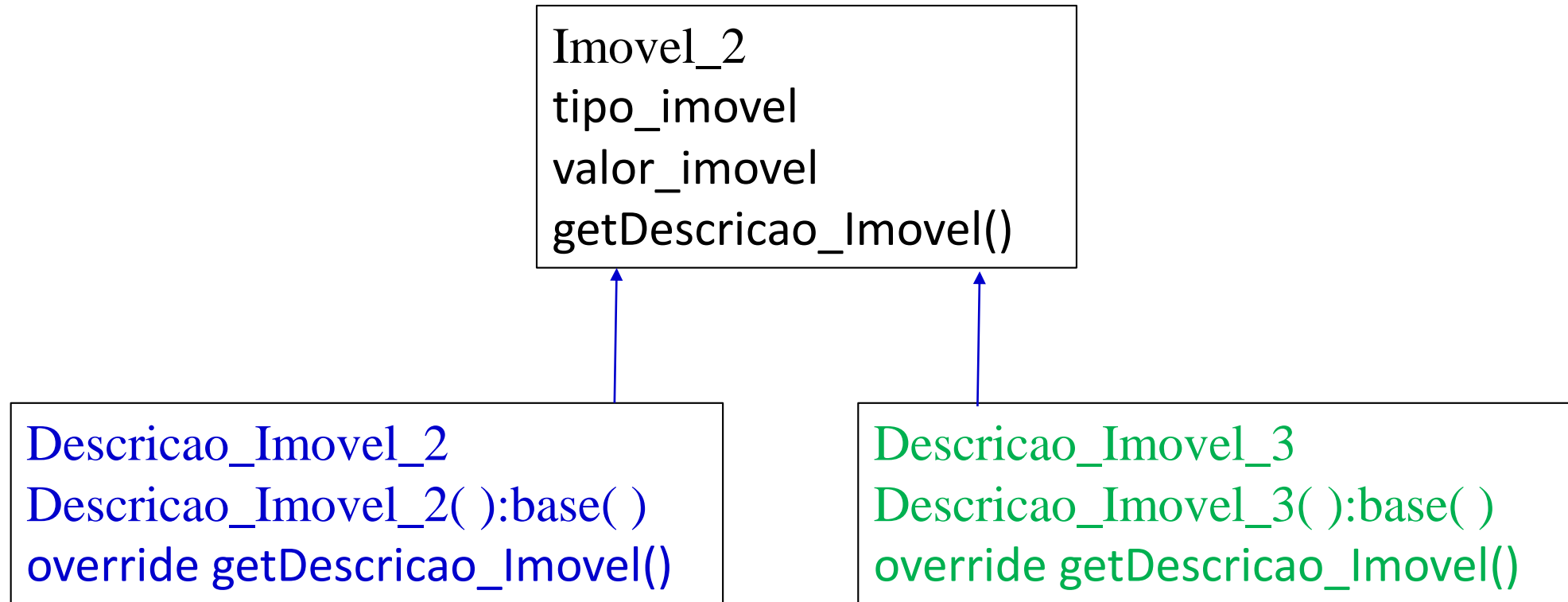
        //método construtor
        public Imovel_2(String tipo_imovel, double valor_imovel)
        { this.tipo_imovel = tipo_imovel;
          this.valor_imovel = valor_imovel;
        }

        /* criação de método abstrato para descrever imóvel;
        métodos abstratos não possuem implementação */
        public abstract String getDescricao_Imovel();
    }
}
```

Herança, Polimorfismo e classe Abstrata

Classe Imovel\_2 → abstrata: não poderá ser criado objeto diretamente de classe abstrata.

Relação de Herança entre a superclasse ou classe mãe abstrata Imovel\_2 e as classes Descreve\_Imovel\_2 e Descreve\_Imovel\_3:



## Classe Descreve\_Imovel\_2

```
Descreve_Imovel_3.cs  Descreve_Imovel_2.cs  Imovel_2.cs  Contrato_Imovel.cs  Imovel_1.cs  MainPage.xaml.cs  Im
C# classes  classes.Classes.Domínio.Descreve_Imovel_2  getDescricao_Imovel()

using System.Text;
using System.Threading.Tasks;

namespace classes.Classes.Domínio
{
    //Descreve_Imovel_2 é subclasse (herdeira) da classe abstrata Imovel_2
    class Descreve_Imovel_2 : Imovel_2
    {
        //declaração do construtor da classe mãe
        public Descreve_Imovel_2(String tipo_imovel, double valor_imovel) : base(tipo_imovel, valor_imovel) { }

        //uso do modificador override para implementar o método abstrato da classe mãe
        public override String getDescricao_Imovel()
        { return "IMÓVEL CASA : " + this.tipo_imovel + ", VALOR DA CASA = R$ " + this.valor_imovel; }
    }
}
```

## Classe Descreve\_Imovel\_3

```
Descreve_Imovel_3.cs* X Descreve_Imovel_2.cs Imovel_2.cs Contrato_Imovel.cs Imovel_1.cs MainPage.xaml.cs Imovel.cs
C# classes classes.Classes.Domínio.Descreve_Imovel_3 getDescricao_Imovel()

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

namespace classes.Classes.Domínio
{
    //Descreve_Imovel_3 é subclasse (herdeira) da classe abstrata Imovel_2
    class Descreve_Imovel_3 : Imovel_2
    {
        //declaração do construtor da classe mãe
        public Descreve_Imovel_3(String tipo_imovel, double valor_imovel) : base(tipo_imovel, valor_imovel) { }

        //uso do modificador override para implementar o método abstrato da classe mãe
        public override String getDescricao_Imovel()
        { return "IMÓVEL APARTAMENTO : " + this.tipo_imovel + ", VALOR DO APARTAMENTO = R$ " + this.valor_imovel; }
    }
}
```



MainPage.xaml.cs: instância dos objetos à classe abstrata; método **mostra( )** implementa Polimorfismo.

```
//objeto é criado a partir da classe herdeira da abstrata Descrição_Imovel_2
Descreve_Imovel_2 f = new Descreve_Imovel_2("casa", 444000.05);

//objeto é criado a partir da classe herdeira da abstrata Descrição_Imovel_3
Descreve_Imovel_3 h = new Descreve_Imovel_3("apartamento", 222000.22);

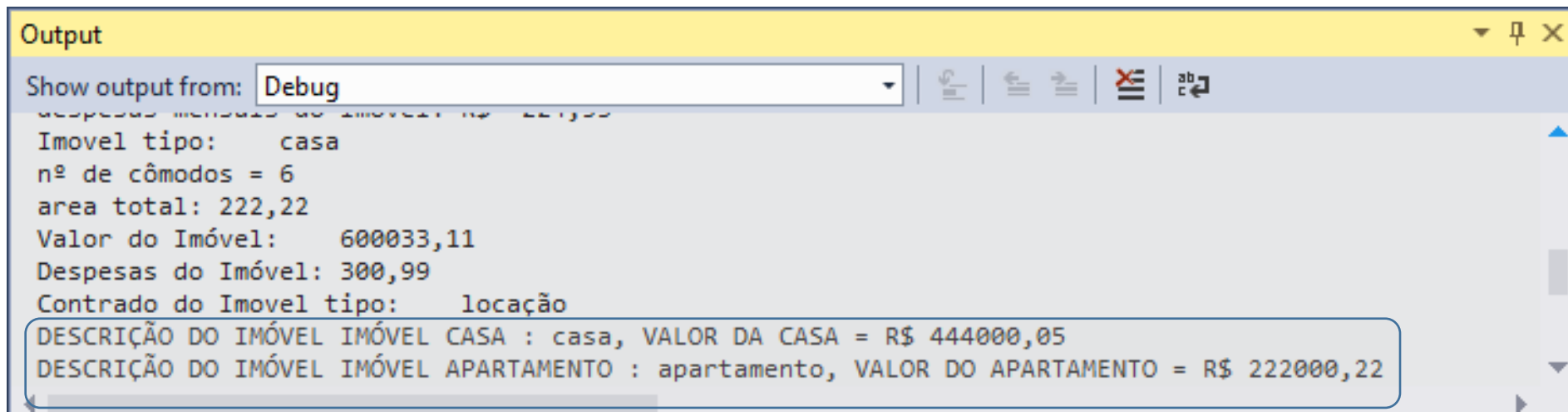
//chamada ao método mostra() na superclasse Imovel_2
mostra(f);
mostra(h);

} //fim do MainPage( )

void imprime(Contrato_Imovel d) { Debug.WriteLine("Contrado do Imovel tipo:\t" + d.gettipo_contrato()); }

/*método mostra() implementa o conceito de Polimorfismo através do método getDescricao_Imovel(),
que será utilizado pelos objetos f e h */
void mostra(Imovel_2 g) { Debug.WriteLine("DESCRIÇÃO DO IMÓVEL " + g.getDescricao_Imovel()); }
```

Resultado: Output



The screenshot shows the 'Output' window in Visual Studio with the 'Debug' filter selected. It displays the output of the application, including property values for a house and an apartment, and the results of the 'mostra()' method calls. A blue box highlights the last two lines of output.

```
Output
Show output from: Debug
Despesas mensais do Imovel: R$ 222,22
Imovel tipo:      casa
nº de cômodos = 6
area total: 222,22
Valor do Imóvel:   600033,11
Despesas do Imóvel: 300,99
Contrado do Imovel tipo:   locação
DESCRIÇÃO DO IMÓVEL IMÓVEL CASA : casa, VALOR DA CASA = R$ 444000,05
DESCRIÇÃO DO IMÓVEL IMÓVEL APARTAMENTO : apartamento, VALOR DO APARTAMENTO = R$ 222000,22
```

Uso de Interface: o uso de interfaces permite criar herança múltipla, ou seja, herdar de mais de uma classe. Interface é uma classe abstrata que não possui implementação e não podem ser instanciadas. As classes que implementam uma interface devem implementar todos os seus métodos.

Criação de Interface:

