Fundamentos de Desenvolvimento com C#

Aula 11: Herança

Professor: Rinaldo Ferreira Junior

E-mail: rinaldo.fjunior@prof.infnet.edu.br

- **Professor:** Rinaldo Ferreira Junior
- **Graduação:** Pós-graduado em Arquitetura de Softwares
- **Atuação:** .Net | C# | SQL | NoSQL | Engenheiro de Software
- **E-mail:** rinaldo.fjunior@prof.infnet.edu.br
- **Linkedin:** https://www.linkedin.com/in/rinaldo-ferreira-junior-787326a

Aula 11: Roteiro

- Herança
- Conversão de Tipos

Aula 11: Herança

- Herança permite a criação de classes que reutilizam, extendem ou modificam comportamentos definidos em outras classes.
- A classe que possui os elementos herdados chama-se Classe Base (Base Class).
- A classe que herda os membros de outra classe, chama-se Classe Derivada (Derived Class).
- Uma classe só pode derivar diretamente de uma única classe (single inheritance), porém, a herança é transitiva:
  - Classe C deriva de Classe B que deriva de Classe A. Classe C herda de A e B.
- Uma classe derivada é uma especialização da classe base.

Aula 11: Herança

- Classes podem impedir herança, por meio da instrução sealed.
- Classes seladas são comuns em bibliotecas de terceiros, onde não é permitida a extensão de funcionalidades.
- Classes seladas podem otimizar a performance de uma biblioteca, pois o CLR não precisa identificar e carregar métodos herdados.

```
public sealed class IOManager
{

}
```

Aula 11: Herança

- Para estabelecer uma relação de herança, o símbolo : é usado após o nome da classe derivada.

```
public class ContaBase
{
    3 references
    public int Conta { get; set; }

    2 references
    public byte Digito { get; set; }

}

3 references
public class ContaBancaria : ContaBase
{

}
```

Aula 11: Acesso à Classe Base

- A instrução base é usada para acessar membros da classe base, a partir da classe derivada.

```
public class ContaBancaria : ContaBase
{

    2 references
    public string Instituicao { get; set; } = string.Empty;

    2 references
    public string Agencia { get; set; } = string.Empty;

    0 references
    public bool Poupanca { get; set; } = false;

    0 references
    public void CheckAndDebit(double value, DateTime dataOperacao)
    {
        if (value > 0)
        {
            base.Debit(value, dataOperacao);
        }

    }

}
```

Aula 11: Construtores na Herança

Instituto Infnet

- Classes bases podem construtores, que por sua vez, podem ser invocados a partir de classes derivadas utilizando-se a instrução base.

```
public class ContaBase
{

    0 references
    public ContaBase()
    {

    }

    1 reference
    public ContaBase(double saldo)
    {
        this.Saldo = saldo;
    }

    5 references
    public double Saldo { get; private set; }

    1 reference
    public DateTime DataOperacao { get; private set; }

    3 references
    public int Conta { get; set; }

    2 references
    public byte Digito { get; set; }

    0 references
    public void Debit(double value)
    {
        this.Saldo -= value;
    }

    0 references
    public void Debit(int value)
    {
        this.Saldo -= value;
    }

    0 references
    public void Debit(double value, DateTime dataOperacao)
    {
        this.Saldo -= value;
        this.DataOperacao = dataOperacao;
    }
}
```

```
public class ContaBancaria : ContaBase
{

    1 reference
    public ContaBancaria()
    {

    }

    0 references
    public ContaBancaria(string instituicao, double saldoInicial) : base(saldoInicial)
    {
        this.Instituicao = instituicao;
    }

    3 references
    public string Instituicao { get; set; } = string.Empty;

    2 references
    public string Agencia { get; set; } = string.Empty;

    0 references
    public bool Poupanca { get; set; } = false;

}
```

Aula 11: Herança e Conversão de Tipos

Instituto Infnet

- Como a herança forma um relacionamento do tipo "é um", conversões de tipo podem ser feitas de uma classe derivada para uma classe base e vice-versa.

- Uma classe derivada sempre pode ser convertida diretamente em sua classe base, de forma implícita. Já a classe base precisa ser convertida explicitamente em uma classe derivada.

```
static void Main(string[] args)
{
    ContaBancaria conta = new()
    {
        Agencia = "18",
        Conta = Utils.CreateAccountNumber(),
        Digito = Utils.CreateAccountDigit(),
        Instituicao = "CEF"
    };

    ContaBase baseAccount = conta;
    ContaBancaria derivedAccount = (ContaBancaria)baseAccount;

    Console.WriteLine($"Banco: {derivedAccount.Instituicao}");
    Console.WriteLine($"Agencia: {derivedAccount.Agencia}");
    Console.WriteLine($"Conta: {baseAccount.Conta:D4} | Válida: {baseAccount.Conta.IsValidAccount(55)}");
    Console.WriteLine($"Digito: {baseAccount.Digito}");
    Console.WriteLine($"Saldo: {baseAccount.Saldo:C}");

    Console.Read();
}
```

Aula 11: Classes Abstratas

- Seu papel é definir uma definição comum que deve ser seguida por outras classes derivadas.
  - Não podem ser instanciadas diretamente.
  - Seus membros devem ser implementados nas classes derivadas

```
public abstract class Auditing
{
    2 references
    public abstract void RegisterEntry(string entry);
}
```

```
public class ContaBase : Auditing
{

    0 references
    public ContaBase()
    {

    }

    1 reference
    public ContaBase(double saldo)
    {

    }

    2 references
    public override void RegisterEntry(string entry)
    {
        throw new NotImplementedException();
    }
}
```