Linguagem de Programação II

Analise e Desenvolvimento de Sistemas

2º semestre de 2019

Prof. Me. Renato Carioca Duarte

Classes e Objetos

- Uma classe é considerada um tipo de dado, como os tipos básicos pré-definidos (inteiro, real, string, etc.) em todas as linguagens de programação.
- Sendo assim da mesma forma que podemos criar uma variável do tipo “int” também podemos criar uma variável do tipo da classe. Esta variável da classe é chamada de OBJETO.
- Ao definirmos uma classe, estamos fornecendo a estrutura para o objeto.
- Toda vez que declaramos uma variável originada de uma classe dizemos que estamos instanciando um objeto.
- Quando definimos uma classe estamos criando um modelo para criarmos objetos e/ou outras classes(herança).

Classes e Objetos

- Um objeto deverá possuir uma identidade, estado e comportamento.

1. Identidade: Todo objeto possui uma identidade. Por exemplo, um novo objeto:

Aluno alu = new Aluno () - é um objeto com a identidade alu;

2. Estado: O estado são as estruturas (atributos) de um objeto. Por exemplo, o objeto alu possui o atributo nome. Quando definimos um valor para esse atributo, nós definimos isso como:

alu.nome = "Paulo Souza";

3. Comportamento: É definido pelos métodos que pertencem ao objeto. Quando nós chamamos esses métodos de um objeto, ele realiza a ação que lhe foi solicitada:

alu.trancar(disciplina);

Exemplo: Conta Bancária

- Uma conta bancária é geralmente composta por um número, nome do titular e saldo. Podemos guardar essas informações em variáveis:

```
int numeroDaConta1 = 1;
string titularDaConta1 = "Joaquim José";
double saldoDaConta1 = 1500.0;
```

- Para representar outros correntistas, precisamos de novas variáveis:

```
int numeroDaConta2 = 2;
string titularDaConta2 = "Silva Xavier";
double saldoDaConta2 = 2500.0;
```

Classe

- Para evitar muitas variáveis espalhadas pelo código, podendo misturar essas informações dentro do programa, vamos criar uma classe CONTA:

```
class Conta
{

}
```

- Dentro da CONTA vamos colocar os **atributos**:

```
class Conta
{
    int numero;
    string titular;
    double saldo;
}
```

Atributos da Classe

- Porém, para que o código da aplicação possa ler e escrever nesses atributos, precisamos declará-los utilizando a palavra **public** :

```
class Conta
{
    // numero, titular e saldo são atributos do objeto
    public int numero;
    public string titular;
    public double saldo;
}
```

Objeto

- Para utilizarmos a classe que criamos dentro de uma aplicação windows form, precisamos criar uma nova conta no código do formulário, fazemos isso utilizando a instrução **new** do C#:

```
private void Button1_Click(object sender, EventArgs e)
{
    Conta c = new Conta();
}
```

Objeto

```
Conta c = new Conta();
```

A instrução **new** faz:

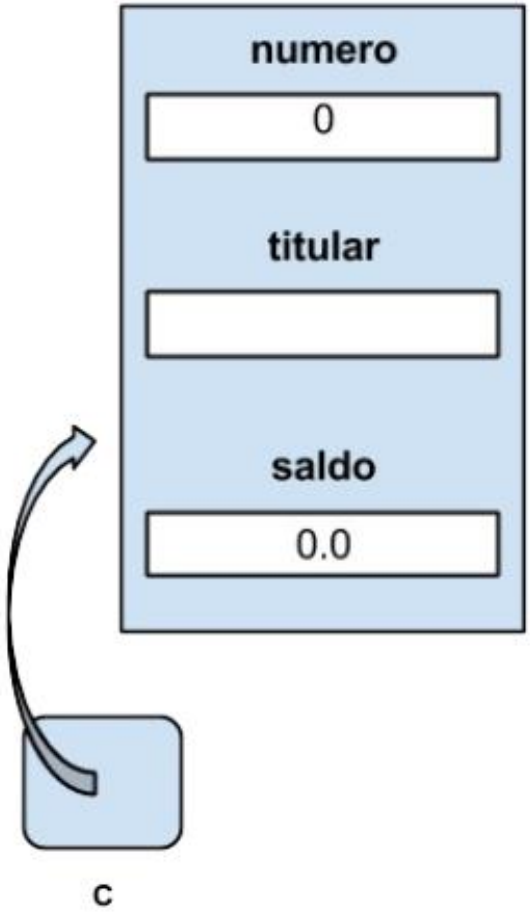
1. Criar uma nova instância (objeto) de Conta na memória, com espaco suficiente para todas as informações definidas dentro da classe.

2. Devolver a referência, uma seta que aponta para o objeto em memória, que será utilizada para manipularmos a Conta criada.

Podemos guardar essa referência dentro de uma variável do tipo Conta

Objeto

C tem a referência de uma variável do tipo Conta



C é um objeto da classe Conta

C é uma instância da classe Conta

Acessando atributos de objeto

- Para definirmos os valores dos atributos que serão armazenados na Conta, precisamos acessar o objeto que vive na memória.
- Fazemos isso utilizando o **operador .** do C#, informando qual é o atributo que queremos acessar.

```
private void Button1_Click(object sender, EventArgs e)
{
    Conta c = new Conta();
    c.numero = 1;
}
```

Acessando atributos de objeto

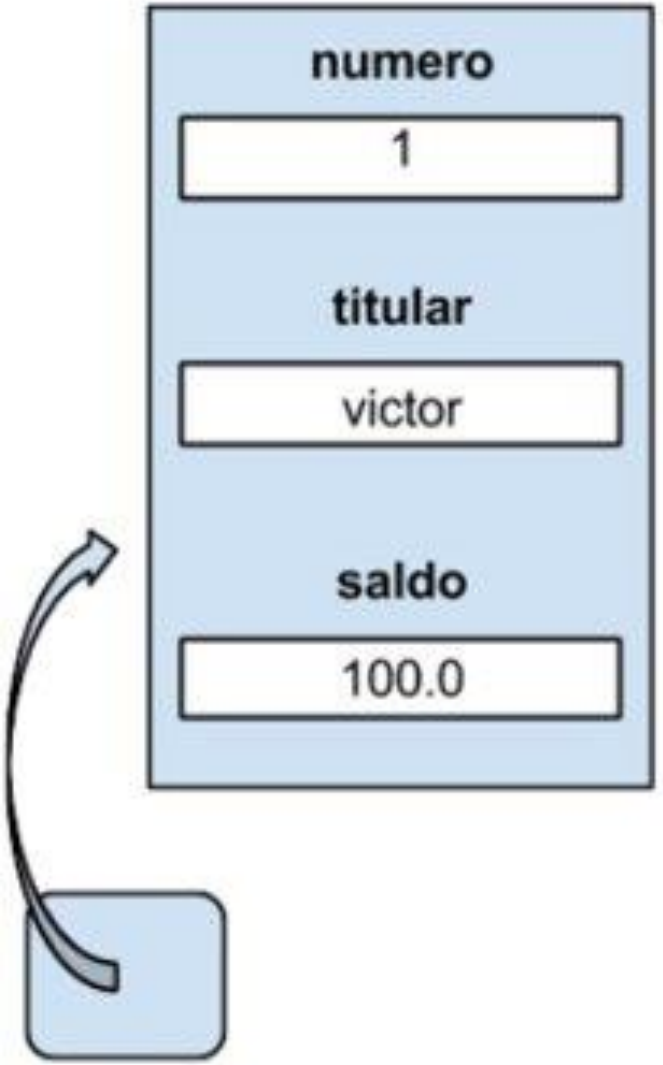
- Com esse código, estamos navegando na referência armazenada na variável c , e acessando o campo número do objeto Conta que vive na memória.
- Dentro desse campo colocamos o valor 1. Podemos fazer o mesmo para os outros campos da Conta :

```
private void Button1_Click(object sender, EventArgs e)
{
    Conta c = new Conta();
    c.numero = 1;
    c.titular = "victor";
    c.saldo = 100;
}
```

Acessando atributos de objeto

- Depois da execução desse código, teremos a seguinte situação na memória da aplicação:

- Veja que, quando utilizamos um objeto para guardar informações, **todos os atributos ficam agrupados** dentro de um único objeto na memória, e não espalhados dentro de diversas variáveis diferentes.



Métodos

- Além de atributos, os objetos também podem possuir métodos. Os métodos são blocos de código que isolam lógicas de negócio do objeto.
- Quando queremos passar um valor para um método, precisamos passar esse valor dentro dos parênteses da chamada do método.
- Para recebermos o valor que foi passado, precisamos declarar um argumento no método.
- O argumento é uma variável declarada dentro dos parênteses do método.
- Um método pode ter qualquer número de argumentos. Precisamos apenas separar a declaração das variáveis com uma vírgula.

Métodos

- No método Saca, queremos verificar o saldo da conta em que o método foi invocado.
- Para acessarmos a referência em que um determinado método foi chamado, utilizamos a palavra **this** .
- Então para acessarmos o saldo da conta, podemos utilizar this.saldo :
- A palavra-chave **this** refere-se à instância atual da classe

```
public void Saca(double valor)
{
    if (this.saldo >= valor)
    {
        this.saldo -= valor;
    }
}
```

Métodos

- Classe Conta com 3 atributos e 1 método

```
class Conta
{
    // numero, titular e saldo são atributos do objeto
    public int numero;
    public string titular;
    public double saldo;

    public void Saca(double valor)
    {
        if (this.saldo >= valor)
        {
            this.saldo -= valor;
        }
    }
}
```

Métodos

- Quando um método devolve um valor, o tipo do valor devolvido deve ficar antes do nome do método em sua declaração.
- Quando um método não devolve valor algum, utilizamos o tipo **void** .
- Dentro da implementação do método, devolvemos um valor utilizamos a palavra **return** seguida do valor que deve ser devolvido.

```
public bool Saca(double valor)
{
    if (this.saldo >= valor)
    {
        this.saldo -= valor;
        return true;
    }
    else
    {
        return false;
    }
}
```

Métodos

```
private void Button1_Click(object sender, EventArgs e)
{
    Conta c = new Conta();
    c.numero = 1;
    c.titular = "victor";
    c.saldo = 100;

    bool deuCerto = c.Saca(30.0);
    if (deuCerto)
    {
        MessageBox.Show("Saque realizado com sucesso");
    }
    else
    {
        MessageBox.Show("Saldo Insuficiente");
    }
}
```

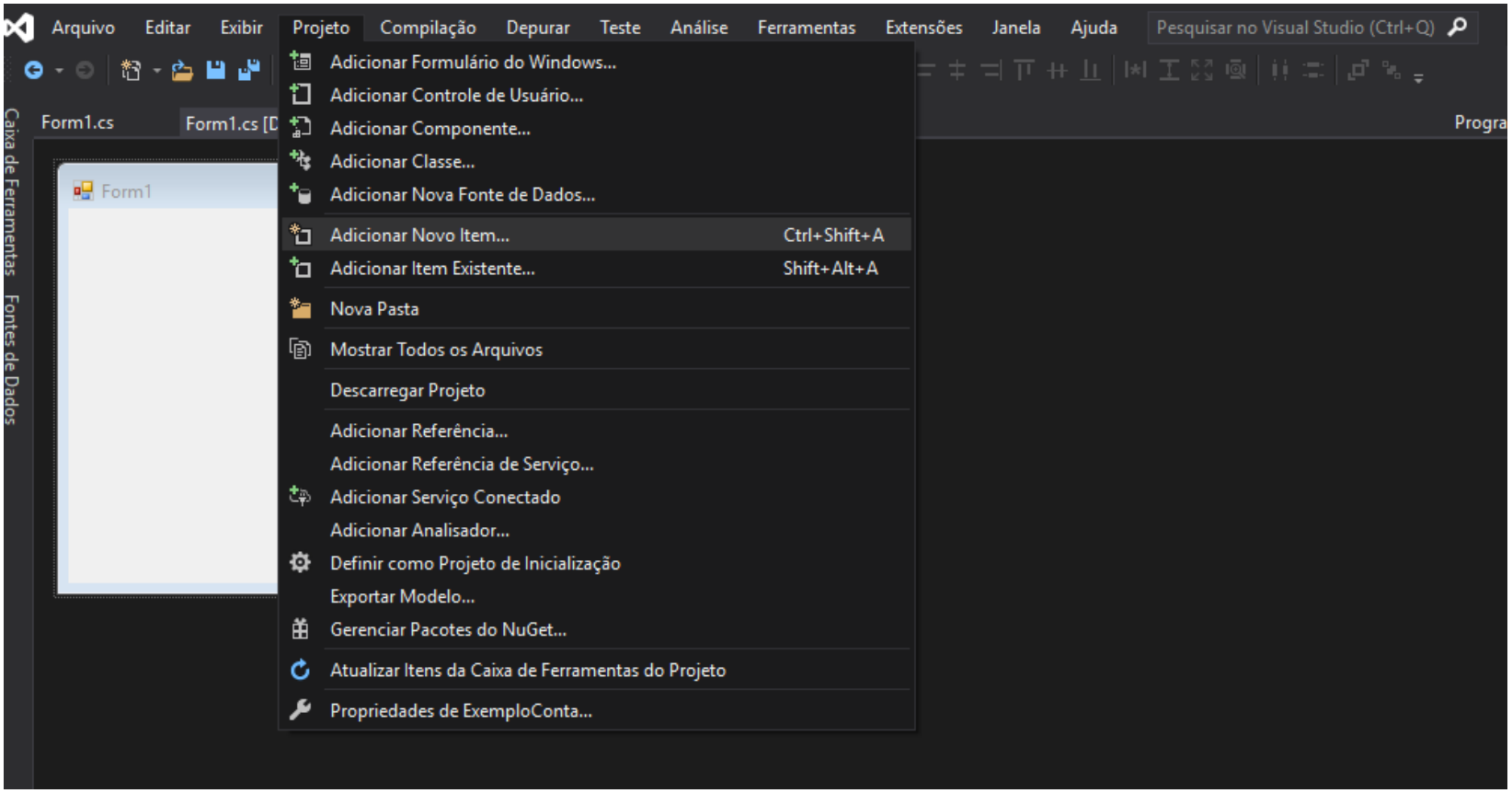
Métodos

```
private void Button1_Click(object sender, EventArgs e)
{
    Conta c = new Conta();
    c.numero = 1;
    c.titular = "victor";
    c.saldo = 100;

    if (c.Saca(30.0))
    {
        MessageBox.Show("Saque realizado com sucesso");
    }
    else
    {
        MessageBox.Show("Saldo Insuficiente");
    }
}
```

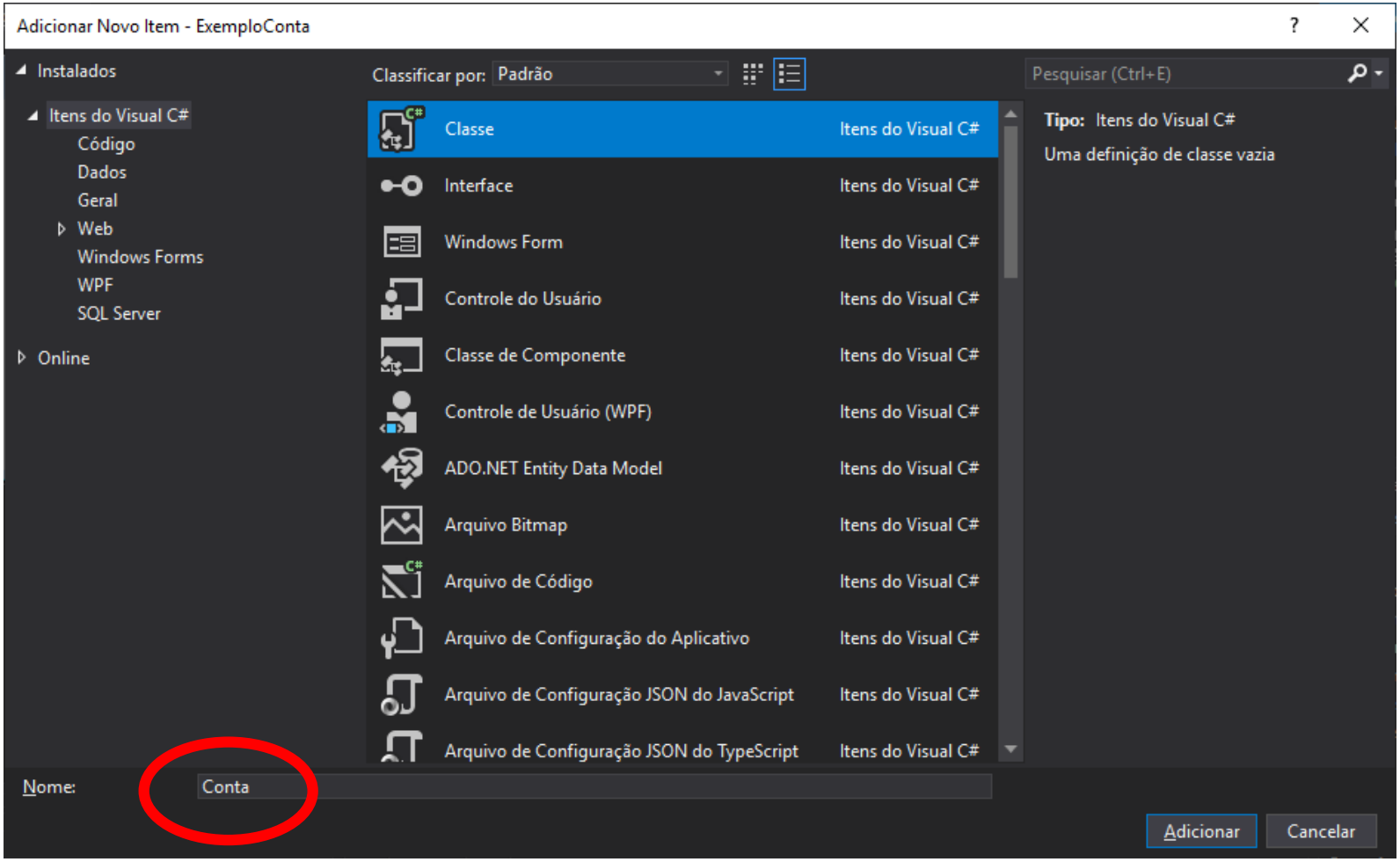
Criando Classe no Visual Studio

Projeto / Adicionar Novo Item

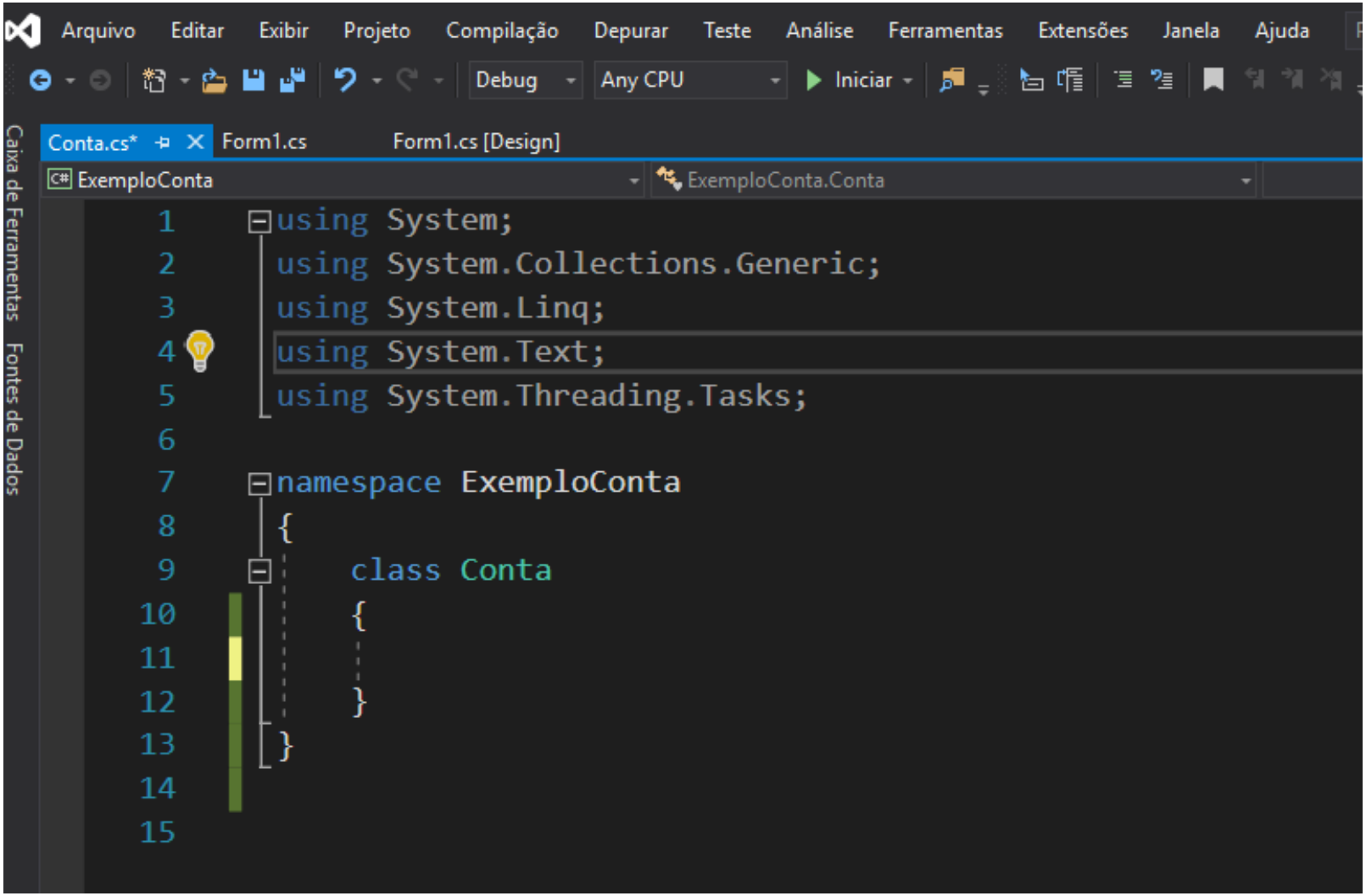


Criando Classe no Visual Studio

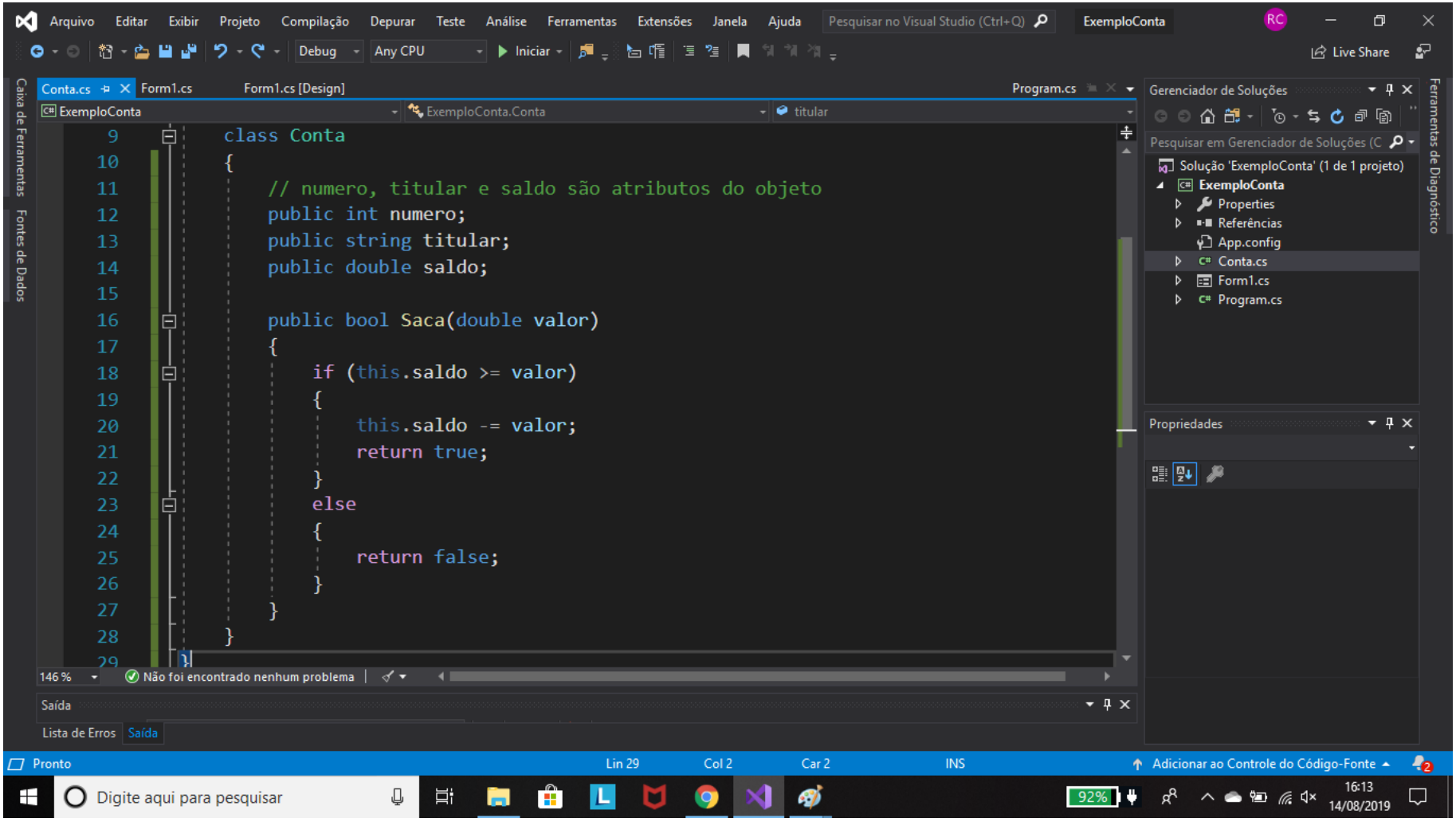
Projeto / Adicionar Novo Item / Classe / Nome:



Criando Classe no Visual Studio



Criando Classe no Visual Studio



Exercícios

1. Fazer o método deposita, informando o valor a ser transferido.
2. Criar 2 contas correntes diferentes
3. Fazer método transfere, informando a conta origem, valor e conta destino.
4. Fazer método saldo que exibe no Console o valor do saldo das duas contas correntes

IMPORTANTE: fazer estes exercícios dentro do método CLICK_

Composição de Classes

- Na composição de classe, temos uma instância de uma Classe sendo usada como Componente de outra Classe.
- Exemplo:

```
class Sistema
{

    Pessoa pessoa = new Pessoa();

}

class Pessoa
{

}
```

Composição de Classes

- Vamos criar um cliente do Banco:

```
class Cliente
{
    public string nome;
    public string cpf;
    public string rg;
    public string endereco;
}
```

Composição de Classes

- Na classe Conta, vamos associar à um cliente, ou seja, a conta vai ter uma referência ao cliente dono da conta:

```
class Conta
{
    // numero, titular e saldo são atributos do objeto
    public int numero;
    public Cliente titular;
    public double saldo;

    public bool Saca(double valor)
    {
        if (this.saldo >= valor)
```

Composição de Classes

- Quando formos criar uma conta, devemos antes criar um cliente para associa-lo à conta criada.

```
private void Button1_Click(object sender, EventArgs e)
{
    Cliente cli = new Cliente();
    cli.nome = "Victor";
    cli.cpf = "888888889";
    cli.endereco = "Rua XXXX, 21";
    cli.rg = "2222222222";

    Conta c = new Conta();
    c.numero = 1;
    c.saldo = 100;
    c.titular = cli;

    if (c.Saca(30.0))
    {
        MessageBox.Show("Saque realizado com sucesso");
```

Composição de Classes

- Outra forma de atribuir informações sobre o titular:

```
private void Button1_Click(object sender, EventArgs e)
{
    Cliente cli = new Cliente();

    Conta c = new Conta();
    c.numero = 1;
    c.saldo = 100;
    c.titular = cli;

    c.titular.nome = "Victor";
    c.titular.cpf = "888888889";
    c.titular.endereco = "Rua XXXX, 21";
    c.titular.rg = "2222222222";

    if (c.Saca(30.0))
    {
        MessageBox.Show("Saque realizado com sucesso");
    }
    else
```