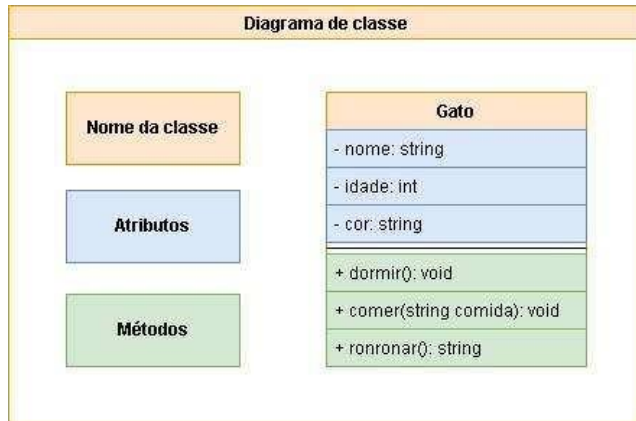


# Generation

## BRASIL

1.



A imagem representa:



Uma classe em UML

☐

Uma classe abstrata em UML



Uma interface em UML

2.



Na imagem podemos ver:



Um exemplo de Polimorfismo

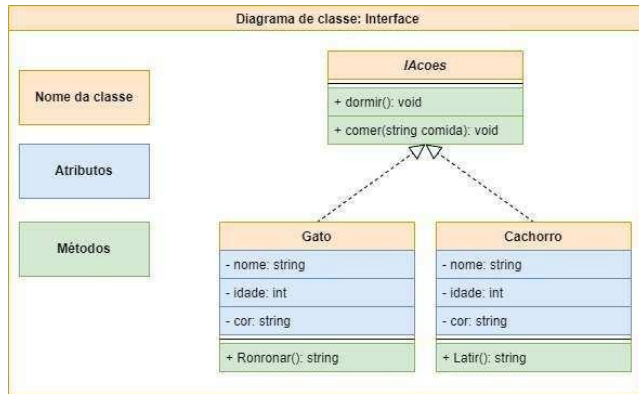
☐

Um exemplo de Interface



Um exemplo de Herança

3.



Na imagem é possível ver:

☐ A Todas estão corretas ☒ B Um exemplo de Interface

☐ C Um exemplo de Herança

4. A **herança** é realizada usando uma derivação, o que significa que uma classe é declarada usando uma classe base, da qual ela herda o comportamento e os dados.

☐ A Falso ☒ B Verdadeiro

5. A classe **Object**, é considerada uma herança implícita, e todas as classes possuem uma herança implícita com a classe **Object** que por si possui 8 membros (métodos) em sua especificação!.

☐ A Falso ☒ B Verdadeiro

6. Uma **interface** não pode conter qualquer tipo de código, muito menos código padrão. Uma classe **abstrata** pode fornecer código completo, código padrão ou ter apenas a declaração de seu esqueleto para ser posteriormente sobrescrita.

☒ A Verdadeiro ☐ B Falso

7.

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace Zoologico.src
7 {
8     2 references
9     public class Gato : Animal
10     {
11         0 references
12         public Gato() { }
13
14         2 references
15         public Gato(string nome, string cor, string classificacao) : base(nome, cor, classificacao) { }
16
17         3 references
18         public override void Comunicar(string comunicacao)
19         {
20             console.WriteLine($"{Nome} está falando: {comunicacao}");
21         }
22     }
23 }
  
```

☒ A Na linha 12, esta definido um construtor de Gato, que define o construtor da classe Animal ☐ B Na linha 10 o método construtor da classe Gato recebe parâmetros que definem a classe Animal

☒ C Na linha 14 esta definido um método que sobrescreve o método Comunicar da classe Animal ☒ D Na linha 14 o método pertence a palavra override informa que o método esta sobrescrevendo

8.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace Calculadora.src
8  {
9      2 references
10     internal interface IOperacoes
11     {
12         1 reference
13         double Somar(double a, double b);
14         1 reference
15         double Subtrair(double a, double b);
16         1 reference
17         double Multiplicar(double a, double b);
18         1 reference
19         double Dividir(double a, double b);
20     }
21 }

```

IOperacoes, é uma:



A Uma interface, que possui apenas a assinatura dos métodos



B Classe abstrata, que possui apenas a assinatura dos métodos



C Classe normal, que possui apenas a assinatura dos métodos

9.

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Threading.Tasks;
5
6  namespace Venda.src
7  {
8      2 references
9      public abstract class Bonus
10     {
11         2 references
12         public abstract double CalculaBonus(double venda);
13     }
14 }

```

Em uma classe **abstrata** é correto afirmar que:



A Os métodos podem ser de assinatura e implementados



B Os métodos podem ser somente de assinatura



C Os métodos podem ser somente implementados

10.

Para quais casos é possível utilizar uma classe **abstrata** e uma **interface**, respectivamente.



A Para casos onde é definido um conjunto de assinatura de métodos que outras classes devem implementar e casos que precisam se preocupar com o comportamento padrão



B Para casos que precisem se preocupar com o comportamento padrão e casos onde é definido um conjunto de assinatura de métodos que outras classes devem implementar



C Para quaisquer casos, na realidade não é importante o conceito de abstrato e interface

