UNIDADE 8 – CLASSES ABSTRATAS

Classes abstratas tem uma função importante na orientação a objeto. De forma objetiva, uma classe abstrata serve apenas como modelo para uma classe concreta (classe que comumente usamos).

Como classes abstratas são modelos de classes, então, não podem ser instanciadas diretamente com o new, elas sempre devem ser herdadas por classes concretas.

Outro fato importante de classes abstratas é que elas podem conter ou não métodos abstratos, que tem a mesma definição da assinatura de método encontrada em interfaces. Ou seja, uma classe abstrata pode implementar ou não um método.

Os métodos abstratos definidos em uma classe abstrata devem obrigatoriamente ser implementados em uma classe concreta. Mas se uma classe abstrata herdar outra classe abstrata, a classe que herda não precisa implementar os métodos abstratos.

SUMÁRIO

1 CLASSES ABSTRATAS

1.1 CLASSES ABSTRATAS

No banco, todas as contas são de um tipo específico. Por exemplo, conta poupança, conta corrente ou conta salário. Essas contas poderiam ser modeladas através das seguintes classes utilizando o conceito de herança:

```
1  class Conta
2  {
3      // Atributos
4      // Propriedades
5      // Construtores
6      // Métodos
7  }
```

Tabela 1: Conta.cs

```
1  class ContaPoupanca : Conta
2  {
3      // Atributos
4      // Propriedades
5      // Construtores
6      // Métodos
7  }
```

Tabela 2: ContaPoupança.cs

```
1  class ContaCorrente : Conta
2  {
3      // Atributos
4      // Propriedades
5      // Construtores
6      // Métodos
7  }
```

Tabela 3: ContaCorrente.cs

Para cada conta do domínio do banco, devemos criar um objeto da classe correspondente ao tipo da conta. Por exemplo, se existe uma conta poupança no domínio do banco, devemos criar um objeto da classe ContaPoupanca.

```
1  ContaPoupanca cp = new ContaPoupanca();
```

Tabela 4: Criando um objeto da classe ContaPoupanca.cs

Faz sentido criar objetos da classe ContaPoupanca pois existem contas poupança no domínio do banco. Dizemos que a classe ContaPoupanca é uma classe concreta pois criaremos objetos a partir dela.

Por outro lado, a classe Conta não define uma conta que de fato existe no domínio do banco. Ela apenas serve como "base" para definir as contas concretos.

Não faz sentido criar um objeto da classe Conta pois estaríamos instanciado um objeto que não é suficiente para representar uma conta que pertença ao domínio do banco. Mas, a princípio não há nada proibindo a criação de objetos dessa classe. Para adicionar essa restrição no sistema, devemos tornar a classe Conta **abstrata**.

Uma classe concreta pode ser diretamente utilizada para instanciar objetos. Por outro lado, uma classe abstrata não pode. Para definir uma classe abstrata, basta adicionar o modificador **abstract**.

```
1  abstract class Conta
2  {
3      // Atributos
4      // Propriedades
5      // Construtores
6      // Métodos
7  }
```

Tabela 5: Conta.cs

Todo código que tenta criar um objeto de uma classe abstrata não compila.

```
1  // Erro de compilação
2  Conta c = new Conta ();
```

Tabela 6: Erro de compilação

1.2 Métodos Abstratos

Suponha que o banco ofereça extrato detalhado das contas e para cada tipo de conta as informações e o formato desse extrato detalhado são diferentes. Além disso, a qualquer momento o banco pode mudar os dados e o formato do extrato detalhado de um dos tipos de conta.

Neste caso, parece não fazer sentido ter um método na classe Conta para gerar extratos detalhados pois ele seria reescrito nas classes específicas sem nem ser reaproveitado.

Poderíamos, simplesmente, não definir nenhum método para gerar extratos detalhados na classe Conta. Porém, não haveria nenhuma garantia que as classes que derivam direta ou indiretamente da classe Conta implementem métodos para gerar extratos detalhados.

Mas, mesmo supondo que toda classe derivada implemente um método para gerar os extratos que desejamos, ainda não haveria nenhuma garantia em relação as assinaturas desses métodos. As classes derivadas poderiam definir métodos com nomes ou parâmetros diferentes. Isso prejudicaria a utilização dos objetos que representam as contas devido a falta de padronização das operações.

Para garantir que toda classe concreta que deriva direta ou indiretamente da classe Conta tenha uma implementação de método para gerar extratos detalhados e além disso que uma mesma assinatura de método seja utilizada, devemos utilizar o conceito de métodos abstratos.

Na classe Conta, definimos um método abstrato para gerar extratos detalhados. Um método abstrato não possui corpo (implementação).

```
1  abstract class Conta
2  {
3      // Atributos
4      // Propriedades
5      // Construtores
6      public abstract void ImprimeExtratoDetalhado();
7  }
```

Tabela 7: Conta.cs

As classes concretas que derivam direta ou indiretamente da classe Conta devem possuir uma implementação para o método ImprimeExtratoDetalhado().

```
1  class ContaPoupanca : Conta
2  {
3      private int diaDoAniversario;
4
5      public override void ImprimeExtratoDetalhado()
6      {
7          Console.WriteLine(" EXTRATO DETALHADO DE CONTA POUPANÇA ");
8
9          DateTime agora = System.DateTime.Now;
10
11         Console.WriteLine(" DATA : " + agora.ToString("D"));
12         Console.WriteLine(" SALDO : " + this.Saldo);
13         Console.WriteLine(" ANIVERSÁRIO : " + this.diaDoAniversario);
14     }
15 }
```

Tabela 8: ContaPoupanca.cs

Se uma classe concreta derivada da classe Conta não possuir uma implementação do método ImprimeExtratoDetalhado() ela não compilará.

```
1  //Erro de compilação
2  class ContaPoupanca : Conta
3  {
4
5  }
```

Tabela 9: Erro de compilação

1.3 Exercícios de Fixação

1) Crie um projeto no chamado Unidade 8 e a pasta para os Exercicio de Fixacao.

2) Defina uma classe genérica para modelar as contas do banco.

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

Tabela 10: Conta.cs

3) Crie um teste simples para utilizar objetos da classe Conta.

```
1  public class TestaConta
2  {
3      static void Main()
4      {
5          Conta c = new Conta();
6          c.Saldo = 1000;
7
8          System.Console.WriteLine(c.Saldo);
9      }
10 }
```

Tabela 11: TestaConta.cs

4) Torne a classe Conta abstrata e verifique o que acontece na classe de teste.

```
1  public abstract class Conta
2  {
3      public double Saldo { get; set; }
4  }
```

Tabela 12: Conta.cs

5) Defina uma classe paramodelar as contas poupança do nosso banco.

```
1  class ContaPoupanca : Conta
2  {
3      public int DiaDoAniversario { get; set; }
4  }
```

Tabela 13: ContaPoupanca.cs

6) Altere a classe TestaConta para corrigir o erro de compilação.

```
1  public class TestaConta
2  {
3      private static void Main()
4      {
5          Conta c = new ContaPoupanca();
6
7          c.Saldo = 1000;
8
9          Console.WriteLine(c.Saldo);
10     }
11 }
```

Tabela 14: TestaConta.cs

7) Defina um método abstrato na classe Conta para gerar extratos detalhados.

```
1  abstract class Conta
2  {
3      public double Saldo { get; set; }
4      public abstract void ImprimeExtratoDetalhado();
5  }
6
```

Tabela 15: Conta.cs

8) O que acontece com a classe ContaPoupanca?

9) Defina uma implementação do método ImprimeExtratoDetalhado() na classe ContaPoupanca.

```
1  class ContaPoupanca : Conta
2  {
3
4      public int DiaDoAniversario { get; set; }
5
6      public override void ImprimeExtratoDetalhado()
7      {
8          Console.WriteLine(" EXTRATO DETALHADO DE CONTA POUPANÇA ");
9
10         DateTime agora = System.DateTime.Now;
11
12         Console.WriteLine(" DATA : " + agora.ToString("D"));
13         Console.WriteLine(" SALDO : " + this.Saldo);
14         Console.WriteLine(" ANIVERSÁRIO : " + this.DiaDoAniversario);
15     }
   }
```

Tabela 16: ContaPoupanca.cs

10) Altere a classe TestaConta para chamar o método ImprimeExtratoDetalhado().

```
1  public class TestaConta
2  {
3      private static void Main()
4      {
5          Conta c = new ContaPoupanca();
6
7          c.Saldo = 1000;
8
9          c.ImprimeExtratoDetalhado();
10     }
11 }
```

Tabela 17: TestaConta.cs

1.4 Exercícios Complementares

1) Crie uma pasta exercícios complementares no projeto Unidade 8.

2) Defina uma classe genérica para modelar os funcionários do banco.

3) Crie um objeto da classe que modela os funcionários do banco e utilize os métodos de acesso com nomes padronizados para alterar os valores dos atributos.

4) Torne a classe que modela os funcionários do banco abstrata e verifique o que acontece na classe de teste.

5) Defina uma classe para modelar os gerentes do nosso banco.

6) Altere a classe de teste e crie um objeto da classe que modela os gerentes.

7) Defina um método abstrato na classe que modela os funcionários para calcular a bonificação dos colaboradores.

8) O que acontece com a classe que modela os gerentes?

9) Implemente o método que calcula a bonificação na classe que modela os gerentes.

10) Altere a classe de teste para que o método que calcula a bonificação seja chamada e o valor seja impresso na tela.

Bons Estudos!