Ambientes Virtuais de Execução

Nullable Types

Tipos anuláveis

- Objectivo
  - Suportar tipos-valor nulos (sem valor atribuído)

- Instâncias de tipos-referência podem não ter objecto associado (valor nulo)
- Instâncias de tipos-valor têm sempre valor não nulo

- Pode ser necessário iniciar uma instância de um tipo-valor com um valor não válido (ex: campos NULL de uma base de dados)

Tipos anuláveis

- No *namespace* `System` está definido o tipo genérico `Nullable<T>`

```
public struct Nullable<T> where T : struct
```

- `Nullable<T>` tem duas propriedades:

```
HasValue : bool
Value : T
```

  - Se `HasValue` é `true`, então `Value` é um objecto válido.
  - Caso contrário, `Value` está indefinido e uma tentativa de acesso à propriedade resulta numa excepção (`InvalidOperationException`).

O tipo genérico Nullable<T>

```
public struct Nullable<T> where T: struct {
    private bool hasValue;
    internal T value;
    public Nullable(T value);
    public bool HasValue { get; }
    public T Value { get; }
    public T GetValueOrDefault();
    public T GetValueOrDefault(T defaultValue);
    public override bool Equals(object other);
    public override int GetHashCode();
    public override string ToString();
}
```

O tipo genérico Nullable<T>

```
public struct Nullable<T> where T : struct {
   private Boolean hasValue = false; // Assume null
   internal T value = default(T); // Assume todos os bits a zero
   public Nullable(T value){
       this.value = value; this.hasValue =true;
   }
   public Boolean HasValue { get { return hasValue; } }
   public T Value {
      get { if (!hasValue) {
            throw new InvalidOperationException("Nullable object
                                                must have a value.");}
           return value;
          }
    }
  public override string ToString() {
    if (!HasValue) return "";
    return value.ToString();
}
//...
}
```

Exemplos

```
Int32? x = 5;
Int32? y = null;
```

```
private static void

ConversionsAndCasting()
{
  Int32? a = 5;
   Int32? b = null;
   Int32 c = (Int32) a;
   Double? d = 5;
     // d fica 5.0
   Double? e = b;
   //e fica null
}
```

Conversão implícita de um Int32 não nulo para Nullable<Int32>

Conversão implícita de um 'null' para Nullable<Int32>

Conversão explícita de um Nullable<Int32> para um Int32 não nulo.

Casting...

Operadores em C#

- Operadores unários(+++, -, --, ! , ~)
  - operando null => resultado null
- Operadores binários (+, -, *, /, %, ^, <<, >>)
  - um dos operandos null => resultado null
- Operadores binários &, |

&	true	false	null
true	true	false	null
false	false	false	false
null	null	false	null

|	true	false	null
true	true	true	true
false	true	false	null
null	true	null	null

- Operadores de igualdade(==, !=)
  - Se ambos os operandos forem null, são iguais. Se apenas um dos operandos for null não são iguais. Se nenhum dos operandos for null, então são comparados os valores.
- Operadores relacionais<, >, <=, >=)
  - Se um dos operandos for null, o resultado é false. Se nenhum dos operandos for null, são comparados os valores

Manipulação de instâncias nullable

```
private static Int32? NullableCodeSize(Int32? a, Int32? b)
{
  return a + b;
}
```

```
private static Nullable<Int32> NullableCodeSize(Nullable<Int32> a,
                                                Nullable<Int32> b) {
  Nullable<Int32> nullable1 = a;
  Nullable<Int32> nullable2 = b;
  if (!(nullable1.HasValue & nullable2.HasValue)) {
       return new Nullable<Int32>();
   }
  return new Nullable<Int32>(nullable1.GetValueOrDefault() +
nullable2.GetValueOrDefault());
 }
```

Tipos anuláveis

- C# 2.0 admite uma notação abreviada para os tipos anuláveis
  - Modificador ? para declarar um tipo como anulável.

```
typeof(int?) == typeof(Nullable<int>)
```

  - Operador ?? para indicar o valor pré-definido numa atribuição de uma instância de um tipo anulável a um não-anulável.

```
(a ?? 0) <=> (a.HasValue ? a.Value : 0)
```

  - Comparação com `null` verifica `HasValue`.

```
(a == null) <=> !a.HasValue
```

Boxing e Unboxing de Nullable Value Types

```
//...
Int32? n = null;
Object o = n; // o é null
n = 5;
o = n; // o refere-se a um boxed Int32
//...
```

```
//...
Object o = 5; Int32? a = (Int32?) o; // a = 5  Unbox para um Nullable<Int32>
Int32 b = (Int32) o; // b = 5.  Unbox para um Int32
o = null;

a = (Int32?) o; // a = null
b = (Int32) o; // NullReferenceException
//...
```

Tipos anuláveis

```
Nullable<int> a = null;
Nullable<int> b = 3;

int? c = null;
int? d = 5;

b += d;                 // b <- 8
d = a + b;              // d <- null   (porque a vale null)

int e = (int)b;         // e <- 8
int f = (int)c;         // excepção    (porque c vale null)

int g = c ?? -1;        // g <- -1     (porque c vale null)
int h = a ?? c ?? 0;    // h <- 0 (porque a e c valem null)
```