Programação e Sistemas de Informação

Módulo 09

Introdução à Programação Orientada a Objetos

Aula 01

Linguagens: do baixo ao alto nível

- C - Controlo quase total sobre o *hardware*. Necessário especificar os detalhes do que se pretende fazer. Gestão manual da memória.
- C++ - Faz o que o C faz, com várias abstrações que escondem a complexidade do *hardware*. Gestão manual da memória.
- Java, C# - Baseadas nas abstrações do C++, mas sem opção de controlar tudo como no C. Gestão automática da memória.
- Python, Ruby, PHP, JavaScript - Muito alto nível, escondem a maioria dos detalhes relacionados com o *hardware*.

Linguagem C#

- Faz parte do .NET
  - .NET também suporta F# e Visual Basic
- Vamos usar o C# 8.0

Estrutura de uma solução .NET

- Solução **MySolution**
  - Ficheiro **MySolution.sln**
  - Projetos // Um ou mais
    - Projeto1
      - Ficheiro Projeto1.csproj
      - Um ou mais ficheiros com a extensão .cs
    - Projeto2
      - Ficheiro Projeto2.csproj
      - Um ou mais ficheiros com a extensão .cs
    - …

Criar solução e projeto

- Na consola (Git Bash / PowerShell)
  - cd MinhaPastaDePSI // Nome à vossa escolha
  - mkdir Semana01
  - cd Semana01
  - dotnet new sln // Cria solução
  - dotnet new console -n OlaMundo // Cria projeto de consola chamado “OlaMundo”
  - dotnet sln add OlaMundo // Adiciona projeto à solução
  - cd .. // Volta à pasta raiz
  - dotnet new gitignore // Cria ficheiro .gitignore apropriado
  - git init
  - git add . // Prepara todos os ficheiros para serem adicionados ao repositório remoto
  - git commit -m “First commit”
  - cd Semana01
  - dotnet run -p OlaMundo // Executa projeto

Adicionar repositório remoto

- Criar repositório para as aulas de PSI no GitHub
- git remote add origin https://github…
- git branch -M main
- git push -u origin main // Primeiro push requer -u
- Verificar se a solução aparece no GitHub

Primeiro programa em C# - *Keywords*

```
using System;

namespace OlaMundo
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello, World!");
        }
    }
}
```

Argumentos da linha de comandos

- ➔ dotnet run -p OlaMundo -- arg1 arg2 arg3…
  - ◆ Argumentos colocados após --

Exercício 1

- Criar novo projeto com nome **Argumentos** na solução **Semana01**
- Escrever programa que mostra no ecrã argumentos passados na linha de comandos
- Notas:
  - A variável args é um array de strings
  - Ciclo **for** é igual em C++ e C#
- Criar novo *commit* com as alterações feitas
- Fazer *push* das alterações locais
- Verificar que *commits* locais aparecem no repositório remoto

Indentação

- Visual Studio Code e C# impõem um estilo próprio
  - Chaveta de abertura na linha seguinte
  - Indentação com 4 espaços
- Visual Studio indenta tudo automaticamente
  - Tecla F1 > Format Document

Comentários

- Tipos de comentários:
  - // Apenas uma linha
  - /* Múltiplas linhas */
- Boas práticas
  - Um comentário por cada secção de código
  - Comentários indentados com o código
- Comentários de documentação
  - Formato XML
  - Permitem produzir documentação em formato HTML

Exercício 2

- No projeto **OlaMundo** escrever /// antes do Main e da classe Program
  - Preencher campos entre as *tags*
- Criar novo *commit* com as alterações feitas
- Fazer *push* das alterações locais

Variáveis

- Têm nome, tipo e valor
  - Declarar uma variável
    - int numero;
  - Atribuir um valor à variável
    - numero = 3;
  - Declarar e atribuir valor na mesma linha
    - int numero = 3;
  - Mostrar valor da variável no ecrã
    - Console.WriteLine(numero);

Tipos simples - Inteiros

- **Signed**

Tipo C#	Tipo sistema	Bytes (bits)	Alcance
sbyte	SByte	1 (8)	-128 a 127
short	Int16	2 (16)	-32768 a 32767
int	Int32	4 (32)	-2^{31} a $2^{31} - 1$
long	Int64	8 (64)	-2^{63} a $2^{63} - 1$

Tipos simples - Inteiros

- **Unsigned**

Tipo C#	Tipo sistema	Bytes (bits)	Alcance
byte	Byte	1 (8)	0 a 255
ushort	UInt16	2 (16)	0 a 65535
uint	UInt32	4 (32)	0 a 2^{32}
ulong	UInt64	8 (64)	0 a 2^{64}

Tipos simples - Reais

Tipo C#	Tipo sistema	Bytes (bits)	Precisão
float	Single	4 (32)	7 dígitos
double	Double	8 (64)	15-16 dígitos
decimal	Decimal	16 (128)	28-29 dígitos

Literais de tipos simples

- Por omissão números reais são considerados **double**
  - double pi = 3.1415;
- Literais **float** requerem um **f** ou **F** no fim
  - float pi = 3.1415f;
- Literais **decimal** requerem um **m** ou **M** no fim
  - decimal pi = 3.1415m;
- Por omissão números inteiros são considerados **int**
  - int i = 3;
- Um **U** no fim indica que literal é um **uint**
  - uint i = 3U;
- Um **L** no fim indica que literal é um **long**
  - long i = 3L;
- **UL** indica que o literal é **ulong**
  - ulong i = 3UL;

Literais de caracteres

- Caracteres de sequência de *escape*

Carácter	Descrição	Valor Unicode
\’	Plica	0x0027
\”	Aspa	0x0022
\\	Barra invertida	0x005C
\0	Null	0x0000
\b	Backspace	0x0008
\n	Nova linha	0x000A
\t	Tab	0x0009

Literais de caracteres

- Especificar valor Unicode com sequência de *escape* \u (ou \x):
  - char copyrightSymbol = ‘\u00A9’;
  - char newLine = ‘\u000A’
- Tabela Unicode
- Caracteres Unicode mais complexos requerem o seguinte:
  - using System.Text; // No início do programa
  - Console.OutputEncoding = Encoding.UTF8; // No início do *Main*

Tipos simples - Booleano

- Tipo **bool**
- Dois valores literais possíveis
  - true
  - false
- Condições requerem sempre um booleano
  - bool condition = true;
  - if (condition) { … }

Exercício 3

- ➔ Criar novo projeto com nome **TiposVarios** na solução **Semana01**
  - ◆ Criar variáveis **inteiras** de diferentes tipos inicializadas com literais adequados
    - Imprimir no ecrã o valor das variáveis criadas
      - ○ Adicionar alterações ao Git e fazer *commit*
  - ◆ Criar variáveis **reais** de diferentes tipos inicializadas com literais adequados
    - Imprimir no ecrã o valor das variáveis criadas
      - ○ Adicionar alterações ao Git e fazer *commit*
  - ◆ Criar variáveis **char** com diferentes valores Unicode
    - Imprimir no ecrã o valor dos diferentes caracteres
      - ○ Adicionar alterações ao Git e fazer *commit*
  - ◆ Criar duas variáveis **bool**, uma inicializada a **true** e uma a **false**
    - Imprimir no ecrã o valor de cada variável booleana
      - ○ Adicionar alterações ao Git e fazer *commit*
- ➔ Fazer *push* de todos os *commits* para o repositório remoto

Tipos de referência - String

- Sequências de caracteres Unicode
- Exemplo de uso:
  - string mensagem = “Olá Mundo!”;
  - message = “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 4

- ➔ Criar projeto **VariasStrings** na solução **LP1Aula01**
  - 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 5

- ➔ 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 6

- 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