

Fundamentos de Desenvolvimento com C#

Professor: arménio cardoso



Kaike Torres da silva

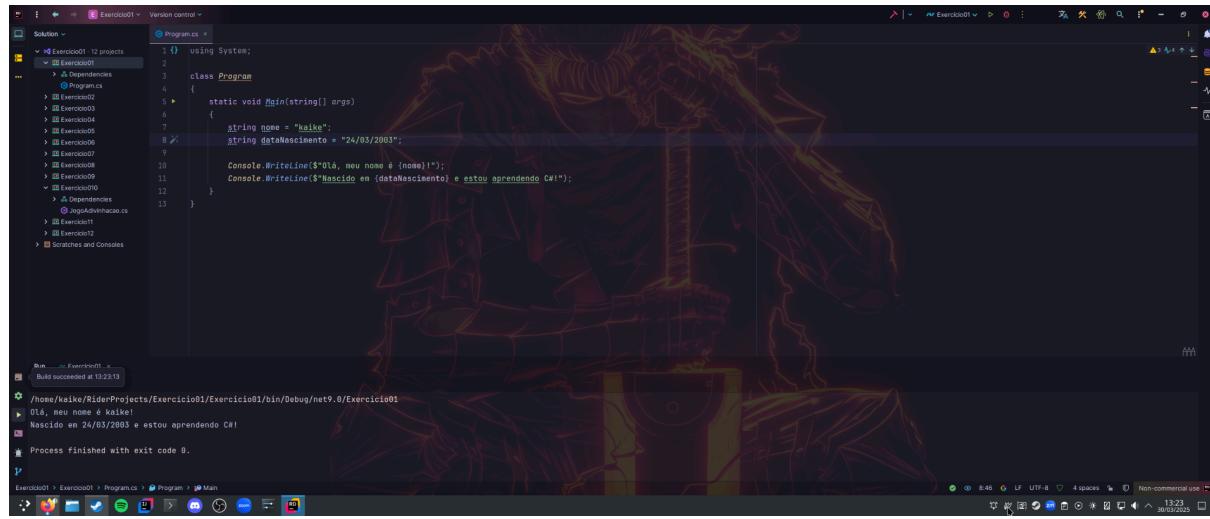
30.03.2025

Análise e Desenvolvimento de Sistemas (ADS)

INTRODUÇÃO

No desenvolvimento de sistemas bancários, a segurança e a integridade dos dados são essenciais para garantir a confiabilidade do serviço e proteger as informações dos clientes. Um dos princípios fundamentais da Programação Orientada a Objetos (POO) que contribui para essa segurança é o encapsulamento, que impede o acesso direto a atributos sensíveis, permitindo que sejam manipulados apenas por meio de métodos controlados.

Exercício 1 - Criando e Executando seu Primeiro Programa



A screenshot of the Visual Studio Code interface. The left sidebar shows a solution named 'Exercicio01' containing 12 projects. The main editor window displays the 'Program.cs' file with the following code:

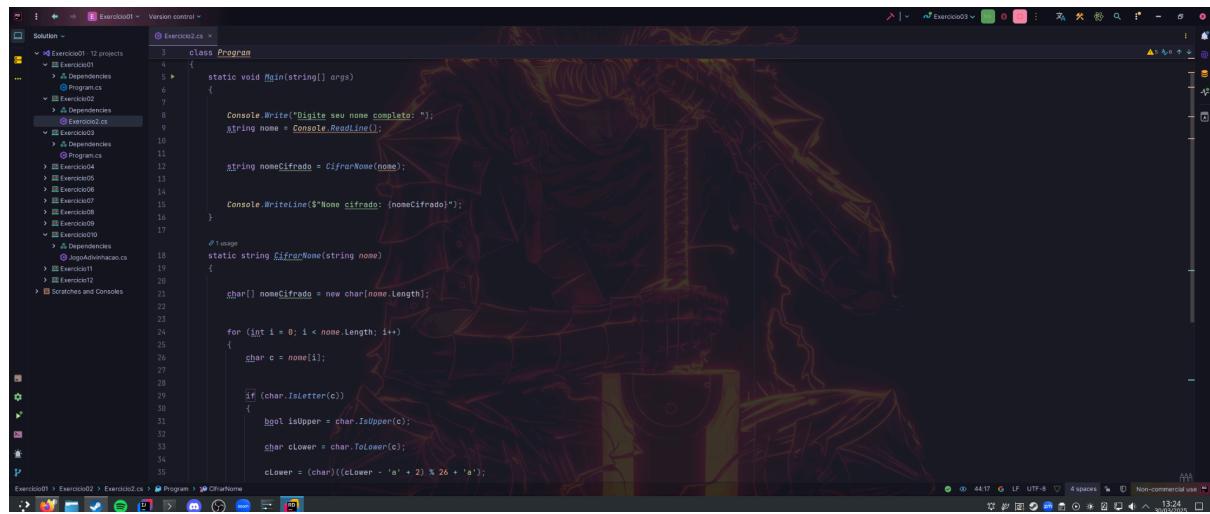
```
1  using System;
2
3  class Program
4  {
5      static void Main(string[] args)
6      {
7          string nome = "Keike";
8          string dataNascimento = "24/03/2003";
9
10         Console.WriteLine($"Olá, meu nome é {nome}!");
11         Console.WriteLine($"Nascido em {dataNascimento} e estou aprendendo C#!");
12     }
13 }
```

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

```
/home/keike/RideProjects/Exercicio01/Exercicio01/bin/Debug/net9.0/Exercicio01
Olá, meu nome é keike!
Nascido em 24/03/2003 e estou aprendendo C#!
```

The status bar at the bottom right indicates the file is non-commercial use.

Exercício 2 - Manipulação de Strings - Cifrador de Nome



A screenshot of the Visual Studio Code interface. The left sidebar shows a solution named 'Exercicio01' containing 12 projects. The main editor window displays the 'Exercicio2.cs' file with the following code:

```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Console.Write("Digite seu nome completo: ");
6          string nome = Console.ReadLine();
7
8          string nomeCifrado = CifrarNome(nome);
9
10         Console.WriteLine($"Nome cifrado: {nomeCifrado}");
11     }
12 }
13
14 // Usage
15 static string CifrarNome(string nome)
16 {
17     char[] nomeCifrado = new char[nome.Length];
18
19     for (int i = 0; i < nome.Length; i++)
20     {
21         char c = nome[i];
22
23         if (char.IsLetter(c))
24         {
25             bool isUpper = char.IsUpper(c);
26
27             char clower = char.ToLower(c);
28
29             clower = (char)((clower - 'a' + 2) % 26 + 'a');
30
31             nomeCifrado[i] = clower;
32         }
33     }
34
35 }
```

The status bar at the bottom right indicates the file is non-commercial use.

```

class Program
{
    static string CifrarNome(string nome)
    {
        for (int i = 0; i < nome.Length; i++)
        {
            char c = nome[i];

            if (char.IsUpper(c))
            {
                hgtol = char.IsUpper(c);

                char clower = char.ToLower(c);

                clower = (char)((clower - 'a' + 2) % 26 + 'a');

                if (hgtol)
                {
                    clower = char.ToUpper(clower);
                }
            }
            else
            {
                nomeCifrado[i] = c;
            }
        }
        return nomeCifrado;
    }
}

```

Output window:

```

/home/kaike/RiderProjects/Exercicio01/Exercicio02/bin/Debug/net9.0/Exercicio02
Digite seu nome completo: kaike torres da silva
Nome cifrado: mckmg vqtgg fc uknxc

```

Process finished with exit code 0.

Exercício 3 - Calculadora de Operações Matemáticas

```

class Program
{
    static void Main(string[] args)
    {
        double numero1, numero2;

        Console.WriteLine("Digite o primeiro número:");
        while (!double.TryParse(Console.ReadLine(), out numero1))
        {
            Console.WriteLine("Entrada inválida. Digite um número válido:");
        }

        Console.WriteLine("Digite o segundo número:");
        while (!double.TryParse(Console.ReadLine(), out numero2))
        {
            Console.WriteLine("Entrada inválida. Digite um número válido:");
        }

        Console.WriteLine("Escolha a operação:");
        Console.WriteLine("1. Soma");
        Console.WriteLine("2. Subtração");
        Console.WriteLine("3. Multiplicação");
        Console.WriteLine("4. Divisão");

        int operacao;
        while (true)
        {
            if (int.TryParse(Console.ReadLine(), out operacao) && operacao >= 1 && operacao <= 4)
                break;
            Console.WriteLine("Escolha uma operação válida (1-4):");
        }

        double resultado = 0;
    }
}

```

Output window:

```

/home/kaike/RiderProjects/Exercicio01/Exercicio03/bin/Debug/net9.0/Exercicio03

```

```

class Program
{
    static void Main(string[] args)
    {
        switch (operacao)
        {
            case 1:
                resultado = numero1 - numero2;
                break;
            case 2:
                resultado = numero1 * numero2;
                break;
            case 3:
                if (numero2 == 0)
                {
                    Console.WriteLine("Erro: Divisão por zero não é permitida.");
                    return;
                }
                resultado = numero1 / numero2;
                break;
            case 4:
                Console.WriteLine($"Resultado: {resultado}");
                break;
        }
    }
}

```

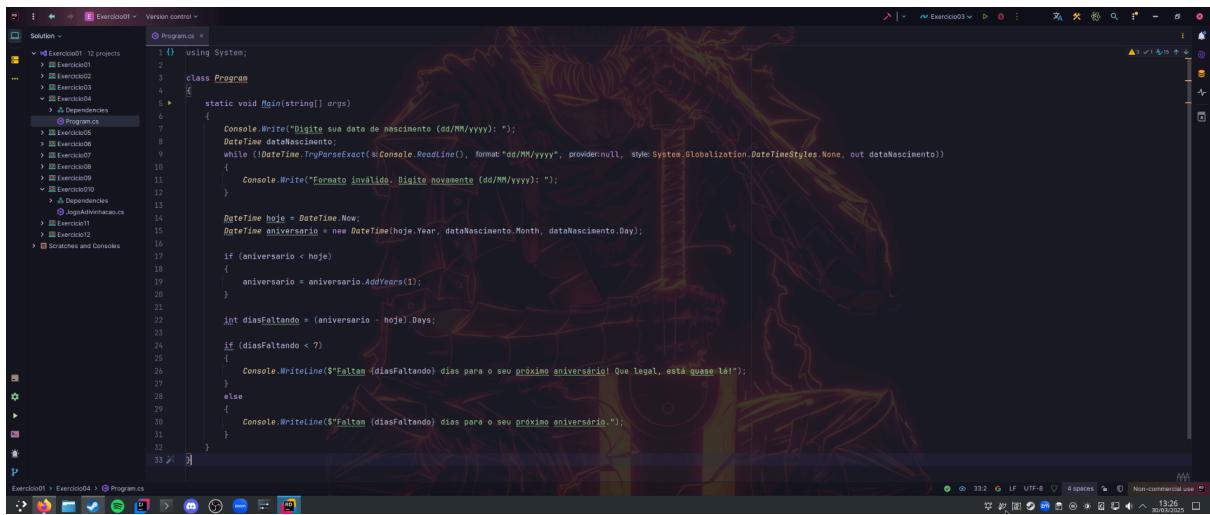
Output window:

```

/home/kaike/RiderProjects/Exercicio01/Exercicio03/bin/Debug/net9.0/Exercicio03
Digite o primeiro número: 2
Digite o segundo número: 4
Escolha a operação:
1. Soma
2. Subtração
3. Multiplicação
4. Divisão
1
Resultado: 8

```

Exercício 4 - Manipulação de Datas - Dias até o Próximo Aniversário



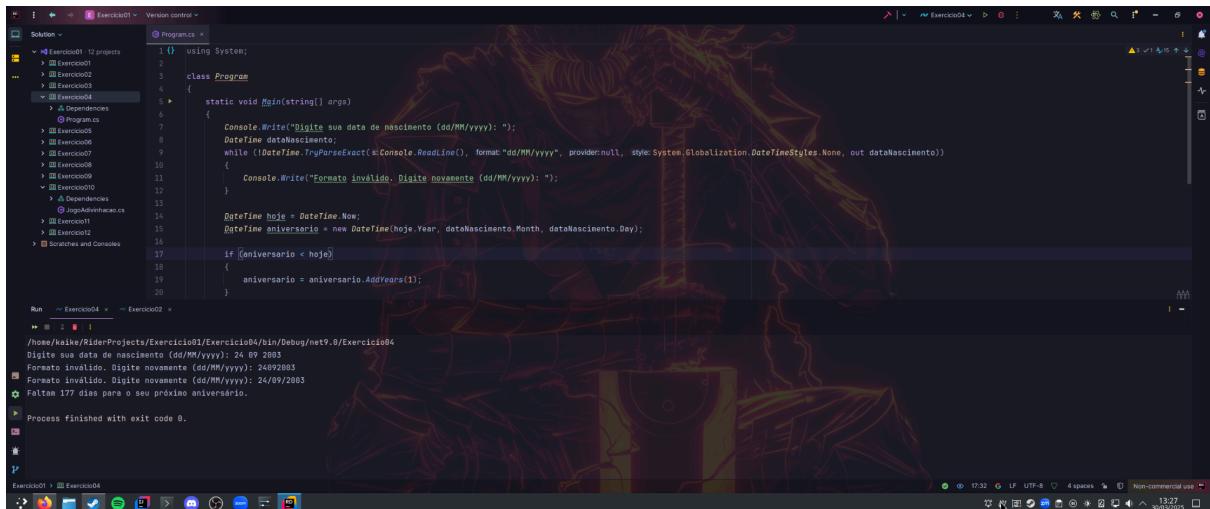
```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Digite sua data de nascimento (dd/MM/yyyy): ");
        DateTime dataNascimento;
        while (!DateTime.TryParseExact(Console.ReadLine(), "dd/MM/yyyy", providerNull, style: System.Globalization.DateTimeStyles.None, out dataNascimento))
        {
            Console.WriteLine("Formato invalido. Digite novamente (dd/MM/yyyy): ");
        }

        DateTime hoje = DateTime.Now;
        DateTime aniversario = new DateTime(hoje.Year, dataNascimento.Month, dataNascimento.Day);

        if (aniversario < hoje)
        {
            aniversario = aniversario.AddYears(1);
        }

        int diasFaltando = (aniversario - hoje).Days;

        if (diasFaltando < 7)
        {
            Console.WriteLine($"Faltam {diasFaltando} dias para o seu próximo aniversário! Que legal, está quase lá!");
        }
        else
        {
            Console.WriteLine($"Faltam {diasFaltando} dias para o seu próximo aniversário.");
        }
    }
}
```



```
using System;
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Digite sua data de nascimento (dd/MM/yyyy): ");
        DateTime dataNascimento;
        while (!DateTime.TryParseExact(Console.ReadLine(), "dd/MM/yyyy", providerNull, style: System.Globalization.DateTimeStyles.None, out dataNascimento))
        {
            Console.WriteLine("Formato invalido. Digite novamente (dd/MM/yyyy): ");
        }

        DateTime hoje = DateTime.Now;
        DateTime aniversario = new DateTime(hoje.Year, dataNascimento.Month, dataNascimento.Day);

        if (aniversario < hoje)
        {
            aniversario = aniversario.AddYears(1);
        }

        int diasFaltando = (aniversario - hoje).Days;

        if (diasFaltando < 7)
        {
            Console.WriteLine($"Faltam {diasFaltando} dias para o seu próximo aniversário! Que legal, está quase lá!");
        }
        else
        {
            Console.WriteLine($"Faltam {diasFaltando} dias para o seu próximo aniversário.");
        }
    }
}
```

```
/home/keike/RiderProjects/Exercicio01/Exercicio04/bin/Debug/net9.0/Exercicio04
Digite sua data de nascimento (dd/MM/yyyy): 24/09/2003
Formato invalido. Digite novamente (dd/MM/yyyy): 24/09/2003
Formato invalido. Digite novamente (dd/MM/yyyy): 24/09/2003
Faltam 177 dias para o seu próximo aniversário.

Process finished with exit code 0.
```

Exercício 5 - Tempo Restante para Conclusão do Curso - Diferença Entre Datas

The image displays two side-by-side screenshots of the Visual Studio IDE. Both screenshots show a solution named 'Exercicio01' containing 12 projects.

Left Screenshot (CalculadorTempoRestante.cs):

```
public class CalculadorTempoRestante
{
    public DateTime DataAtual { get; set; }

    public DateTime DataFormatura { get; set; }

    public string CalcularTempoRestante()
    {
        if (DataAtual > DateTime.Now)
        {
            return "Erro: A data informada não pode ser no futuro!";
        }

        if (DateFormatura < DateTime.Now)
        {
            return "Parabéns! Você já deveria estar formado!";
        }

        var anosRestantes = DataFormatura.Year - DataAtual.Year;
        var mesesRestantes = DataFormatura.Month - DataAtual.Month;
        var diasRestantes = DataFormatura.Day - DataAtual.Day;

        if (diasRestantes < 0)
        {
            mesesRestantes--;
            diasRestantes += DateTime.DaysInMonth(DataAtual.Year, DataAtual.Month);
        }

        if (mesesRestantes < 0)
        {
            anosRestantes--;
            mesesRestantes += 12;
        }
    }
}
```

Right Screenshot (Program.cs):

```
using Exercicio05;
class Program
{
    static void Main(string[] args)
    {
        Console.Write("Digite a data atual (dd/MM/yyyy): ");
        DateTime dataAtual;
        while (!DateTime.TryParseExact(Console.ReadLine(), "dd/MM/yyyy", provider: null, style: System.Globalization.DateTimeStyles.None, out dataAtual))
        {
            Console.WriteLine("Formato inválido. Digite novamente (dd/MM/yyyy): ");
        }

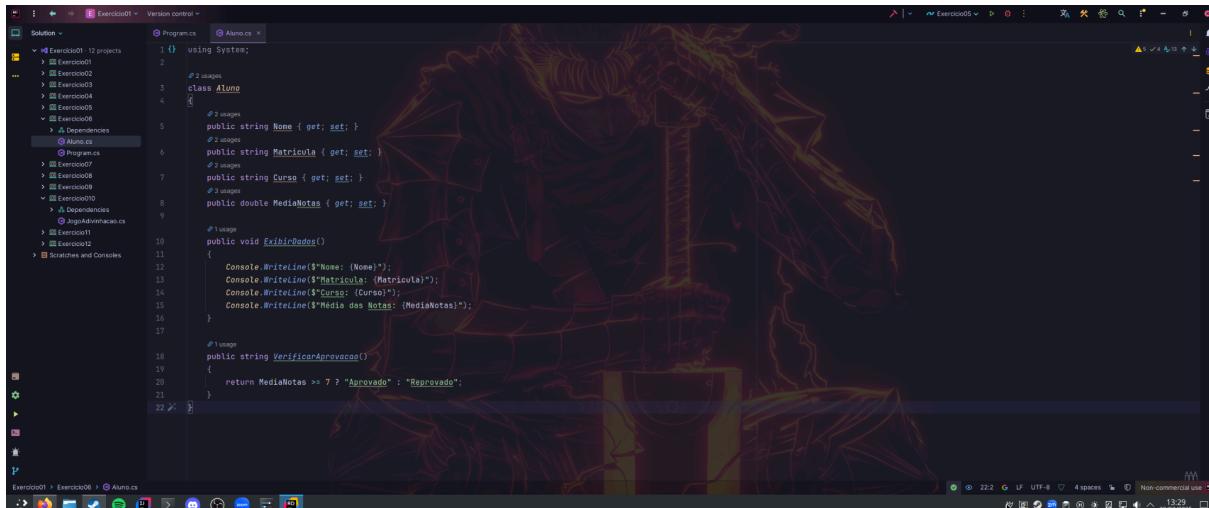
        DateTime dataFormatura = new DateTime(2026, month: 12, day: 15);

        CalculadorTempoRestante calculador = new CalculadorTempoRestante();
        {
            DataAtual = dataAtual,
            DataFormatura = dataFormatura
        };

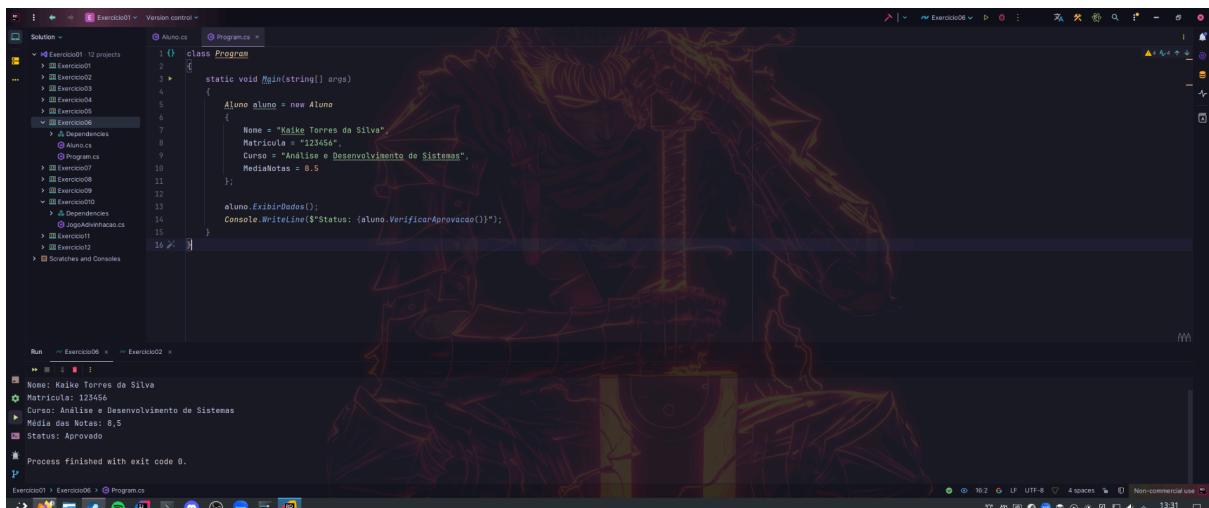
        Console.WriteLine(calculador.CalcularTempoRestante());
    }
}
```

The bottom status bar of both screenshots indicates the file path 'Exercicio01 > Exercicio05 > Program.cs' and the current time '13:27'.

Exercício 6 - Cadastro de Alunos



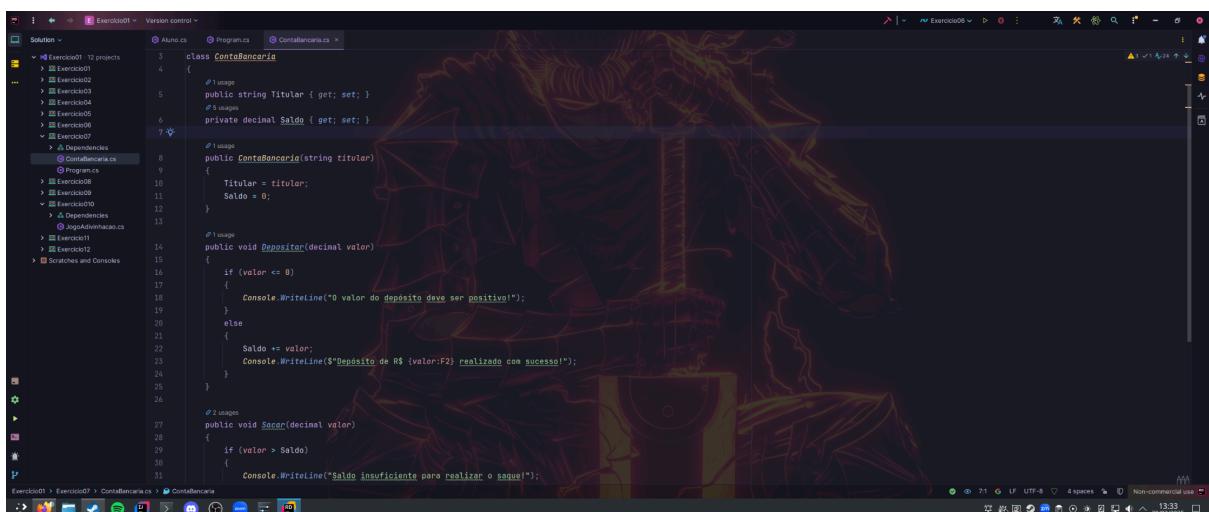
```
1  using System;
2
3  class Aluno
4  {
5      public string Nome { get; set; }
6
7      public string Matricula { get; set; }
8
9      public string Curso { get; set; }
10
11     public double MediaNotas { get; set; }
12
13     public void ExibirDados()
14     {
15         Console.WriteLine($"Nome: {Nome}");
16         Console.WriteLine($"Matrícula: {Matricula}");
17         Console.WriteLine($"Curso: {Curso}");
18         Console.WriteLine($"Média das Notas: {MediaNotas}");
19     }
20
21     public string VerificarAprovacao()
22     {
23         return MediaNotas >= 7 ? "Aprovado" : "Reprovado";
24     }
25 }
```



```
1  class Program
2  {
3      static void Main(string[] args)
4      {
5          Aluno aluno = new Aluno
6          {
7              Nome = "Keike Torres da Silva",
8              Matricula = "123456",
9              Curso = "Análise e Desenvolvimento de Sistemas",
10             MediaNotas = 8.5
11         };
12
13         aluno.ExibirDados();
14         Console.WriteLine($"Status: {aluno.VerificarAprovacao()}");
15     }
16 }
```

```
Name: Keike Torres da Silva
Matrícula: 123456
Curso: Análise e Desenvolvimento de Sistemas
Média das Notas: 8,5
Status: Aprovado
```

Exercício 7 - Banco Digital (Encapsulamento)



```
1  class ContaBancaria
2  {
3      public string Titular { get; set; }
4
5      private decimal Saldo { get; set; }
6
7      public void Depositar(decimal valor)
8      {
9          if (valor <= 0)
10          {
11              Console.WriteLine("O valor do depósito deve ser positivo!");
12          }
13          else
14          {
15              Saldo += valor;
16              Console.WriteLine($"Depósito de R$ {valor:F2} realizado com sucesso!");
17          }
18      }
19
20      public void Sacar(decimal valor)
21      {
22          if (valor > Saldo)
23          {
24              Console.WriteLine("Saldo insuficiente para realizar o saque!");
25          }
26      }
27  }
```

```
class Program
{
    static void Main(string[] args)
    {
        // Criação do objeto ContaBancaria
        ContaBancaria conta = new ContaBancaria("titular: João Silva");

        // Realização de transações
        conta.Depositar(500);
        conta.ExibirSaldo();

        conta.Sacar(700); // Saque com valor maior que o saldo
        conta.Sacar(200); // Saque válido
        conta.ExibirSaldo();
    }
}
```

Output window:

```
/home/kaile/RiderProjects/Exercicio01/Exercicio07/bin/Debug/net9.0/Exercicio07
Depósito de R$ 500,00 realizado com sucesso!
Saldo atual: R$ 500,00
Saldo insuficiente para realizar o saque!
Saque de R$ 200,00 realizado com sucesso!
Saldo atual: R$ 300,00

Process finished with exit code 0.
```

Exercício 8 - Cadastro de Funcionários (Herança)

```
class Funcionario
{
    public string Cargo { get; set; }
    public double SalarioBase { get; set; }

    public Funcionario(string nome, string cargo, double salarioBase)
    {
        Nome = nome;
        Cargo = cargo;
        SalarioBase = salarioBase;
    }

    public virtual double CalcularSalario()
    {
        return SalarioBase;
    }
}
```

Output window:

```
/home/kaile/RiderProjects/Exercicio01/Exercicio07/bin/Debug/net9.0/Exercicio07
Depósito de R$ 500,00 realizado com sucesso!
Saldo atual: R$ 500,00
Saldo insuficiente para realizar o saque!
Saque de R$ 200,00 realizado com sucesso!
Saldo atual: R$ 300,00

Process finished with exit code 0.
```

```
class Gerente : Funcionario
{
    public Gerente(string nome, double salarioBase)
        : base(nome, cargo: "Gerente", salarioBase) {}

    public override double CalcularSalario()
    {
        return SalarioBase * 1.2; // Bônus de 20%
    }
}
```

Output window:

```
/home/kaile/RiderProjects/Exercicio01/Exercicio07/bin/Debug/net9.0/Exercicio07
Depósito de R$ 500,00 realizado com sucesso!
Saldo atual: R$ 500,00
Saldo insuficiente para realizar o saque!
Saque de R$ 200,00 realizado com sucesso!
Saldo atual: R$ 300,00

Process finished with exit code 0.
```

A screenshot of the Visual Studio IDE interface. The solution explorer shows multiple projects under 'Exercicio01'. The code editor displays 'Program.cs' with the following content:

```
1 class Program
2 {
3     static void Main()
4     {
5         Funcionario funcionario = new Funcionario(nome: "Carlos Silva", cargo: "Analista", salarioBase: 3000);
6         Gerente gerente = new Gerente(nome: "Ana Souza", salarioBase: 5000);
7
8         Console.WriteLine($"Funcionario: {funcionario.Nome}, Cargo: {funcionario.Cargo}, Salário: R${funcionario.CalcularSalario():F2}");
9         Console.WriteLine($"Gerente: {gerente.Nome}, Cargo: {gerente.Cargo}, Salário: R${gerente.CalcularSalario():F2}");
10    }
11 }
```

The output window shows the results of the program execution:

```
/home/kaike/RiderProjects/Exercicio01/Exercicio08/bin/Debug/net9.0/Exercicio08
Funcionario: Carlos Silva, Cargo: Analista, Salário: R$3000,00
Gerente: Ana Souza, Cargo: Gerente, Salário: R$5000,00
Process finished with exit code 0.
```

Exercício 9 - Controle de Estoque via Linha de Comando

A screenshot of the Visual Studio IDE interface. The solution explorer shows multiple projects under 'Exercicio01'. The code editor displays 'Produto.cs' with the following content:

```
1 class Produto
2 {
3     public string Nome { get; set; }
4
5     public int Quantidade { get; set; }
6
7     public decimal Preco { get; set; }
8
9     public Produto(string nome, int quantidade, decimal preco)
10    {
11        Nome = nome;
12        Quantidade = quantidade;
13        Preco = preco;
14    }
15 }
```

The output window shows the results of the program execution:

```
/home/kaike/RiderProjects/Exercicio01/Exercicio09/bin/Debug/net9.0/Exercicio08
Funcionario: Carlos Silva, Cargo: Analista, Salário: R$3000,00
Gerente: Ana Souza, Cargo: Gerente, Salário: R$5000,00
Process finished with exit code 0.
```

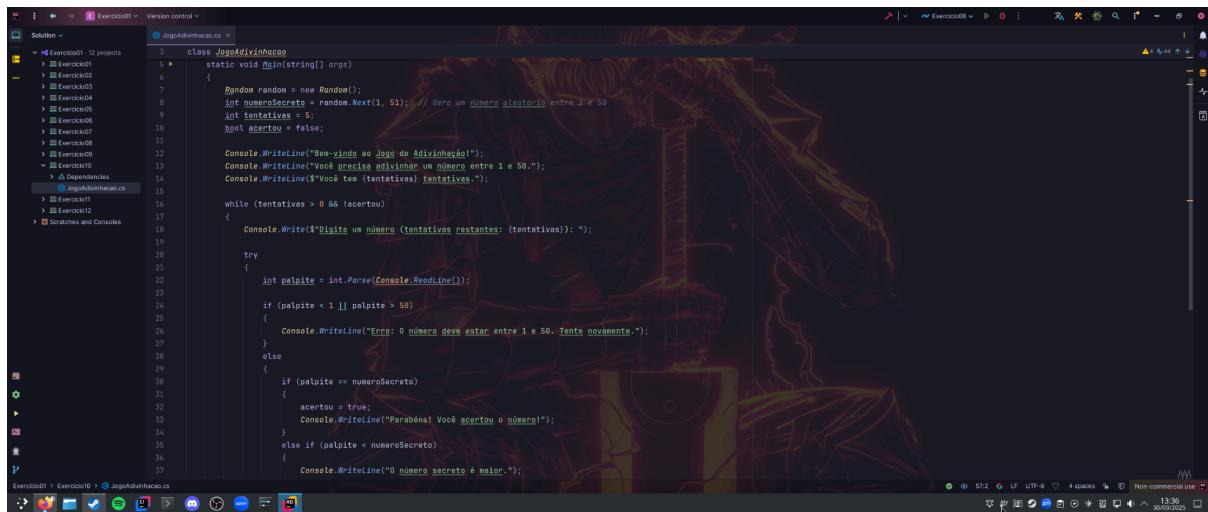
A screenshot of the Visual Studio IDE interface. The solution explorer shows multiple projects under 'Exercicio01'. The code editor displays 'Program.cs' with the following content:

```
1 class Program
2 {
3     static void Main(string[] args)
4     {
5         while (true)
6         {
7             if (opcao == "1")
8                 Console.ReadKey();
9             else if (opcao == "2")
10                Console.ReadKey();
11             else if (opcao == "3")
12                {
13                    break;
14                }
15             else
16                 {
17                     Console.WriteLine("Opção inválida. Tente novamente.");
18                     Console.ReadKey();
19                 }
20         }
21     }
22 }
```

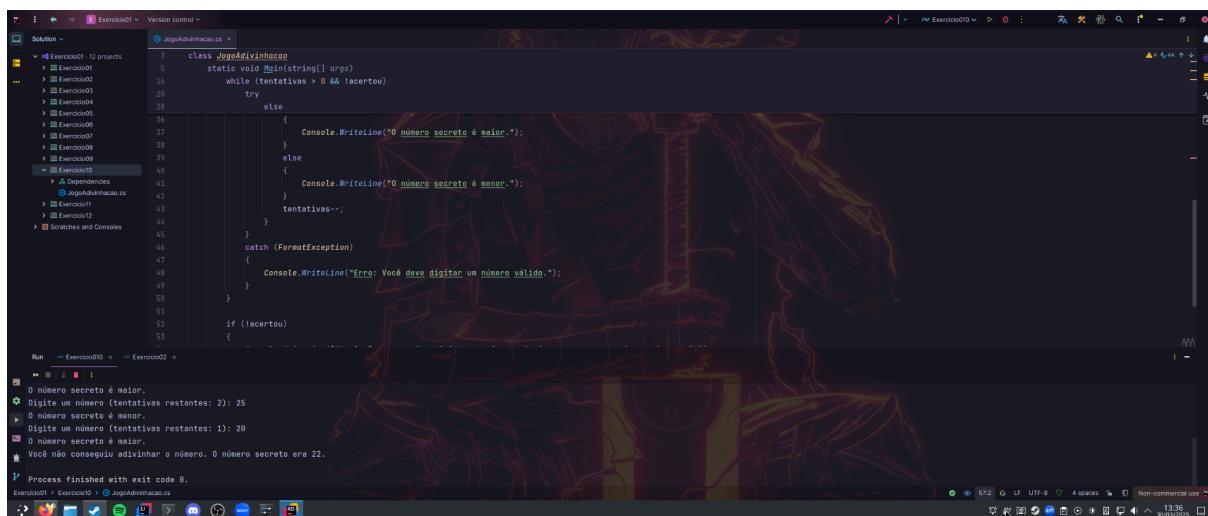
The output window shows the results of the program execution:

```
/home/kaike/RiderProjects/Exercicio01/Exercicio08/bin/Debug/net9.0/Exercicio08
Funcionario: Carlos Silva, Cargo: Analista, Salário: R$3000,00
Gerente: Ana Souza, Cargo: Gerente, Salário: R$5000,00
Process finished with exit code 0.
```

Exercício 10 - Jogo de Adivinhação

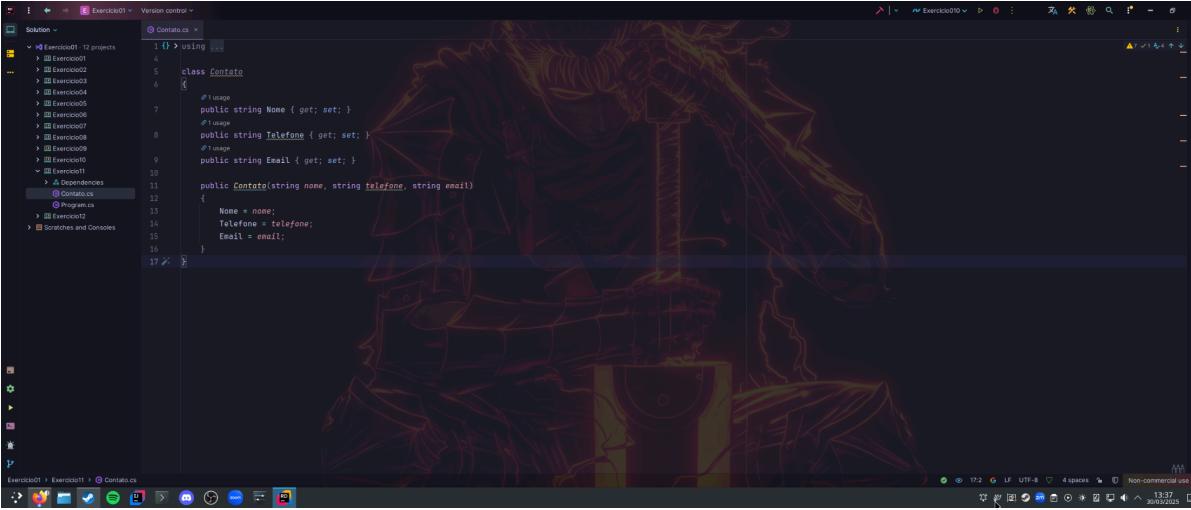


```
3 class JogoAdivinhação
4 {
5     static void Main(string[] args)
6     {
7         Random random = new Random();
8         int numeroSecreto = random.Next(1, 51); // Gera um número aleatório entre 1 e 50
9         int tentativas = 5;
10        bool acertou = false;
11
12        Console.WriteLine("Bem-vindo ao Jogo de Adivinhação!");
13        Console.WriteLine("Você precisa adivinhar um número entre 1 e 50.");
14        Console.WriteLine($"Você tem {tentativas} tentativas.");
15
16        while (tentativas > 0 && !acertou)
17        {
18            Console.Write($"Digite um número (tentativas restantes: {tentativas}): ");
19
20            try
21            {
22                int palpito = int.Parse(Console.ReadLine());
23
24                if (palpito < 1 || palpito > 50)
25                {
26                    Console.WriteLine("Erro: O número deve estar entre 1 e 50. Tente novamente.");
27                }
28                else
29                {
30
31                    if (palpito == numeroSecreto)
32                    {
33                        acertou = true;
34                        Console.WriteLine("Parabéns! Você acertou o número!");
35                    }
36                    else if (palpito < numeroSecreto)
37                    {
38                        Console.WriteLine("O número secreto é maior.");
39                    }
40                    else
41                    {
42                        Console.WriteLine("O número secreto é menor.");
43                    }
44
45                }
46            catch (FormatException)
47            {
48                Console.WriteLine("Erro: Você deve digitar um número válido.");
49            }
50
51            if (!acertou)
52            {
53
54            }
55        }
56    }
57}
```

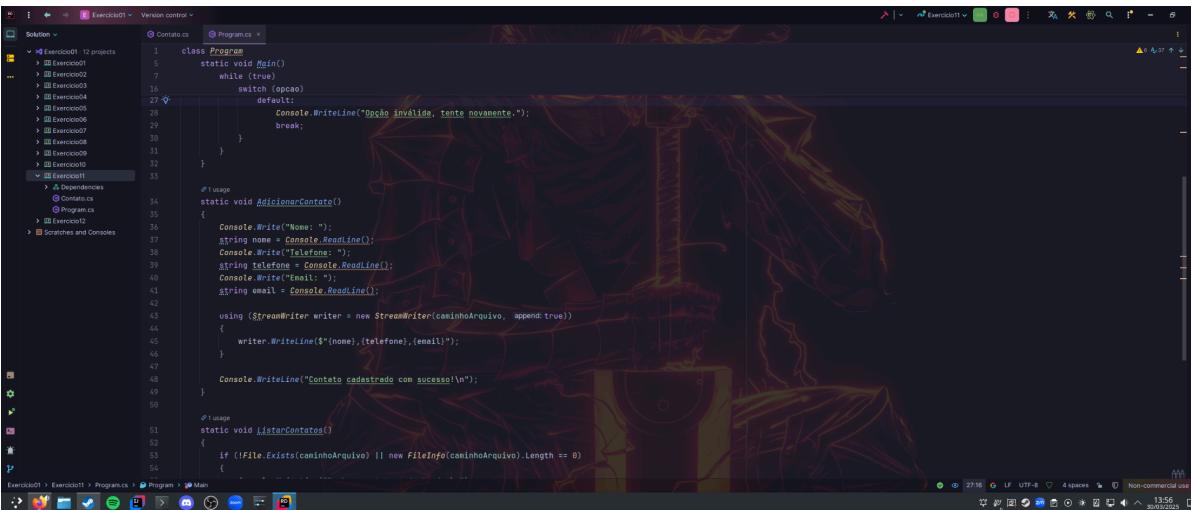


```
3 class JogoAdivinhação
4 {
5     static void Main(string[] args)
6     {
7         while (tentativas > 0 && !acertou)
8         {
9             try
10             {
11                 Console.WriteLine("Digite um número (tentativas restantes: 2): 25");
12
13                 if (palpito == numeroSecreto)
14                 {
15                     acertou = true;
16                     Console.WriteLine("Parabéns! Você acertou o número!");
17                 }
18                 else if (palpito < numeroSecreto)
19                 {
20                     Console.WriteLine("O número secreto é menor.");
21                 }
22                 else
23                 {
24                     Console.WriteLine("O número secreto é maior.");
25                 }
26
27                 tentativas--;
28             }
29             catch (FormatException)
30             {
31                 Console.WriteLine("Erro: Você deve digitar um número válido.");
32             }
33
34             if (!acertou)
35             {
36
37             }
38         }
39     }
40 }
```

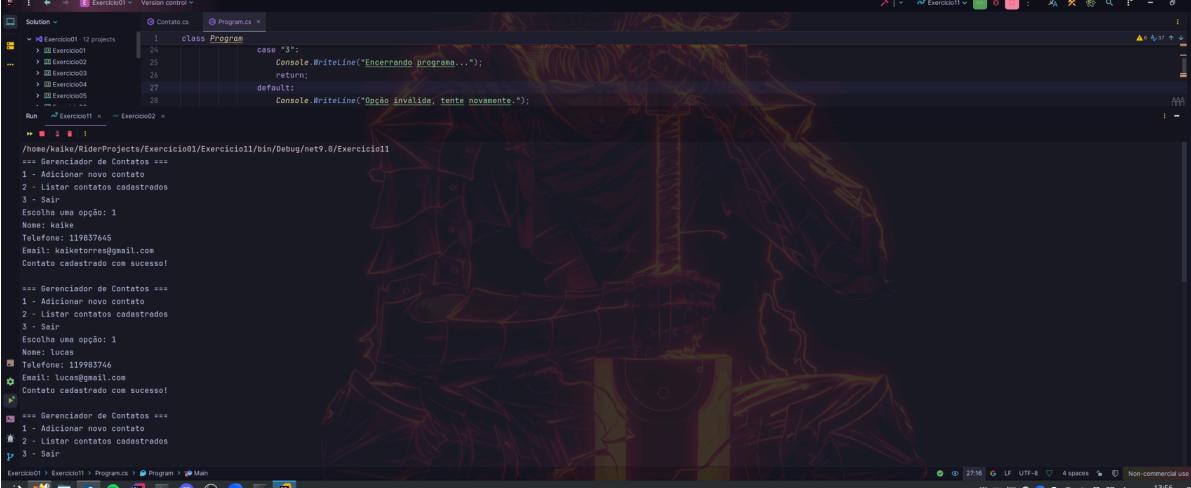
Exercício 11 - Manipulação de Arquivos - Cadastro e Listagem de Contatos



```
Contato.cs
1  using ...
2
3  class Contato
4
5      public string Nome { get; set; }
6
7      public string Telefone { get; set; }
8
9      public string Email { get; set; }
10
11     public Contato(string nome, string telefone, string email)
12     {
13         Nome = nome;
14         Telefone = telefone;
15         Email = email;
16     }
17 }
```



```
Program.cs
1  class Program
2
3      static void Main()
4      {
5          while (true)
6          {
7              switch (opcao)
8              {
9                  case "1":
10                      AdicionarContato();
11                      break;
12
13                  case "2":
14                      ListerContatos();
15                      break;
16
17                  default:
18                      Console.WriteLine("Opção inválida, tente novamente.");
19                      break;
20
21              }
22          }
23      }
24
25      static void AdicionarContato()
26      {
27          Console.Write("Nome: ");
28          string nome = Console.ReadLine();
29          Console.Write("Telefone: ");
30          string telefone = Console.ReadLine();
31          Console.Write("Email: ");
32          string email = Console.ReadLine();
33
34          using (StreamWriter writer = new StreamWriter(caminhoArquivo, append: true))
35          {
36              writer.WriteLine($"{nome},{telefone},{email}");
37          }
38
39          Console.WriteLine("Contato cadastrado com sucesso!\n");
40
41      }
42
43      static void ListerContatos()
44      {
45          if (!File.Exists(caminhoArquivo) || new FileInfo(caminhoArquivo).Length == 0)
46          {
47              Console.WriteLine("Arquivo vazio!");
48          }
49
50          else
51          {
52              string[] linhas = File.ReadAllLines(caminhoArquivo);
53
54              foreach (string linha in linhas)
55              {
56                  string[] dados = linha.Split(',');
57
58                  Console.WriteLine($"Nome: {dados[0]}, Telefone: {dados[1]}, Email: {dados[2]}");
59              }
60          }
61      }
62
63  }
```



```
Contato.cs
1  class Program
2
3      static void Main()
4      {
5          while (true)
6          {
7              switch (opcao)
8              {
9                  case "1":
10                      AdicionarContato();
11                      break;
12
13                  case "2":
14                      ListerContatos();
15                      break;
16
17                  default:
18                      Console.WriteLine("Opção inválida, tente novamente.");
19                      break;
20
21              }
22          }
23      }
24
25      static void AdicionarContato()
26      {
27          Console.Write("Nome: ");
28          string nome = Console.ReadLine();
29          Console.Write("Telefone: ");
30          string telefone = Console.ReadLine();
31          Console.Write("Email: ");
32          string email = Console.ReadLine();
33
34          using (StreamWriter writer = new StreamWriter(caminhoArquivo, append: true))
35          {
36              writer.WriteLine($"{nome},{telefone},{email}");
37          }
38
39          Console.WriteLine("Contato cadastrado com sucesso!\n");
40
41      }
42
43      static void ListerContatos()
44      {
45          if (!File.Exists(caminhoArquivo) || new FileInfo(caminhoArquivo).Length == 0)
46          {
47              Console.WriteLine("Arquivo vazio!");
48          }
49
50          else
51          {
52              string[] linhas = File.ReadAllLines(caminhoArquivo);
53
54              foreach (string linha in linhas)
55              {
56                  string[] dados = linha.Split(',');
57
58                  Console.WriteLine($"Nome: {dados[0]}, Telefone: {dados[1]}, Email: {dados[2]}");
59              }
60          }
61      }
62
63  }
```

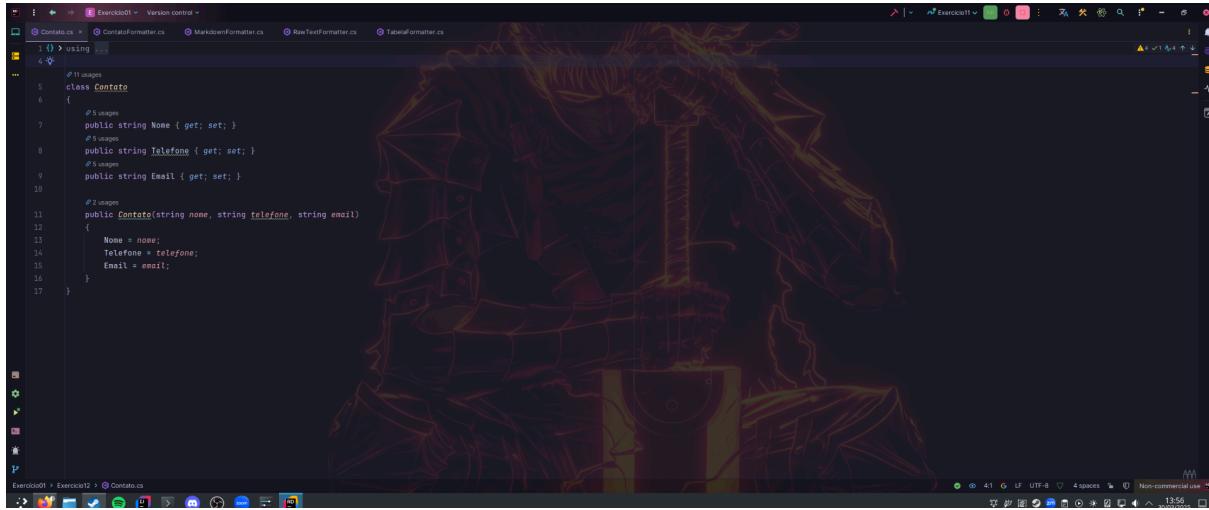
```
Exercício01> Exercicio11> Program.cs> Main> Main
/home/kakie/RiderProjects/Exercicio01/Exercicio11/bin/Debug/net9.0/Exercicio11
*** Gerenciador de Contatos ***
1 - Adicionar novo contato
2 - Listar contatos cadastrados
3 - Sair
Escolha uma opção: 1
Nome: kaku
Telefone: 119837465
Email: kaketeiros@gmail.com
Contato cadastrado com sucesso!

*** Gerenciador de Contatos ***
1 - Adicionar novo contato
2 - Listar contatos cadastrados
3 - Sair
Escolha uma opção: 1
Nome: lucas
Telefone: 119837474
Email: lucas@gmail.com
Contato cadastrado com sucesso!

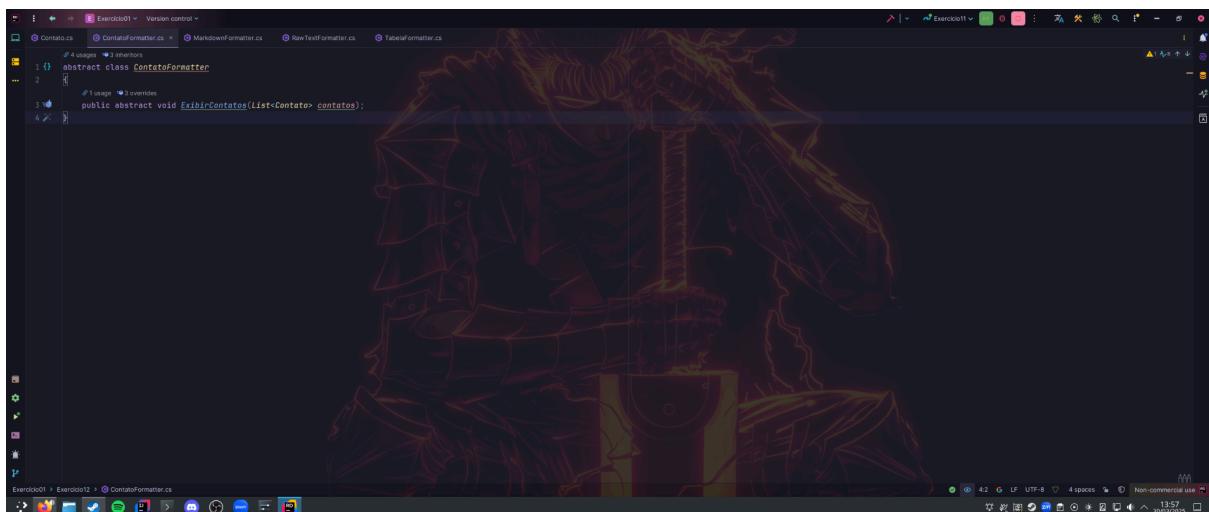
*** Gerenciador de Contatos ***
1 - Adicionar novo contato
2 - Listar contatos cadastrados
3 - Sair

```

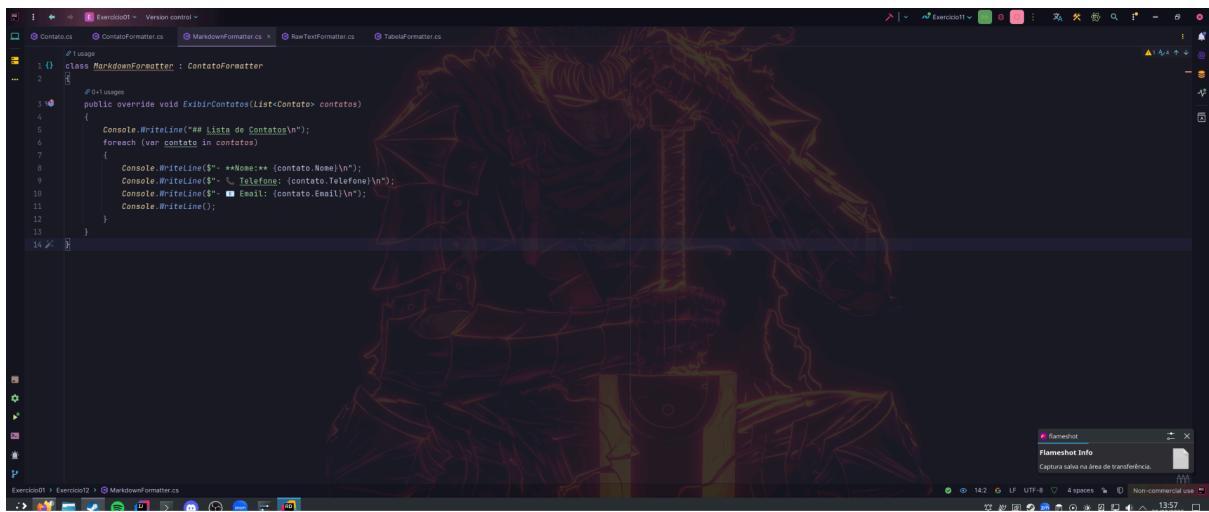
Exercício 12 - Manipulação de Arquivos com Herança e Polimorfismo - Formatos de Exibição



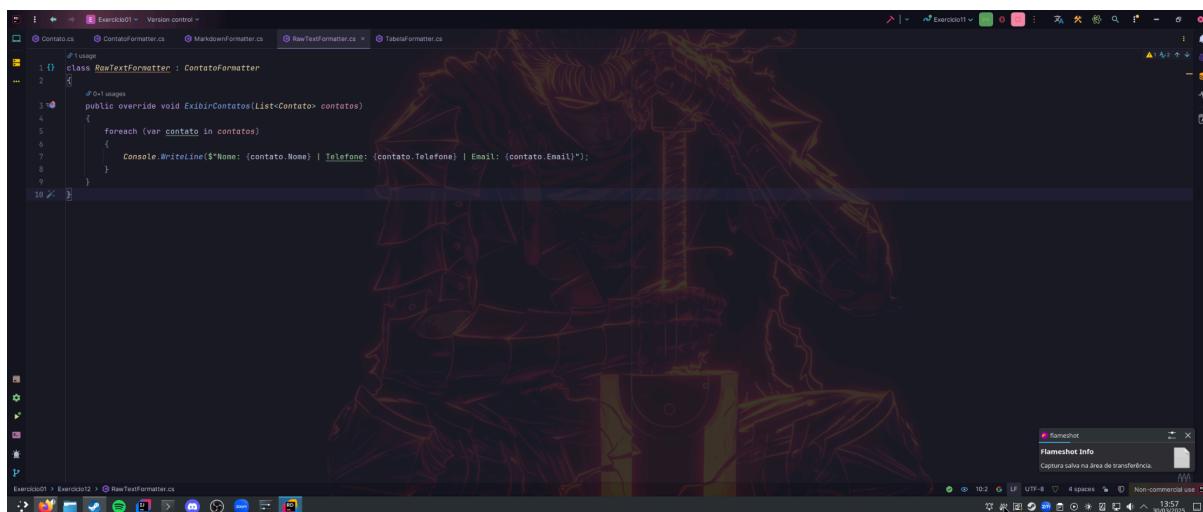
```
1 ① ② using ...
2 ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
3 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
4 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
5 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
6 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
7 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
8 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
9 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
10 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
11 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
12 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
13 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
14 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
15 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
16 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
17 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
```



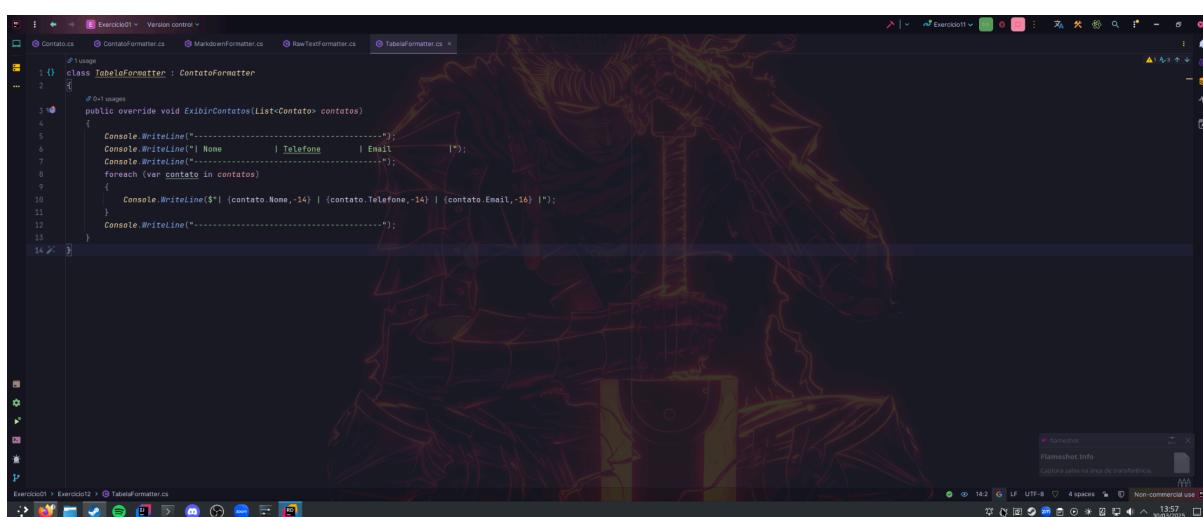
```
1 ① ② abstract class ContatoFormatter
2 ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
3 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
4 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
```



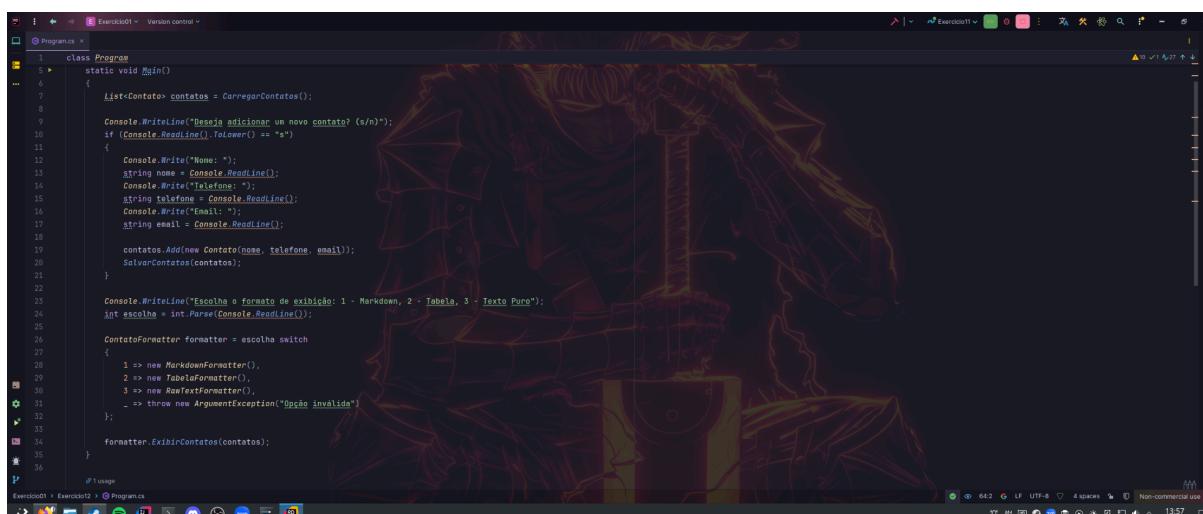
```
1 ① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
2 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
3 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
4 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
5 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
6 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
7 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
8 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
9 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
10 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
11 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
12 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
13 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
14 ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰
```



```
1 // Usage
2 class RawTextFormatter : ContatoFormatter
3 {
4     // ...
5     public override void ExhibirContatos(List<Contato> contatos)
6     {
7         foreach (var contato in contatos)
8         {
9             Console.WriteLine($"Nome: {contato.Nome} | Telefone: {contato.Telefone} | Email: {contato.Email}");
10        }
11    }
12 }
```



```
1 // Usage
2 class TableFormatter : ContatoFormatter
3 {
4     // ...
5     public override void ExhibirContatos(List<Contato> contatos)
6     {
7         Console.WriteLine("-----");
8         Console.WriteLine("Nome | Telefone | Email");
9         Console.WriteLine("-----");
10        foreach (var contato in contatos)
11        {
12            Console.WriteLine($"{contato.Nome,-14} | {contato.Telefone,-14} | {contato.Email,-16} ");
13        }
14        Console.WriteLine("-----");
15    }
16 }
```



```
1 class Program
2 {
3     static void Main()
4     {
5         List<Contato> contatos = CarregaContatos();
6
7         Console.WriteLine("Deseja adicionar um novo contato? (s/n)");
8         if (Console.ReadLine().ToLower() == "s")
9         {
10             string nome = Console.ReadLine();
11             string telefone = Console.ReadLine();
12             string email = Console.ReadLine();
13             string email = Console.ReadLine();
14
15             contato.Add(new Contato(nome, telefone, email));
16             SalvarContatos(contatos);
17         }
18
19         Console.WriteLine("Escolha o formato de exibição: 1 - Markdown, 2 - Tabela, 3 - Texto Puro");
20         int escolha = int.Parse(Console.ReadLine());
21
22         ContatoFormatter formatter = escolha switch
23         {
24             1 => new MarkdownFormatter(),
25             2 => new TableFormatter(),
26             3 => new RawTextFormatter(),
27             _ => throw new ArgumentException("Opção inválida")
28         };
29
30         formatter.ExibirContatos(contatos);
31     }
32
33     // ...
34
35     // ...
36 }
37 
```

```
Program.cs
1 class Program
2 {
3     static void Main()
4     {
5         formatter.ExibirContatos(contatos);
6     }
7
8     #usage
9     static List<Contato> CarregarContatos()
10    {
11        List<Contato> contatos = new List<Contato>();
12        if (File.Exists(caminhoArquivo))
13        {
14            foreach (var linha in File.ReadAllLines(caminhoArquivo))
15            {
16                var dadosString[] = linha.Split(';');
17                if (dados.Length == 3)
18                {
19                    contatos.Add(new Contato(nome: dados[0], telefone: dados[1], email: dados[2]));
20                }
21            }
22        }
23        return contatos;
24    }
25
26    #usage
27    static void SalvarContatos(List<Contato> contatos)
28    {
29        using (StreamWriter writer = new StreamWriter(caminhoArquivo))
30        {
31            foreach (var contato in contatos)
32            {
33                writer.WriteLine($"{contato.Nome};{contato.Telefone};{contato.Email}");
34            }
35        }
36    }
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
159
```

```
Solution - Exercicio01 - 12 projects
Exercicio01
Exercicio02
Exercicio03
Exercicio04
Exercicio05
Exercicio06
Exercicio07
Exercicio08
Exercicio09
Exercicio10
Exercicio11
Exercicio12
ExercicioFormatter.cs
Contato.cs
ContatoFormatter.cs

Run - Exercicio11 - Exercicio12

Name: keike
Telefone: 998276363
Email: keiektorres@gmail.com
Escolha o formato de exibição: 1 - Markdown, 2 - Tabela, 3 - Texto Puro
1
## Lista de Contatos

- **Name:** keike
  - Telephone: 998276343
  - Email: keiektorres@gmail.com

Process finished with exit code 0.
```

Conclusão

Ao final deste exercício, você terá aplicado o conceito de encapsulamento para proteger dados sensíveis em um sistema bancário digital, garantindo que o saldo da conta seja acessado e modificado apenas de maneira segura. Além disso, você terá implementado métodos para validar entradas, impedir transações inválidas e assegurar a integridade das operações financeiras. Esses conhecimentos são fundamentais para o desenvolvimento de aplicações seguras e estruturadas, sendo amplamente utilizados em sistemas financeiros e em ambientes de desenvolvimento profissional.