

Paradigma de Orientação a Eventos

Mário Leite

...

A afirmação do Windows[®] no mercado em 1986 deu início a uma crescente demanda por sistemas baseados nesta nova plataforma; assim, surgiu a necessidade da criação dos chamados “*programas for Windows*”. Até então, o que existia eram programas rodando sob sistemas operacionais baseados em caracteres tais como: MS-DOS, DR-DOS e CP/M nas chamadas “telas pretas”, muito populares entre os programadores da época; e o MS-DOS ou simplesmente DOS (Sistema Operacional em Disco) era o preferido como representante dos “*programas for DOS*”. Com a chegada do Windows[®] os programadores se viram forçados a repensarem profundamente o modo de criar suas aplicações, pois já não era mais possível programar da mesma maneira e com a mesma lógica, até então no ambiente DOS, pois neste novo paradigma quem deveria comandar os eventos era o próprio usuário através de ações disparadas sobre os *controles* nas interfaces gráficas (janelas) e não o programa em si. Deste modo, até a famosa e clássica técnica “Top Down” teve que ser adaptada quando aplicada na criação do algoritmo de solução do problema, pois a preocupação passou a ser exclusivamente com os eventos que poderiam ser escolhidos e disparados pelo usuário quase que aleatoriamente. Esta mudança de paradigma na programação foi relativamente brusca e “pegou” muita gente desprevenida; muitos ainda defendiam a velha programação baseada exclusivamente no ambiente DOS (chamados de sauDOSistas). Entretanto, uma nova dúvida surgiu de imediato: *quais ferramentas deveriam ser usadas para criar aplicações nesse novo ambiente!?* A solução inicial foi usar o “poderoso” C como linguagem de desenvolvimento; entretanto, era um martírio para os desenvolvedores - mesmo para aplicações bem simples - pois além de enfrentar a codificação rigorosa do C exigia-se o **SDK** (Kit de Desenvolvimento de Software do Windows[®] distribuído pela Microsoft[®]). Mas o tempo era demasiadamente longo para vencer todas as etapas de criação de um programa “*for Windows*” por mais simples que fosse; por exemplo, mesmo para exibir a clássica frase “Hello World!” numa janela, era preciso mais de setenta linhas de código em C. Em princípio, somente as grandes empresas estavam adequadamente estruturadas para investir na produção e distribuição de *softwares* do tipo “*for Windows*“, mas para as pequenas empresas e programadores domésticos era muito difícil e arriscado, migrar para esse novo ambiente. Com estas aparentes dificuldades iniciais as *softhouses* reconheceram que deveriam investir em uma nova linguagem de programação que fosse de fácil aprendizado e aplicação; e a solução imediata foi usar o Visual Basic, criado pela própria Microsoft[®] em 1991, justamente para este fim; depois surgiu um concorrente imediato do VB: o Delphi, e mais tarde outras ferramentas. De qualquer forma, mesmo usando uma boa linguagem o profissional deveria mudar sua maneira de desenvolver programas, pois as características de uma aplicação para o ambiente Windows[®] são bem diferentes das aplicações para o ambiente DOS. Observe os dois esquemas (**figuras 1 e 2**) que ilustram as diferenças de funcionamento destes dois ambientes e como as aplicações são gerenciadas por cada um deles. Na **figura 1** - programas “*for DOS*” - o processamento é orientado por procedimentos; *iniciam, pedem os dados, executam, exibem os resultados e terminam*. Neste caso o programa toma o controle total sobre o ambiente, executando de maneira procedural seguindo uma lista de instruções escritas no programa. O usuário não tem como interferir no processamento de maneira a buscar dadas em outras fontes; o máximo que se consegue é colocar um programa do tipo TSR (Terminate-and-Stay-Resident) na memória. Já na **figura 2** - com três programas “*for Windows*” - o processamento é orientado a eventos, rodando num ambiente compartilhado; o programa não detém o controle do processo e nem dos recursos do computador. Neste segundo caso as rotinas são executadas de acordo com os eventos disparados na interface da aplicação; isto dá uma grande flexibilidade ao programador durante o processo de desenvolvimento do programa, já que poderá até interagir em *run time* com outros recursos oferecidos pelo ambiente, e por isto uma aplicação deste tipo é bem mais racional, além de ser mais fácil de operar.

A **figura 3a** mostra um programa-fonte para calcular o fatorial de um número, codificado em **Julia**, no ambiente de linha de comando desta ferramenta, como era feito nos primórdios da programação “*for DOS*”; por exemplo, com as primeiras linguagens: Fortran, Basic, Turbo C, etc. Neste caso foram necessárias

apenas cinco linhas de código (desconsiderando as linhas em branco) para resolver o clássico problema de calcular o fatorial de 20. Tudo muito simples, já que o programador tem que se concentrar apenas no algoritmo da solução do problema: nada mais do que isto! A **figura 3b** mostra o mesmo programa no IDE (Integrated Development Environment) do VS Code, agora oferecendo um certo “conforto” ao programador, o que não existia no início do desenvolvimento de programas no ambiente do DOS. Na verdade, um IDE é apenas uma “capa” para conter as linhas de código de um programa e com algumas opções para *editar/executar* de imediato ou salvar as linhas de código em arquivo que poderá ser executado mais tarde. Por outro lado, note que a complexidade aumentou muito para resolver o mesmo problema, já que para usar uma IDE é preciso que o programador conheça alguns detalhes desse ambiente de programação e não somente a lógica da solução em si. E neste caso foi empregado o VS Code; um ambiente de programação muito popular entre os desenvolvedores *front-end*.

A **figura 4a** mostra o código-fonte do mesmo programa desenvolvido no ambiente do Windows® e codificado na linguagem Visual Basic (ou simplesmente VB) que como mencionado anteriormente, foi a primeira ferramenta criada para desenvolver programas “*for Windows*”; neste caso é a versão 6 (**VB6**).

O código-fonte é mostrado na janela de codificação desta ferramenta, nesta figura. As três últimas figuras - **4b1**, **4b2** e **4b3** - apresentam situações típicas de um programa “*for Windows*”. A primeira mostra o *design* da interface da aplicação antes de ser executada; a segunda o programa rodando, mas ainda sem qualquer ação disparada na interface e a terceira com o usuário digitando um número na primeira caixa de texto e clicando no botão [**Executar**]. Esta ação executa as linhas de código da dentro da rotina denominada **Sub btnExecutar_Click()**, respondendo ao evento *clik* no primeiro botão; calculando o fatorial do número digitado na primeira caixa e exibindo o resultado na segunda caixa de texto. Além disto, um recurso extra do VB6, ilustrado nesta última figura, é a impressão dos valores parciais do fatorial diretamente na interface; o que é muito difícil em outras linguagens (mesmo com as versões mais modernas). E clicando no botão [**Fechar**] o programa executa o comando **End** da rotina **Sub btnFechar_Click()** e termina.

Conclusão: nos programas do tipo “*for DOS*” (aqui exemplificado com a linguagem **Julia**) o programador digita as linhas de código e em seguida executa (roda) o programa e pronto; o sistema dá o resultado e termina. Já nos programas “*for Windows*” (aqui implementado em **VB6**) só após os disparos de ações sobre os controles da interface em execução é que as linhas de código pertinentes àquela ação são executadas. Por isto, além das janelas foram exigidas muito mais linhas de código para escrever o mesmo programa com algumas instruções extras inseridas automaticamente como: “*Private btnExecutar_Click()*” que marca o início da rotina que executa o evento *click* no botão [**Executar**] e “*End Sub*” que marca o fim desta rotina; assim como *Dim* para declarar variáveis, o que não aparece no programa “*for DOS*” em **Julia**. E como foi mencionado acima, outras linguagens de programação “*for Windows*” também poderiam ser empregadas para implementar o programa para calcular o fatorial de um número: por exemplo, **C#**; mas, aí a implementação exigiria muito mais linhas de código, pois além de ser orientada a eventos esta linguagem também é orientada a objetos, o que faria a complexidade da solução aumentar mais ainda. E apenas para ilustrar o uso desta nova opção de linguagem, observe na **figura 5** de como seria a implementação da solução do mesmo problema em **C#**: trinta e cinco linhas de código foram necessárias: as primeiras nove linhas inseridas com a palavra-chave *using* só para importar as bibliotecas básicas necessárias, além de termos bem estranhos introduzidos automaticamente pela ferramenta, como: *Form namespace, class, public, private, partial, void* e rotinas com argumento bem esquisitos: o (*object, sender, EventArgs e*). Este é o preço que o programador paga para agradar o usuário e parecer mais “moderno” perante seus colegas, ou mesmo devido às exigências da *softhouses* desenvolvedora dos programas.

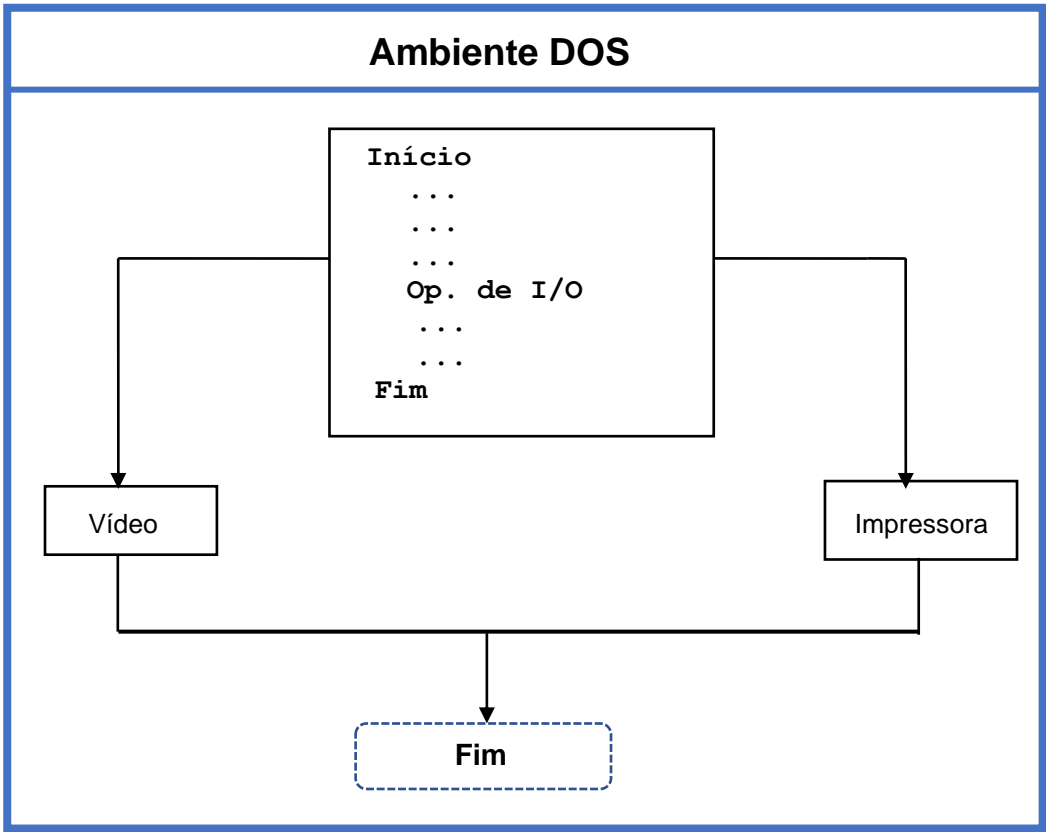


Figura 1 - Processamento no Ambiente DOS

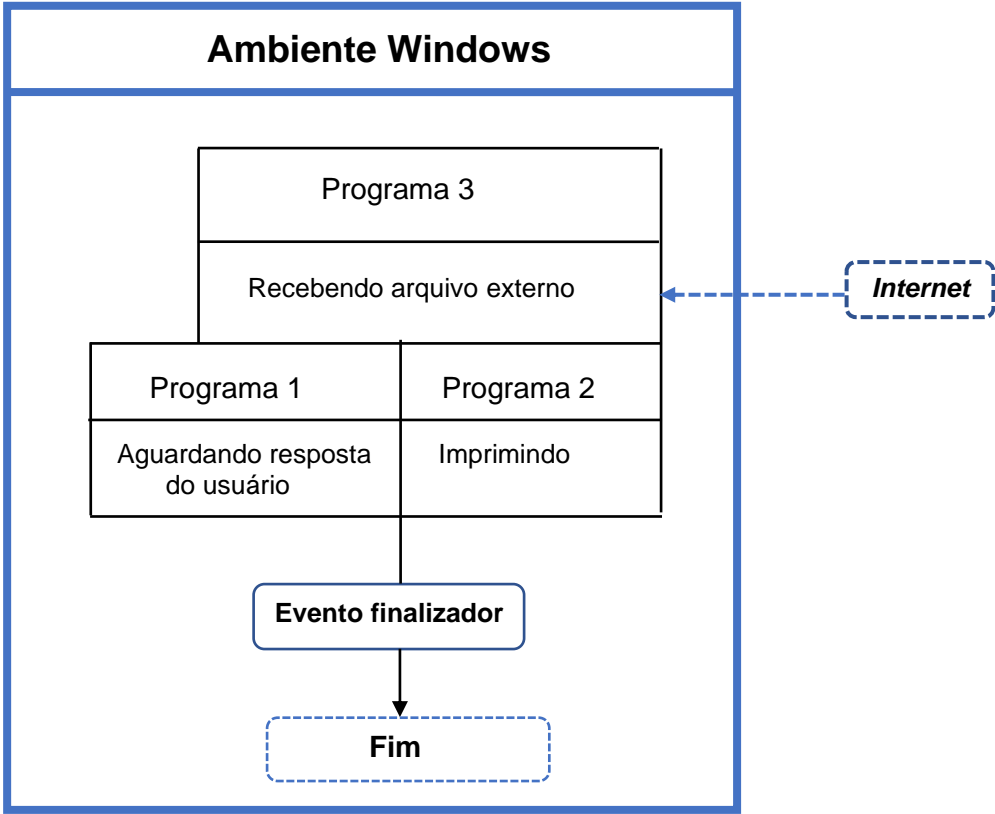


Figura 2 - Processamento no Ambiente Windows®



The screenshot shows the Julia REPL window titled "julia-1.2.0". The window has a dark background with a Julia logo in the top left. The text in the window is as follows:

```
Documentation: https://docs.julialang.org
Type "?" for help, "]"? for Pkg help.
Version 1.2.0 (2019-08-20)
Official https://julialang.org/ release

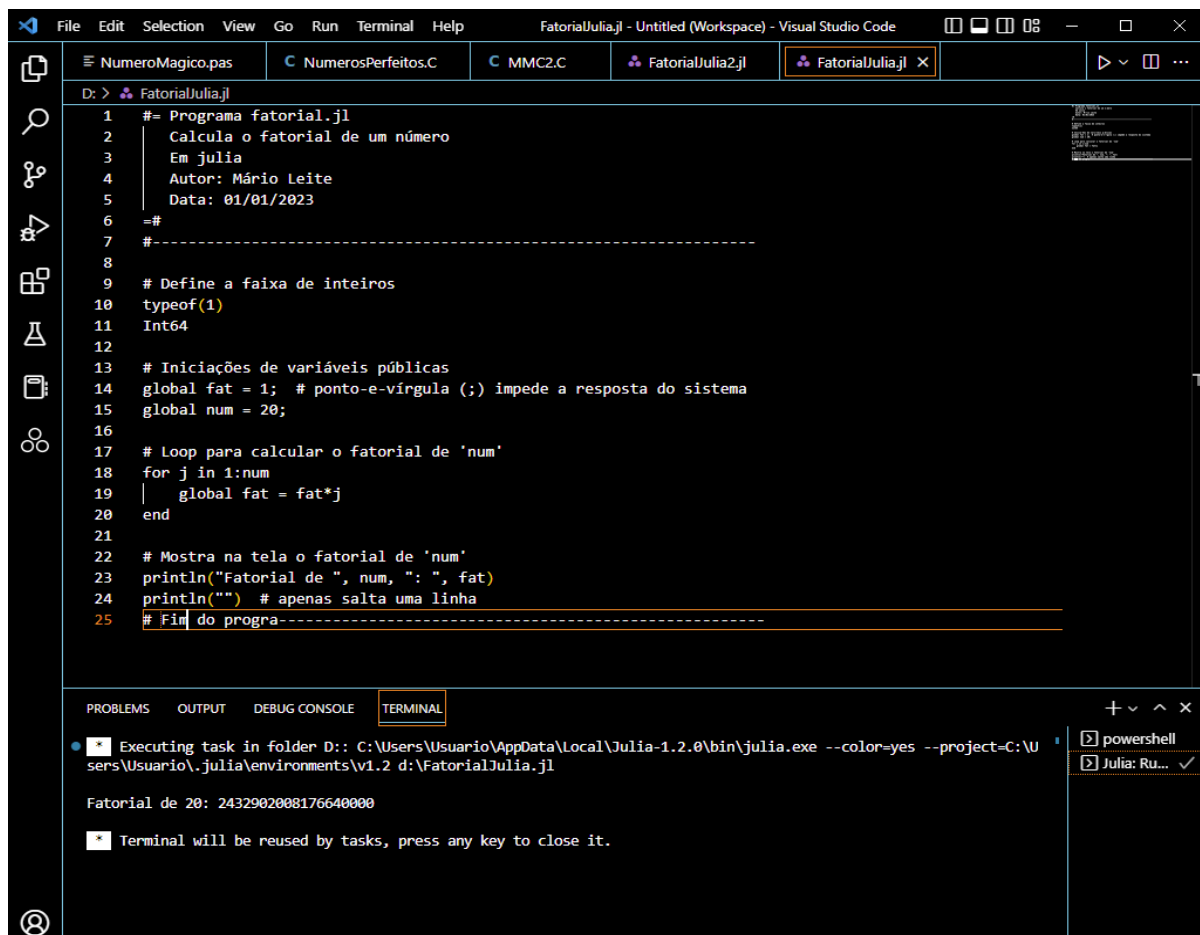
julia> typeof(1)
Int64

julia> global fat = 1;
julia> global num = 20;
julia> for j in 1:num
    global fat = fat*j
end

julia> println("Fatorial de ", num, ": ", fat)
Fatorial de 20: 2432902008176640000

julia> _
```

Figura 3a - Digitando e executando o programa no ambiente de linha de comando de “Julia”



The screenshot shows the Visual Studio Code IDE with the file "FatorialJulia.jl" open. The code in the file is as follows:

```
1  #= Programa fatorial.jl
2  | Calcula o fatorial de um número
3  | Em julia
4  | Autor: Mário Leite
5  | Data: 01/01/2023
6  |=#
7  |-----
8
9  | # Define a faixa de inteiros
10 | typeof(1)
11 | Int64
12 |
13 | # Iniciações de variáveis públicas
14 | global fat = 1; # ponto-e-vírgula (;) impede a resposta do sistema
15 | global num = 20;
16 |
17 | # Loop para calcular o fatorial de 'num'
18 | for j in 1:num
19 | | global fat = fat*j
20 | end
21 |
22 | # Mostra na tela o fatorial de 'num'
23 | println("Fatorial de ", num, ": ", fat)
24 | println("") # apenas salta uma linha
25 | # Fim do programa-----
```

The terminal at the bottom shows the execution of the program:

```
Executing task in folder D: C:\Users\Usuario\AppData\Local\Julia-1.2.0\bin\julia.exe --color=yes --project=C:\U
sers\Usuario\julia\environments\v1.2 d:\FatorialJulia.jl

Fatorial de 20: 2432902008176640000

Terminal will be reused by tasks, press any key to close it.
```

Figura 3b - Digitando e executando o programa em Julia no IDE do VS Code

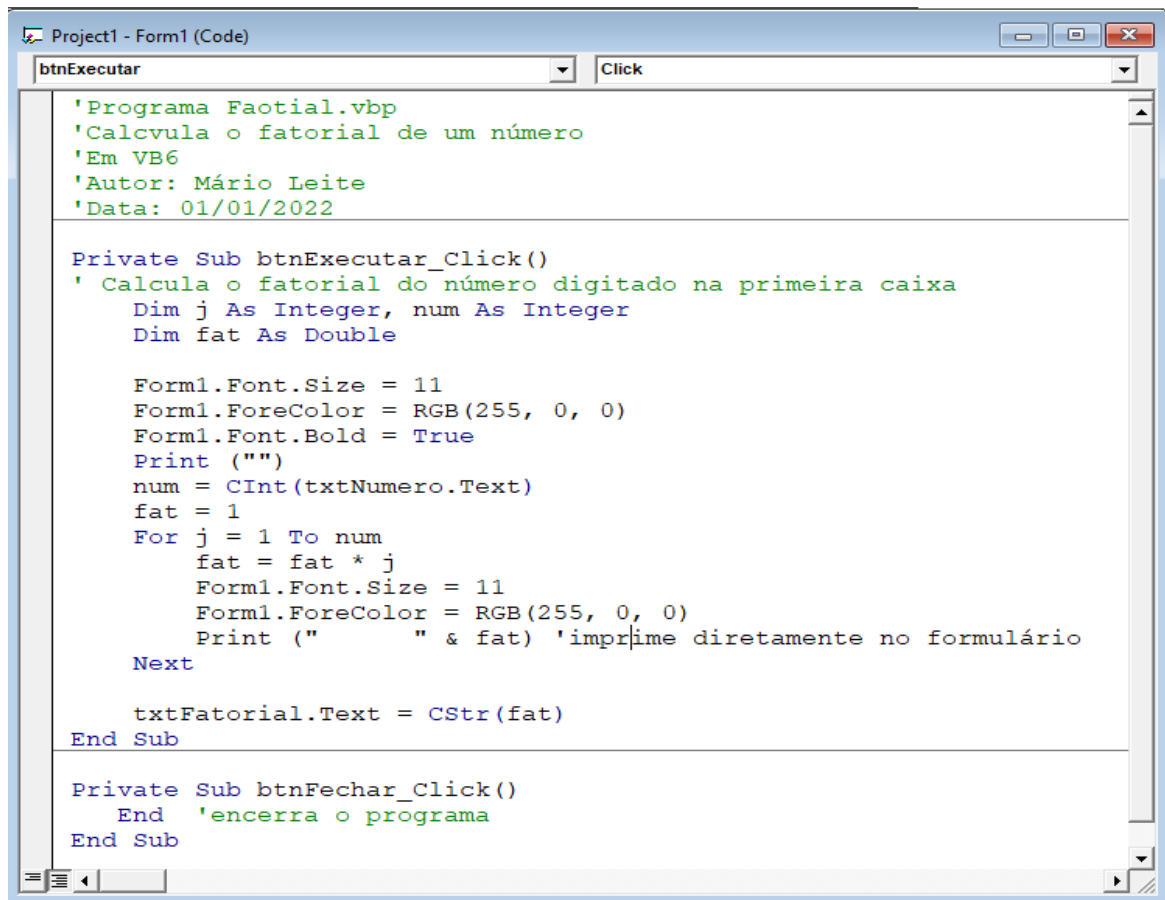


Figura 4a – Digitando o código-fonte do programa em VB6 no editor desta ferramenta

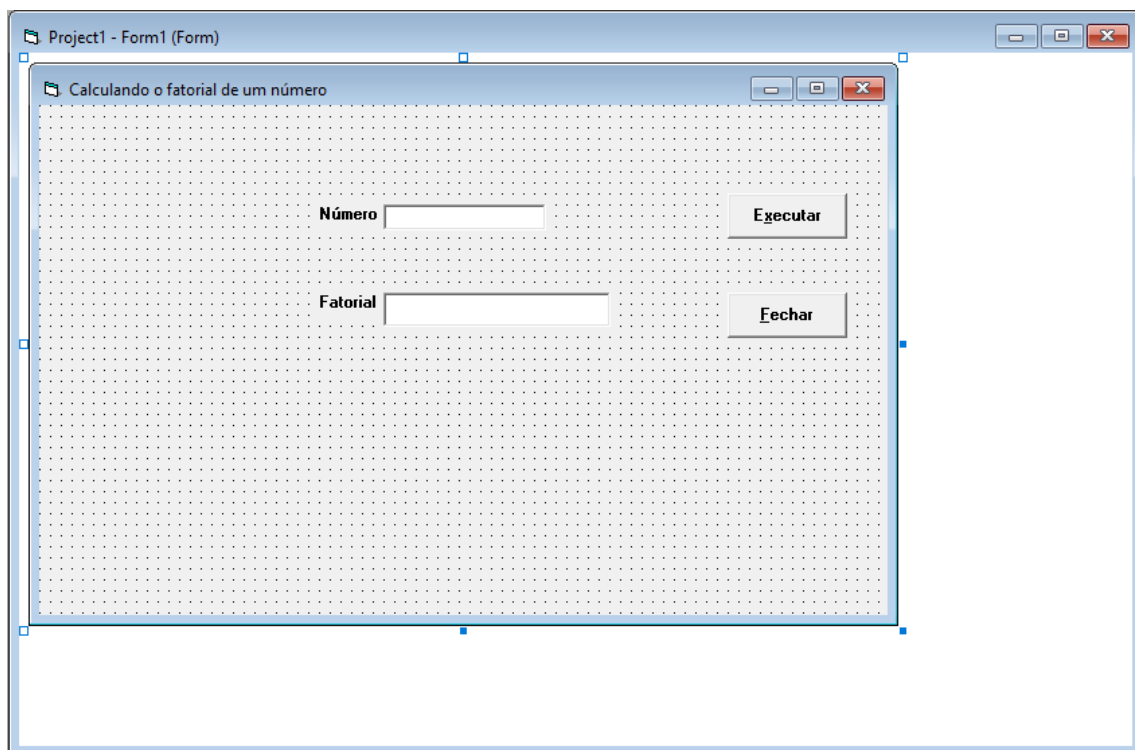


Figura 4b1 - interface do programa em VB6: em tempo de projeto

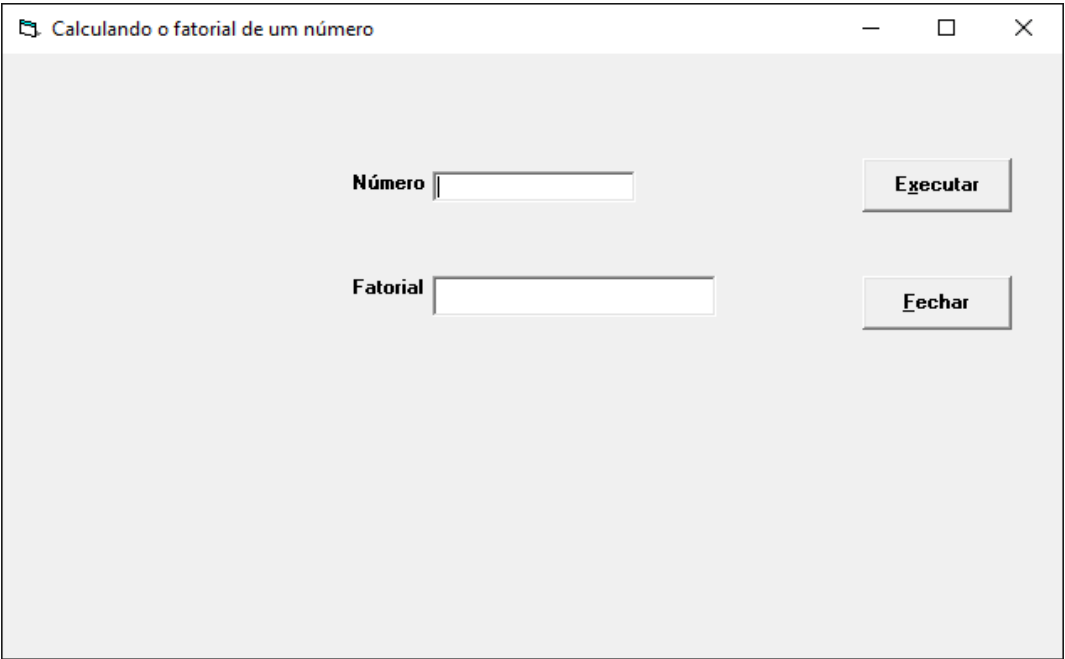


Figura 4b2 - interface do programa em VB6: em execução sem disparos de ações na interface

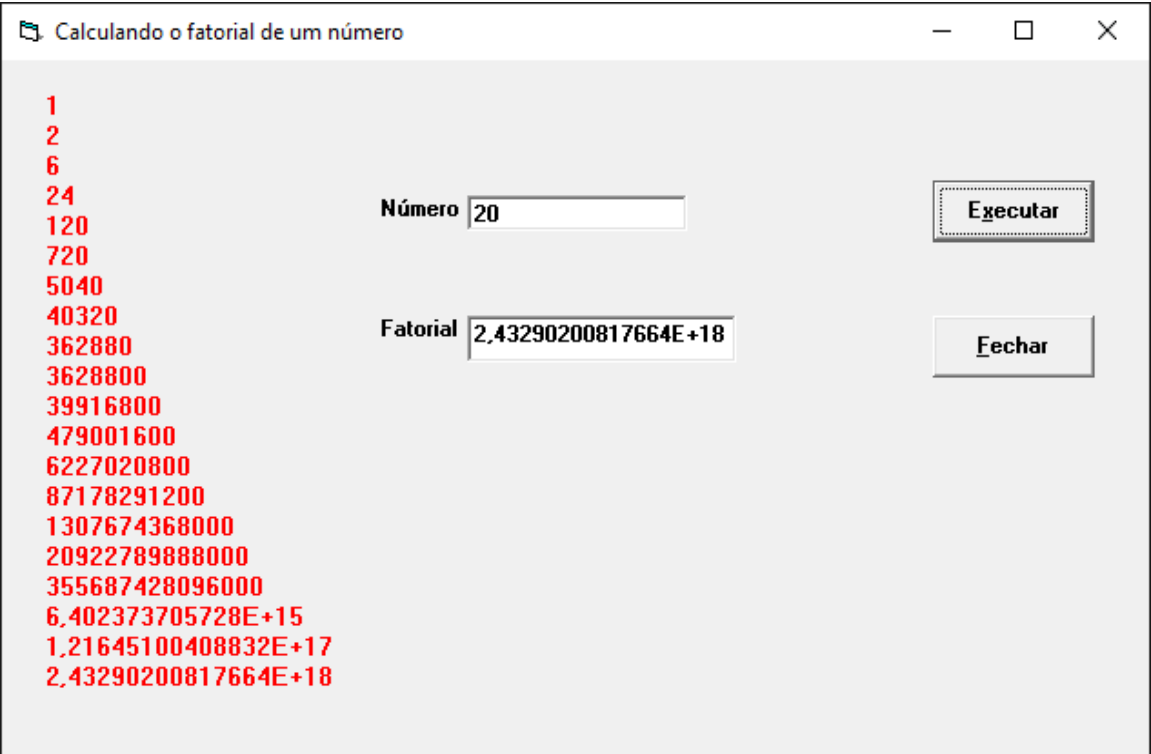
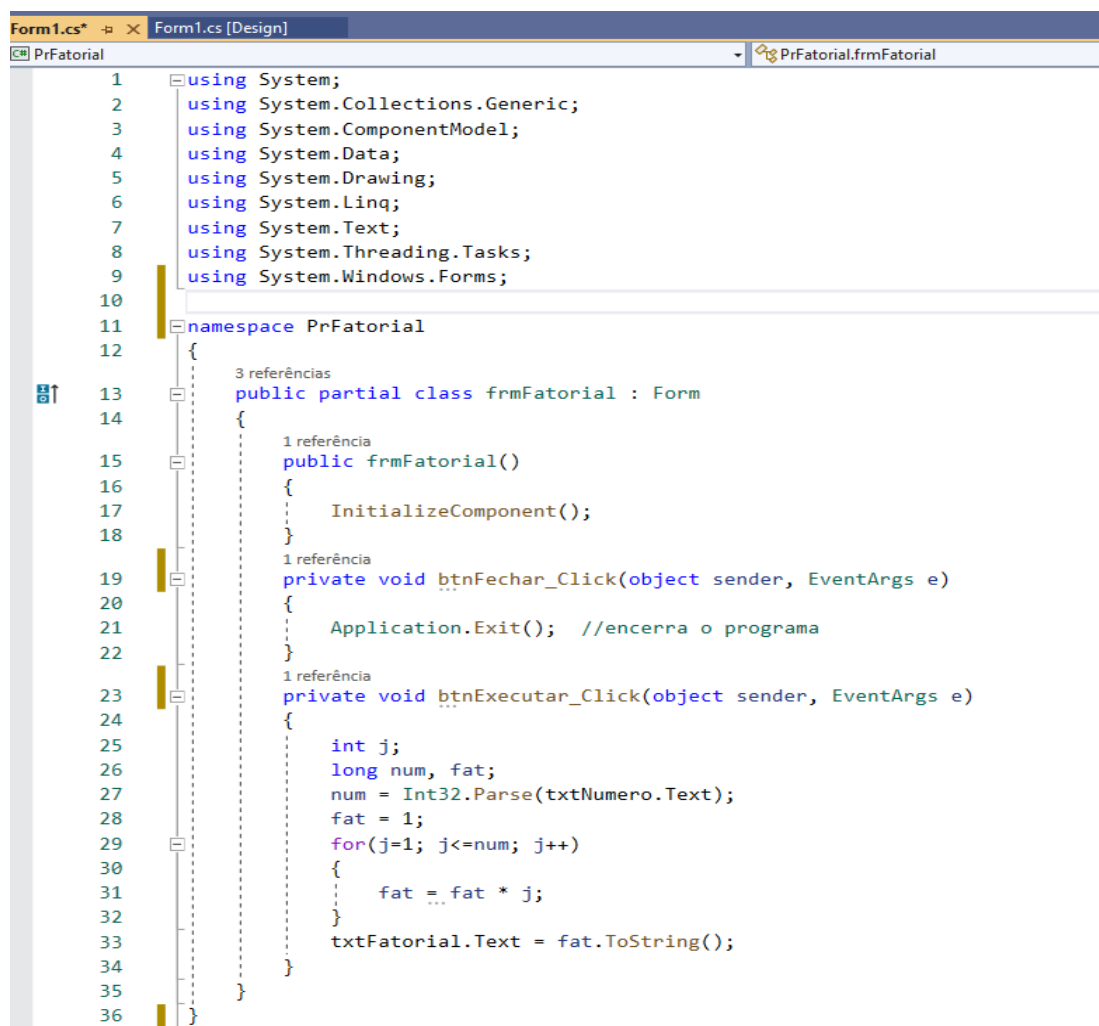


Figura 4b3 - interface do programa em VB6 em execução: após entrar o dado e clicar no botão [Executar]



```
1 using System;
2 using System.Collections.Generic;
3 using System.ComponentModel;
4 using System.Data;
5 using System.Drawing;
6 using System.Linq;
7 using System.Text;
8 using System.Threading.Tasks;
9 using System.Windows.Forms;
10
11 namespace PrFatorial
12 {
13     public partial class frmFatorial : Form
14     {
15         public frmFatorial()
16         {
17             InitializeComponent();
18         }
19
20         private void btnFechar_Click(object sender, EventArgs e)
21         {
22             Application.Exit(); //encerra o programa
23         }
24
25         private void btnExecutar_Click(object sender, EventArgs e)
26         {
27             int j;
28             long num, fat;
29             num = Int32.Parse(txtNumero.Text);
30             fat = 1;
31             for(j=1; j<=num; j++)
32             {
33                 fat = fat * j;
34             }
35             txtFatorial.Text = fat.ToString();
36         }
37     }
38 }
```

Figura 5 - Código-fonte do programa em C# na janela de código do Visual Studio 2022