Programação e Sistemas de Informação

Módulo 14

Linguagem de Manipulação de Dados

Aula 14

Preparação para a semana

1) Abrir uma consola // Git Bash/PowerShell
2) cd MinhaPastaDePSI // Nome à vossa escolha
3) mkdir Semana14
4) cd Semana14
5) dotnet new sln // Cria solução
6) git add .
7) git commit -m “Adicionar solução para semana 14”
8) git push
9) Visual Studio Code → Open Folder → Semana14

Cábula para comandos dotnet

- dotnet new console -n NomeDoProjeto
  - Cria novo projeto chamado “NomeDoProjeto”
- dotnet sln add NomeDoProjeto
  - Adiciona projeto “NomeDoProjeto” à solução existente
- dotnet run -p NomeDoProjeto
  - Compila e executa projeto “NomeDoProjeto”

Cábula para projetos do dotnet 6.0

- dotnet new -i Classic.Console.Templates
  - Instala *templates* para versões anteriores do dotnet
  - Apenas é preciso correr este comando uma vez por sistema
- dotnet new console-classic --nrt=false --langVersion 8.0 -n NomeDoProjeto
  - Cria projeto com template de *framework* anterior à versão 6.0

Conteúdos

➔ Expressões de pesquisa e LINQ

LINQ - Language Integrated Query

- Componente que adiciona funcionalidades de consulta em .NET
- Permite manipular objetos que implementem IEnumerable<T>
- Tem como unidades básicas:
  - **Sequências** - quaisquer objetos que implementem IEnumerable<T>
  - **Elementos** - itens em sequências
- Operadores de *query* são métodos que transformam sequências
  - Aceitam sequência de entrada, devolvem sequência de saída
  - Classe **Enumerable** do *namespace* **System.LINQ** contém cerca de 40 operadores de *query*

LINQ - Algumas *queries* com o operador Where

- Exemplo: *array* **string[] animais = { “Cão”, “Gato”, “Abelha” };**
  - Pode-se usar o operador **Where** para obter nomes com mais de 3 caracteres
  - **IEnumerable<string> animaisFiltrados = animais.Where (nome => nome.Length > 3);**
    - Variável iterável **animaisFiltrados** fica com os valores **“Gato”** e **“Abelha”**
- Outro exemplo: classe **Videojogo** com as propriedades **Nome** e **Rating**
  - Cria-se a variável **jogos**, do tipo **List<Videojogo>**
  - Usa-se o operador **Where** para obter jogos com *rating* acima de 80
  - **IEnumerable<Videojogo> jogosFiltrados = jogos.Where(jogo => jogo.Rating > 80);**
    - Variável iterável **jogosFiltrados** contém apenas jogos com *rating* acima de 80

Outros operadores de *query*

- Select
  - Transforma cada elemento de uma sequência noutro tipo
- Take, Skip
  - Devolvem os primeiros ou últimos elementos de uma sequência
- First, Last, ElementAt
  - Devolvem um único elemento
- Count, Min, Max, Average
  - Devolvem um número que corresponde a uma operação efetuada sobre elementos
  - Apenas aplicáveis em tipos numéricos
- Contains, Any, All
  - Devolvem um booleano que corresponde à pergunta efetuada
- Union, Intersect, Except
  - Operadores que produzem conjuntos de elementos

LINQ - Alguns exemplos

- Nome de jogos com *rating* maior que 90

```
IEnumerable<string>> melhoresJogos =
    jogos.Where(jogo => jogo.Rating > 90).Select(jogo => jogo.Nome);
```

- Número de jogos com *rating* menor que 50

```
int jogosMaus = jogos.count(jogo => jogo.Score < 50);
```

- Booleana para verificar se existe algum jogo com *rating* acima de 95

```
bool jogoPerfeitoExiste = jogos.Any(jogo => jogo.Score > 95);
```

Exercício 1

- ➔ Projeto **LINQLeitor** na solução **Semana14**
  - Programa deve ler um ficheiro de texto para *array* de strings
  - Usando operadores de *query*, deve-se obter a seguinte informação:
    - Número de linhas lidas com mais de 30 caracteres
    - Média do nº de caracteres das linhas lidas
    - Booleana que indica se existe alguma linha com mais de 120 caracteres
    - Iterável contento a primeira palavra de linhas que contenham um determinado carácter
- ➔ Fazer vários *commits* e, no fim do exercício, *push* para o repositório remoto

LINQ - expressões de *query*

- Expressões *lambda* (=>) tornam pesquisas compactas e fáceis de realizar
- Encadeamento de operadores (sintaxe fluente) permite *queries* ao estilo SQL

```
IEnumerable<string>> melhoresJogos =
    jogos.Where(jogo => jogo.Rating > 90).Select(jogo => jogo.Nome);
```

- Esta sintaxe pode ser substituída por expressões de *query*
    - Estas tornam o código mais semelhante a SQL

```
IEnumerable<string>> melhoresJogos =
    from jogo in jogos
    where jogo.Rating > 90
    select jogo.Nome;
```

Exercício 2

- No projeto **LINQLeitor**:
  - Usar expressões de *query* onde for apropriado
- Fazer *commit* e, no fim do exercício, *push* para o repositório remoto