Programação e Sistemas de Informação

Módulo 09

Introdução à Programação Orientada a Objetos

Aula 02

Preparação para a semana

1) Abrir uma consola // Git Bash/PowerShell
2) cd MinhaPastaDePSI // Nome à vossa escolha
3) mkdir Semana02
4) cd Semana02
5) dotnet new sln // Cria solução
6) git add .
7) git commit -m “Adicionar solução para semana 2”
8) git push
9) Visual Studio Code → Open Folder → Semana02

Tipos de referência - String

- Sequências de caracteres Unicode
- Exemplo de uso:
  - string mensagem = “Olá Mundo!”;
  - mensagem = “Nova mensagem”;

String - Literais

- Podem conter caracteres de *escape* e símbolos Unicode
  - string s = “Um tab \t e um símbolo de copyright \u00A9”;
- Podem ficar difíceis de ler
  - string s = “C:\\Users\\nome\\Documents”;
- Solução: *strings verbatim* // Literalmente
  - string s = @“C:\Users\nome\Documents”;
  - string s = @“Não é um tab \t e não é um símbolo de copyright \u00A9”;
- Aspas em *strings* normais e *verbatim*
  - string s = “\“Esta string está entre aspas\””;
  - string s = @“““Esta string está entre aspas”””;

Exercício 1

- ➔ Criar projeto **VariasStrings** na solução **Semana02**
  - Adicionar algumas variáveis do tipo *string*
    - Algumas normais, outras precedidas de @
    - Testar vários caracteres de *escape* e Unicode
  - Imprimir no ecrã o valor de cada variável
  - Git: *add* alterações + *commit* com mensagem apropriada
  - Fazer *push* de todos os *commits* para o repositório remoto

String - Concatenação e interpolação

- Concatenação com o operador **+**:
  - string s = “Uma ” + “string ” + “concatenada”;
  - s += “ mais texto”;
- Interpolação com $:
  - string s = $“{x} mais {y} é igual a {x + y}”;
- Interpolação em strings *verbatim*:
  - string s = $@“Uma string verbatim e interpolada {x}”;

Formatação explícita de *strings*

- Método **String.Format()**
  - string s = String.Format(“Arg {0} e {1}”, x, y);
  - string s = String.Format(“Arg {1} e {0}”, 5, 10);
  - string s = String.Format(“Olá {0}, No. {1}”, “mundo”, 3);
- Método **Console.WriteLine()** também suporta esta abordagem
  - Console.WriteLine(“Valor de x é {0}”, x);

Exercício 2

- ➔ No projeto **VariasStrings:**
  - Adicionar mais variáveis do tipo string
    - Algumas concatenadas:
      - “a” + 2;
      - “b” + x;
    - Algumas interpoladas:
      - $“x = {x}”;
      - $“{x} + {y} = {x + y}”;
    - Outras criadas com **String.Format()**
      - String.Format(“Y = {1}, X = {0}”, x, y);
    - Outras criadas com **Console.WriteLine()**
      - Console.WriteLine(@“Verbatim com x = {0}”, x);
  - Imprimir no ecrã o valor de cada variável
  - Git: *add* alterações, *commit* com mensagem apropriada, *push* de todos os commits para o repositório remoto

Formatação de *strings* em C#

- **{variável/índice,alinhamento:formato}**
- Exemplos:
  - double x = 1.234;
  - string s = $“x={x:f1}
    - Resultado: x=1.2
  - Console.WriteLine(“.{0,4:x}. e {1,-4:x}.”, 10, 11);
    - Resultado: .   a. e .b   .”
  - string s = String.Format(“‘{1,-6:f2}’ e ‘{0:p1}’”, 0.2, 5);
    - Resultado: ‘5.00 ’ e ‘20.0%’

Formatação de *strings* em C#

- Alinhamento
  - Inteiro positivo → espaços à esquerda
  - Inteiro negativo → espaços à direita
- Formatos // Podem ter um inteiro a indicar a precisão/casas decimais
  - c ou C → Moeda
  - d ou D → Inteiros
  - f ou F → Reais
  - p ou P → Percentagem
  - x ou X → Hexadecimal

Exercício 3

- No projeto **VariasStrings**
  - Adicionar as seguintes variáveis no início do Main():
    - **double** dd = 0.12345;
    - **int** ii = 18;
  - Imprimir **dd** com a seguinte formatação:
    - Número real com duas casas decimais
    - Percentagem com uma casa decimal
  - Imprimir **ii** com a seguinte formatação:
    - Hexadecimal
    - Moeda
  - Executar os comandos necessários para enviar as alterações para o repositório remoto

Operadores típicos

Tipo	Operadores
Aritméticos	+, -, *, /, %
Incremento / Decremento	++, --
Atribuição	=, +=, -=, +=, /=, %=
Relacionais	==, >, <, !=, >=, <=
Condicionais	&&, ||, !
Bit a bit	&, |, ^, ~, <<, >>
Outros	sizeof(), ?:, =>

Incremento e decremento

- Operador de incremento: **++** // +1
- Operador de decremento: **--** // -1
- Uso como sufixo
  - int a = 2, b = -5, c;
  - c = a++ + b++; // a = 3, b = 4, c = -3
- Uso como prefixo
  - int a = 2, b = 5, c;
  - c = ++a + ++b; // a = 3, b = 4, c = -1

Exercício 4

1) Criar projeto **IncDec**, solução **Semana02**
2) Mostrar no ecrã resultado de operação em que **++** seja usado como **sufixo**
3) *Commit* das alterações
4) Mostrar no ecrã resultado de operação em que **--** seja usado como **prefixo**
5) *Commit* das alterações
6) Fazer *push* para o repositório remoto

Operadores bit-a-bit

- Operadores lógicos

AND &

&	0	1
0	0	0
1	0	1

OR |

|	0	1
0	0	1
1	1	1

NOT ~

x	~x
0	1
1	0

XOR ^

^	0	1
0	0	1
1	1	0

- Operadores de deslocamento
  - Shift Right >>
  - Shift Left <<

Exercício 5

```
1  using System;
2
3  class Program
4  {
5      static void Main(string[] args)
6      {
7          int i1 = 7;
8          int i2 = 5;
9          int i3 = -9;
10         int x = 2;
11         // Instrução
12         Console.WriteLine(x);
13     }
14 }
```

- Indicar o que aparece no ecrã quando a linha 11 é substituída pelas instruções numeradas

1) x = i1 - i3;
2) x = i3 / i1;
3) x += i2;
4) x *= i1
5) x %= i2 - 0b1001;
6) x = i1 << 2;
7) x <<= x;
8) x &= 0x000A ^ i1;
9) x = ~(i1 | i2)

Operadores bit-a-bit

- Operadores lógicos condicionais

AND &&

&&	F	T
F	F	F
T	F	T

OR ||

||	F	T
F	F	T
T	T	T

NOT !

x	!x
F	T
T	F

XOR ^

^	F	T
F	F	T
T	T	F

  - T = true
  - F = false

Operadores bit-a-bit

- Operadores relacionais
  - Produzem sempre booleanos
- Exemplos:
  - bool a = 5 > 3;
  - bool b = 5 < 3;
  - bool c = 5 >= 3;
  - bool d = 5 == 3;
  - bool e = 5 != 3;
- Normalmente usados diretamente em condições:
  - if (idade >= 18)...
  - for (int i = 0; i < 10; i++)...

Exercício 6

```
1  using System;
2
3  class Program
4  {
5      static void Main(string[] args)
6      {
7          float f1 = 1.4f;
8          float f2 = -13.7f;
9          bool b1 = true;
10         bool b2 = false;
11         bool x = true;
12         // Instrução
13         Console.WriteLine(x);
14     }
15 }
```

- Indicar o que aparece no ecrã quando a linha 12 é substituída pelas instruções numeradas

1) x = true && false;
2) x = !true || false;
3) x = true ^ b2;
4) x &= f2 < f1;
5) x ^= !(f2 != f1);
6) x |= true ^ b1;
7) x = b1 && b2 && !(f1 >= f2);
8) x &= b1 ^ b2 ^ (f1 == f2);
9) x ^= !(b1 || b2);

Input do utilizador e conversão em outros tipos

- Ler *string* inserida pelo utilizador
  - `string str = Console.ReadLine();`
- Conversão de *string* noutros tipos
  - `int i = int.Parse(str);`
  - `float f = float.Parse(str);`
  - `double d = double.Parse(str);`
  - …
- Conversão de um tipo para outro tipo específico
  - `int i = Convert.ToInt32(3.45f);` // float para int
  - `int j = Convert.ToInt32(false);` // bool para int
  - `short s = Convert.ToInt16(“3”);` // string para short
  - `float f = Convert.ToSingle(9.9)` // double para float
  - …

Exercício 7

- ➔ Criar projeto **Cilindro** na solução **Semana02**
- ➔ Programa deve:
  - ◆ Pedir ao utilizador a altura (**a**) e o raio (**r**) de um cilindro
  - ◆ Apresentar o volume (**V**) e a área de superfície ($\mathbf{S_a}$) desse cilindro
    - $V = \pi r^2 a$
    - $S_a = 2 \pi r (r + a)$
    - $\pi = 3.1415926$
- ➔ Fazer vários *commits* durante o exercício
- ➔ Fazer *push* de todos os *commits* para o repositório remoto