UNIDADE 12 – OBJECT

Nessa unidade veremos resumo bem objetivo sobre o que significa a classe object.

SUMÁRIO

1 OBJECT

Todas as classes derivam direta ou indiretamente da classe Object. Consequentemente, todo conteúdo definido nessa classe estará presente em todos os objetos.

Além disso, qualquer referência pode ser armazenada em uma variável do tipo Object. Ou seja, a ideia do polimorfismo pode ser aplicada para criar métodos genéricos que podem ser aplicados em objetos de qualquer classe. Na linguagem C# podemos utilizar a palavra chave object como alias para Object.

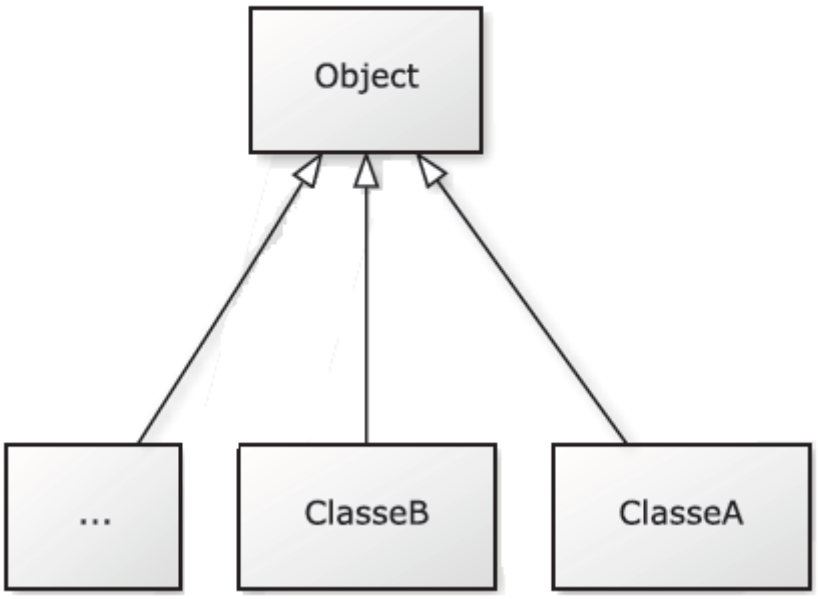

Figura 1: A classe Object

1.1 POLIMORFISMO

Aproveitando o polimorfismo gerado pela herança da classe Object, é possível criar uma classe para armazenar objetos de qualquer tipo como se fosse uma repositório de objetos.

```
1  class Repositorio
2  {
3      // codigo da classe
4  }
```

Tabela 1: Repositorio.cs

Um array de objetos pode ser utilizado como estrutura básica para manter os objetos da repositório.

```
1  class Repositorio
2  {
3      // object : alias para System . Object
4      private object[] objetos = new object[100];
5
6  }
```

Tabela 2: Repositorio.cs

Alguns métodos podem ser criados para formar a interface do repositório. Por exemplo, métodos para adicionar, retirar e pesquisar elementos.

```
1  class Repositorio
2  {
3      private object[] objetos = new object[100];
4
5
6      public void Adiciona(object o)
7      {
8          // implementacao
9      }
10
11     public void Remove(object o)
12     {
13         // implementacao
14     }
15
16     public object Pega(int posicao)
17     {
18         // implementacao
19     }
20 }
```

Tabela 3: Repositorio.cs

Com esses métodos o repositório teria a vantagem de armazenar objetos de qualquer tipo. Porém, na compilação, não haveria garantia sobre os tipos específicos. Em outras palavras, já que objetos de qualquer tipo podem ser armazenados no repositório então objetos de qualquer tipo podem sair dele.

```
1  Repositorio repositorio = new Repositorio();
2  repositorio.Adiciona("Rafael ");
3  object o = repositorio.Pega(0);
```

Tabela 4: Utilizando o repositório

Por outro lado, na maioria dos casos, os programadores criam repositórios para armazenar objetos de um determinado tipo. Por exemplo, um repositório para armazenar somente nomes de pessoas, ou seja, para armazenar objetos do tipo String. Nesse caso, em tempo de compilação é possível "forçar" o compilador a tratar os objetos como string aplicando casting de referência.

```
1  Repositorio repositorio = new Repositorio();
2  repositorio.Adiciona(" Rafael ");
3  object o = repositorio.Pega(0);
4  string s = (string) o;
```

Tabela 5: Casting de referência

1.2 O MÉTODO TOSTRING()

Às vezes, é necessário trabalhar com uma descrição textual de determinados objetos. Por exemplo, suponha a seguinte classe:

```
1  class Conta
2  {
3      public int Numero { get; set; }
4      public double Saldo { get; set; }
5  }
```

Tabela 6: Conta.cs

Queremos gerar um documento no qual deve constar as informações de algumas contas. Podemos implementar um método, na classe Conta, que gere uma descrição textual dos objetos dessa classe.

```
1  class Conta
2  {
3      public int Numero { get; set; }
4      public double Saldo { get; set; }
5
6      public string GeraDescricao()
7      {
8          return " Conta número : " + this.Numero + " possui saldo igual a " + this.Saldo;
9      }
10 }
```

Tabela 7: Conta.cs

A utilização do método que gera a descrição textual das contas seria mais ou menos assim:

```
1  Conta conta = ...
2  string descricao = conta.GeraDescrica();
3  Console.WriteLine(descricao);
```

Tabela 8: Utilizando o método GeraDescricao()

Contudo, a classe Object possuiummétodo justamente com o mesmo propósito do GeraDescricao() chamado ToString(). Como todas as classes derivamdireta ou indiretamente da classe Object, todos os objetos possuem o método ToString().

A implementação padrão do método ToString() monta uma descrição genérica baseada no nome da classe mais específica dos objetos.

```
1  Conta conta = ...
2  string descricao = conta.ToString();
3  Console.WriteLine(descricao);
```

Tabela 9: Utilizando o método ToString()

No código acima, a descrição gerada pelo método ToString() definido na classe Object seria: "Conta".

Para alterar o comportamento do método ToString(), basta reescrevê-lo na classe Conta.

```
1  class Conta
2  {
3      public int Numero { get; set; }
4      public double Saldo { get; set; }
5
6      public override string ToString()
7      {
8          return " Conta número : " + this.Numero + " possui saldo igual a " + this.Saldo;
9      }
   }
```

Tabela 10: Conta.cs

A vantagem em reescrever o método ToString() ao invés de criar um outro método com o mesmo propósito é que diversas classes das bibliotecas do .NET utilizam o método ToString(). Inclusive, quando passamos uma variável não primitiva para o método WriteLine(), o ToString() é chamado internamente para definir o que deve ser impresso na tela.

```
1  Conta conta = ...
2  // o método ToString () será chamado internamente no WriteLine()
3  Console.WriteLine(conta);
```

Tabela 11: Utilizando o método ToString()

1.3 O MÉTODO EQUALS()

Para verificar se os valores armazenados em duas variáveis de algum tipo primitivo são iguais, devemos utilizar o operador "==". Esse operador também pode ser aplicado em variáveis de tipos não primitivos.

```
1  Conta c1 = ...
2  Conta c2 = ...
3
4  Console.WriteLine(c1 == c2);
```

Tabela 12: Comparando com

O operador "==", aplicado à variáveis não primitivas, verifica se as referências armazenadas nessas variáveis apontam para omesmo objeto na memória. Esse operador, por padrão, não compara o conteúdo dos objetos correspondentes às referências armazenadas nas variáveis submetidas à comparação.

Para comparar o conteúdo de objetos, podemos utilizar métodos. Podemos implementar um método de comparação na classe Conta.

```
1  class Conta
2  {
3      public int Numero { get; set; }
4      public double Saldo { get; set; }
5
6      public bool Compara(Conta outra)
7      {
8          return this.Numero == outra.Numero;
9      }
10 }
```

Tabela 13: Conta.cs

A utilização do método Compara() seria mais ou menos assim:

```
1  Conta c1 = ...
2  Conta c2 = ...
3
4  Console.WriteLine(c1.Compara(c2));
```

Tabela 14: Comparando com Compara()

Contudo, na classe Object, já existe um método com o mesmo propósito. O método ao qual nos referimos é o Equals(). A implementação padrão do método Equals() na classe Object delega a comparação ao operador "==". Dessa forma, o conteúdo dos objetos não é comparado por padrão. Podemos rescrever o método Equals() para alterar esse comportamento e passar a considerar o conteúdo dos objetos na comparação.

```
1  class Conta
2  {
3      public int Numero { get; set; }
4      public double Saldo { get; set; }
5
6      public override bool Equals(object obj)
7      {
8          Conta outra = obj as Conta;
9          return this.Numero == outra.Numero;
10     }
11 }
```

Tabela 15: Comparando com Compara()

A reescrita do método Equals() deve respeitar diversas regras definidas na documentação da plataforma .NET(http://msdn.microsoft.com/en-us/library/bsc2ak47.aspx).

1.4 Exercícios de Fixação

1) Crie um projeto no chamado **Unidade 12** e uma pasta **ExercicioFixacao**.

2) Adicione no projeto **Object** uma classe para modelar os funcionários do banco.

```
1  class Funcionario
2  {
3      public string Nome { get; set; }
4
5      public double Salario { get; set; }
6  }
```

Tabela 16: Funcionario.cs

3) Crie um objeto da classe Funcionario e imprima a referência desse objeto na tela.

```
1  class Aplicacao
2  {
3      static void Main()
4      {
5          Funcionario f = new Funcionario();
6
7          f.Nome = " Jonas Hirata ";
8          f.Salario = 3000;
9
10         Console.WriteLine(f);
11     }
12 }
```

Tabela 17: Aplicacao.cs

Execute o projeto!

4) Reescreva o método ToString() na classe Funcionario para alterar a descrição textual dos objetos que representam os funcionários.

```
1  class Funcionario
2  {
3      public string Nome { get; set; }
4
5      public double Salario { get; set; }
6
7      public override string ToString()
8      {
9          return " Funcionário : " + this.Nome + " - Salário : " + this.Salario;
10     }
11 }
```

Tabela 18: Funcionario.cs

5) Execute novamente o projeto.

6) Altere o método Main a classe Aplicacao. Crie dois objetos da classe Funcionario. Utilize o operador "==" e o método Equals() para compará-los.

```
1  class Aplicacao
2  {
3      static void Main()
4      {
5          Funcionario f1 = new Funcionario();
6
7          f1.Nome = " Jonas Hirata ";
8          f1.Salario = 3000;
9
10         Funcionario f2 = new Funcionario();
11
12         f2.Nome = " Jonas Hirata ";
13         f2.Salario = 3000;
14
15         Console.WriteLine(f1 == f2);
16         Console.WriteLine(f1.Equals(f2));
17     }
18 }
```

Tabela 19: Aplicacao.cs

7) Execute novamente o projeto.

8) Reescreva o método Equals() na classe Funcionario para alterar o critério de comparação dos funcionários.

```
1  class Funcionario
2  {
3      public string Nome { get; set; }
4
5      public double Salario { get; set; }
6
7      public override string ToString()
8      {
9          return " Funcionário : " + this.Nome + " - Salário : " + this.Salario;
10     }
11
12     public override bool Equals(object obj)
13     {
14         Funcionario outro = (Funcionario) obj;
15         return this.Nome == outro.Nome;
16     }
17 }
```

Tabela 20: Funcionario.cs

9) Execute novamente o projeto.

Bons Estudos!