1 – Herança em Orientação a Objetos

Herança é um princípio da orientação a objetos, que permite que classes compartilhem atributos e métodos. Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos.

Como exemplo pode-se observar as classes 'aluno' e 'professor', onde ambas possuem atributos como nome, endereço e telefone. Nesse caso pode-se criar uma nova classe chamada por exemplo, 'pessoa', que contenha as semelhanças entre as duas classes, fazendo com que aluno e professor herdem as características de pessoa, desta maneira pode- se dizer que aluno e professor são subclasses de pessoa.

Para criar uma herança em C#, basta adicionar na declaração da classe um dois pontos (:) seguido do nome da classe que será pai.

Exemplo prático: 02herança

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Teste
{
    public class Pessoa
    {
        public string nome { set; get; }
        public int idade { set; get; }

        public  void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
    }

    public class Maria : Pessoa
    {
        public int sapato { get; set; }
    }

     public static void Main()
    {
        string comida;
        Pessoa teste = new Pessoa();
        Console.Write("Qual comida você dará a Pessoa? ");
        comida = Console.ReadLine();
        teste.comer(comida);
        Maria teste2 = new Maria();
        teste2.sapato = 37;
        teste2.comer(comida);
        Console.WriteLine("Maria calça {0}.", teste2.sapato);
        Console.ReadLine();
    }
}
```

2 – Override (Sobrescrita)

Redefinição de métodos, sobrescrita ou overriding é um mecanismo da programação orientada a objetos. Ele permite que uma subclasse forneça um método que já é fornecido por uma de suas superclasses.

A redefinição ocorre quando um método cuja assinatura já tenha sido especificada recebe uma nova definição (ou seja, um novo corpo) em uma classe derivada.

Como indicar que o método poderá ser reescrito? Utilizando o atributo *virtual* na hora de construir o método da classe pai, exemplo:

```
public virtual void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
```

Para fazer a classe ser sobrescrita, é necessário utilizar a herança. Veja o exemplo 03override:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Teste
{
    public class Pessoa
    {
        public string nome { set; get; }
        public int idade { set; get; }

        public virtual void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
    }

    public class Sergio : Pessoa
    {
        public override void comer(string alimento)
        {
            if (alimento == "peixe")
                Console.WriteLine("Sérgio não gosta de peixe. Não vai comer.");
            else
                base.comer(alimento); //base chama o método pai
        }
    }

     public static void Main()
    {
        string comida;
        Pessoa teste = new Pessoa();
        Console.Write("Qual comida você dará a Pessoa e Sergio? ");
        comida = Console.ReadLine();
        teste.comer(comida);
        Sergio teste3 = new Sergio();
        teste3.comer(comida);
        Console.ReadLine();
    }
}
```

3 – Overload (Sobrecarga de Métodos)

Sobrecarga de método permite a existência de vários métodos de mesmo nome, porém com assinaturas levemente diferentes ou seja variando no número , tipo de argumentos , no valor de retorno e até variáveis diferentes. Ficará a cargo do compilador escolher de acordo com as listas de argumentos os procedimentos ou métodos a serem executados.

Para fazer uma sobrecarga de métodos, basta utilizar a palavra virtual e criar os métodos no mesmo objeto, lembrando que deverão ter argumentos diferentes.

O método chamado será o que tiver os argumentos correspondentes.

Veja o exemplo 04Overload

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

public class Teste
{
    public class Pessoa
    {
        public string nome { set; get; }
        public int idade { set; get; }

        public virtual void comer()
        {
            Console.WriteLine("A Pessoa comendo.");
        }

        public  virtual void comer(string alimento)
        {
            Console.WriteLine("A Pessoa comendo {0}",alimento);
        }
    }


     public static void Main()
    {
        string comida;
        Pessoa teste = new Pessoa();
        Console.Write("Qual comida você dará a Pessoa? ");
        comida = Console.ReadLine();
        teste.comer();
        teste.comer(comida);

        Console.ReadLine();
    }
}
```

Exercício

1 – Criar um classe chamado conta. Esta classe terá as propriedades (atributos) nro_conta e saldo e os métodos depósito (para acrescentar valor a saldo), saque (para retirar valor de saldo, desde que ele não fique negativo) e versaldo (para mostrar o valor de saldo). Depois, crie uma classe chamado conta_especial, que herda a classe conta e acrescenta a propriedade limite, que é um saldo extra acrescido ao saldo. Na classe conta_especial o método saque deve ser reescrito para permitir que haja saque, mesmo que o saldo fique negativo, até o valor de limite.

2 –Instancie um objeto do tipo conta especial e atribua os valores iniciais a esta conta para testar os métodos escritos anteriormente no exercício 1, criando um menu que permita escolher as opções de saque, depósito ou sair.