IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Tipos Genéricos

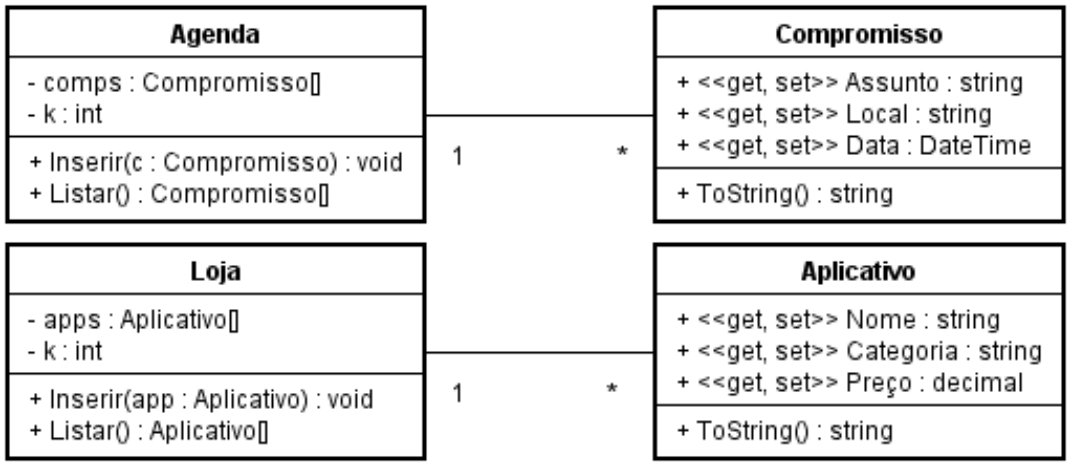
Prof. Gilbert Azevedo

Objetivos

- Conhecer e utilizar tipos genéricos
  - Contextualização, parâmetros de tipo e classes genéricas
- Conhecer e utilizar métodos genéricos e interfaces genéricas

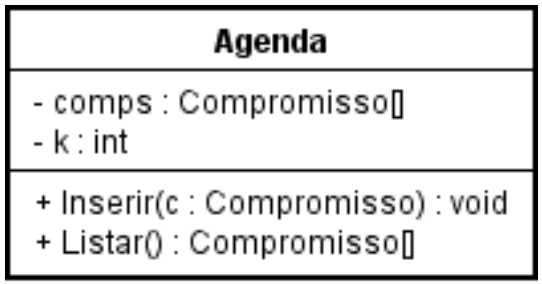
Contextualização

- Na modelagem de sistemas, é comum classes distintas possuírem atributos e métodos semelhantes.
- As classes Agenda e Loja, por exemplo, são contêineres para Compromissos e Aplicativos.
- As classes Agenda e Loja tem atributos e métodos semelhantes.



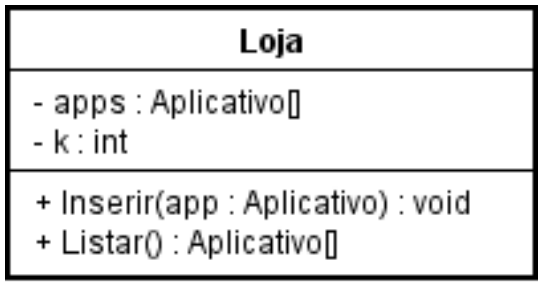
Classe Agenda

```
class Agenda {
  private Compromisso[] comps = new Compromisso[50];
  private int k;
  public void Inserir(Compromisso c) {
    if (k < 50) comps[k++] = c;
  }
  public Compromisso[] Listar() {
    Compromisso[] r = new Compromisso[k];
    Array.Copy(comps, r, k);
    return r;
  }
}
```



Classe Loja

```
class Loja {
  private Aplicativo[] apps = new Aplicativo[50];
  private int k;
  public void Inserir(Aplicativo app) {
    if (k < 50) apps[k++] = app;
  }
  public Aplicativo[] Listar() {
    Aplicativo[] r = new Aplicativo[k];
    Array.Copy(apps, r, k);
    return r;
  }
}
```



Comparação entre Agenda e Loja

- O tipo dos elementos é a única diferença entre as classes

```
class Agenda {
  private Compromisso[] comps = new
    Compromisso[50];
  private int k;
  public void Inserir(Compromisso c) {
    if (k < 50) comps[k++] = c;
  }
  public Compromisso[] Listar() {
    Compromisso[] r = new Compromisso[k];
    Array.Copy(comps, r, k);
    return r;
  }
}
```

```
class Loja {
  private Aplicativo[] apps = new
    Aplicativo[50];
  private int k;
  public void Inserir(Aplicativo app) {
    if (k < 50) apps[k++] = app;
  }
  public Aplicativo[] Listar() {
    Aplicativo[] r = new Aplicativo[k];
    Array.Copy(apps, r, k);
    return r;
  }
}
```

Parâmetro de Tipo

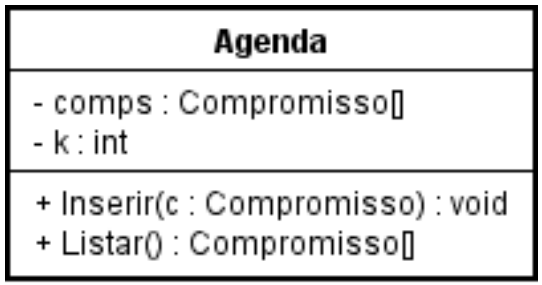
- A classe Coleção é um modelo que utiliza um parâmetro de tipo

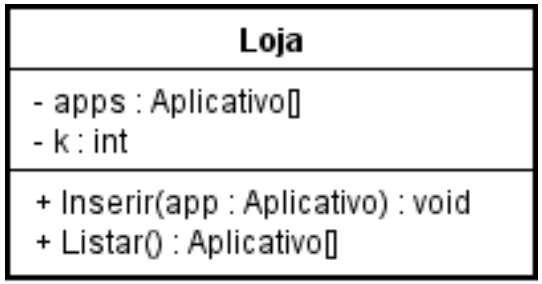
```
class Agenda {
  private Compromisso[] comps = new
    Compromisso[50];
  private int k;
  public void Inserir(Compromisso c) {
    if (k < 50) comps[k++] = c;
  }
  public Compromisso[] Listar() {
    Compromisso[] r = new Compromisso[k];
    Array.Copy(comps, r, k);
    return r;
  }
}
```

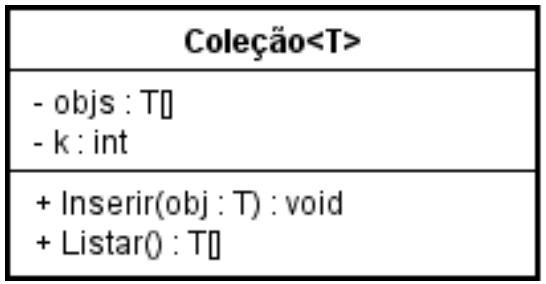
```
class Coleção<T> {
  private T[] objs = new
    T[50];
  private int k;
  public void Inserir(T obj) {
    if (k < 50) objs[k++] = obj;
  }
  public T[] Listar() {
    T[] r = new T[k];
    Array.Copy(objs, r, k);
    return r;
  }
}
```

Classe Genérica

- Classe genérica é uma classe modelo que usa parâmetros de tipo
  - Parâmetros de tipo (T) são usados para informar, ao instanciar uma classe, o tipo de atributos, parâmetros e retorno de métodos
  - Classes genéricas possibilitam a escrita de classes com comportamentos padronizados, evitando a reescrita de código
  - Genéricos são bastante usados em coleções de objetos, devido aos comportamentos padrões das coleções: inserir, listar, excluir, atualizar, ...

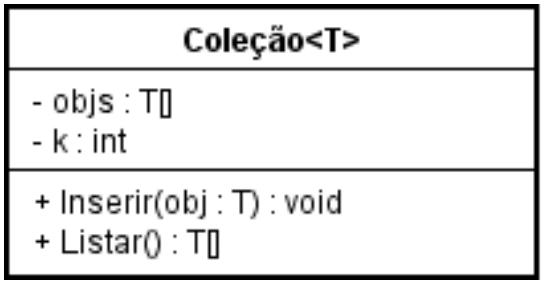






Instanciando Classes Genéricas

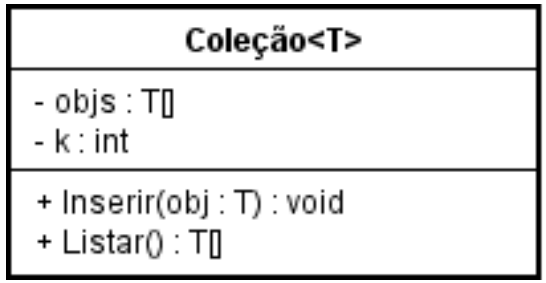
- Quando uma classe genérica é instanciada, o parâmetro de tipo deve ser informado na referência e no construtor.
  - O tipo informado é substituído em todos os atributos, parâmetros e métodos da classe.




```
public static void Main (string[] args) {
  Agenda agenda1 = new Agenda();
  Coleção<Compromisso> agenda2 = new Coleção<Compromisso>();
  Loja loja1 = new Loja();
  Coleção<Aplicativo> loja2 = new Coleção<Aplicativo>();
}
```

Métodos da Classe Genérica

- Os métodos da classe genérica verificam o tipo do parâmetro informado na chamada (type-safe)




```
public static void Main (string[] args) {
  Coleção<Aplicativo> loja = new Coleção<Aplicativo>();
  loja.Inserir(new Aplicativo{Nome = "Nome", Categoria = "Categoria",
    Preço = 0.0M});
}
```

Métodos Genéricos

- Métodos genéricos permitem especificar parâmetros e tipos de retorno utilizando um parâmetro de tipo e podem ser escritos em qualquer classe (mesmo não genérica)

```
public static void Swap<T>(ref T x, ref T y) {
  T aux = x;
  x = y;
  y = aux;
}
```

Usando o Método Genérico

- Invocando o método genérico para trocar os valores de dois inteiros

```
int a = 1, b = 2;
Swap<int>(ref a, ref b);
```

- Invocando o método genérico para trocar os valores de duas strings

```
string s1 = "hello", s2= "world";
Swap<string>(ref s1, ref s2);
```

Interfaces Genéricas

- Definem comportamentos para classes utilizando parâmetros de tipo
- IComparable<T>
  - Define o comportamento de comparação entre objetos
- IEnumerable<T>
  - Define o comportamento de iteração em uma coleção

Implementação de IComparable<T>

- IComparable<T> elimina a necessidade do type-cast

```
class Aluno : IComparable {
  private string nome;
  private DateTime nasc;
  public Aluno(string n, DateTime d) {
    this.nome = n;
    this.nasc = d;
  }
  public int CompareTo(object obj) {
    return
      nome.CompareTo(((Aluno)obj).nome);
  }
}
```

```
class Aluno : IComparable<Aluno> {
  private string nome;
  private DateTime nasc;
  public Aluno(string n, DateTime d) {
    this.nome = n;
    this.nasc = d;
  }
  public int CompareTo(Aluno obj) {
    return nome.CompareTo(obj.nome);
  }
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- Genéricos – Guia de Programação em C#
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/generics/
- Métodos Genéricos – Guia de Programação em C#
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/generics/generic-methods
- Interfaces – Guia de Programação em C#
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/interfaces/