IFRN

PROGRAMAÇÃO ORIENTADA A OBJETOS EM C#

Interfaces – Parte 02/02

Prof. Gilbert Azevedo

Objetivos

- Revisar os conceitos de interface
- Definir e utilizar novas interfaces na escrita de sistemas
- Utilizar interfaces de iteração em Coleções
  - IEnumerable, IEnumerator

Interface – Resumo

- Estabelece um “contrato” para um tipo, ou seja, um comportamento desejado para uma classe ou estrutura (struct)
  - Define apenas um comportamento
  - Não possuem estado nem identidade
- A interface especifica uma lista de métodos e propriedades
  - Não é possível especificar atributos
  - Não é possível especificar construtores
- Todos os membros da interface são públicos
  - Não utiliza modificadores de acesso (public, private, protected, internal, ...)

Interface – Resumo

- A interface IComparable do *framework* define o método CompareTo como padrão de comparação entre objetos

```
interface IComparable {
    int CompareTo (object obj);
}
```

- O método CompareTo deve comparar dois objetos e o seu resultado é usado por um método de ordenação
- O método Array.Sort, por exemplo, utiliza a comparação definida pelo CompareTo na ordenação de um vetor

Referência para Interface – Resumo

- Variáveis da interface podem referenciar objetos que a implementam
  - Apenas os métodos da interface podem ser invocados

```
IComparable a = 10;
IComparable b = 20;
IComparable x = "Java";
IComparable y = "C#";
Console.WriteLine(a.CompareTo(b)); // -1
Console.WriteLine(x.CompareTo(y)); // 1
```

Definição de Interfaces

- Interfaces são definidas com a palavra reservada *interface* e iniciam, por padrão, com a letra I

```
interface IPessoa {
  string GetNome();
  DateTime GetNascimento();
}
```

- A interface IPessoa define os métodos GetNome e GetNascimento
- Tipos que implementem a interface IPessoa devem definir os dois métodos

Implementação de IPessoa

- Implementação de IPessoa pela classe Aluno
  - Os métodos GetNome e GetNascimento devem ser públicos

```
class Aluno : IComparable, IPessoa {
  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);
  }
  public string GetNome() { return nome; }
  public DateTime GetNascimento() { return nasc; }
}
```

Outra Implementação de IPessoa

- Implementação de IPessoa pela classe Professor
  - A classe pode ter vários métodos além dos definidos na interface

```
class Professor : IPessoa {
  private string nome, grad;
  private DateTime nasc;
  public Professor(string n, string g, DateTime d) {
    this.nome = n; this.grad = g;
    this.nasc = d;
  }
  public string GetNome() { return nome; }
  public DateTime GetNascimento() { return nasc; }
  public string GetGraduacao() { return grad; }
}
```

Referências de Interface

- É possível referenciar objetos das classes Aluno e Professor com uma referência para IPessoa

```
IPessoa a = new Aluno("Pedro", DateTime.Parse("01/01/1992"));
IPessoa b = new Professor("Alice", "Engenharia",
  DateTime.Parse("01/01/1970")) ;
Console.WriteLine(a.GetNome());
Console.WriteLine(a.GetNascimento());
Console.WriteLine(a.CompareTo(b)); // Erro
Console.WriteLine(b.GetNome());
Console.WriteLine(b.GetNascimento());
Console.WriteLine(b.GetGraducao()); // Erro
```

Utilização da Interface

- A classe Relatorio imprime uma lista de aniversariantes de qualquer classe que implemente IPessoa
  - Lista de aniversariantes de alunos e professores usam a mesma classe de relatório

```
class Relatorio {
  public static void Aniversariantes(IPessoa[] v, int mes) {
    foreach (IPessoa p in v)
      if (p.GetNascimento().Month == mes)
        Console.WriteLine($"{p.GetNome()} {p.GetNascimento():dd/MM}");
  }
}
```

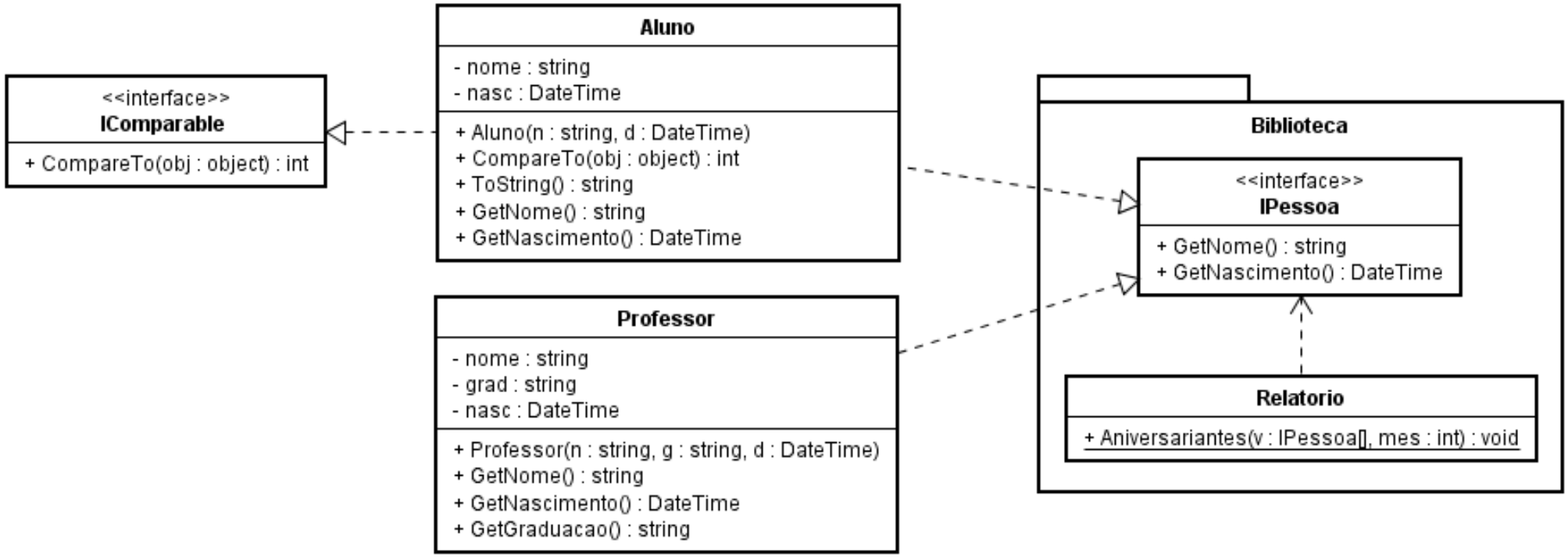
Exemplo IPessoa

```
class MainClass {
  public static void Main (string[] args) {
    Aluno a = new Aluno("Pedro", DateTime.Parse("01/01/1992"));
    Professor b = new Professor("Alice", "Engenharia",
      DateTime.Parse("01/01/1970")) ;
    IPessoa[] v = {a, b};
    Relatorio.Aniversariantes(v, 1);
  }
}

class Relatorio {
  public static void Aniversariantes(IPessoa[] v, int mes) {
    foreach (IPessoa p in v)
      if (p.GetNascimento().Month == mes)
        Console.WriteLine($"{p.GetNome()} {p.GetNascimento():dd/MM}");
  }
}
```

Diagrama de Classes

- IPessoa e Relatorio são totalmente independentes e funcionais
  - A interface e a classe podem ser empacotadas e utilizadas em diversas aplicações



Interfaces com Propriedades

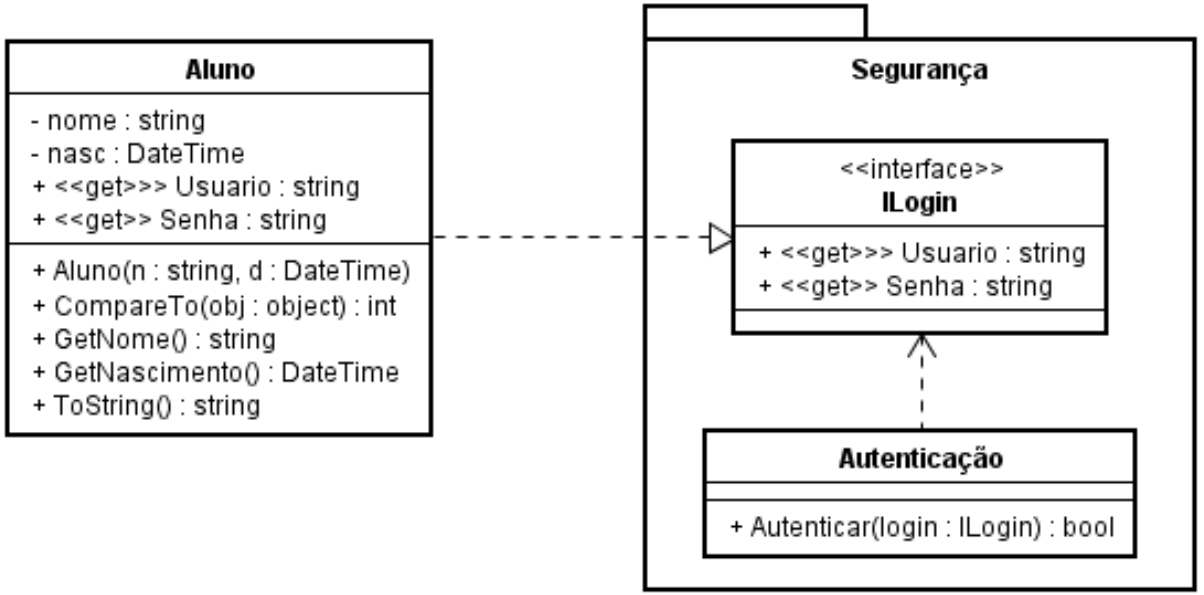
- Além de métodos, interfaces podem listar propriedades a serem implementadas por uma estrutura ou classe.

```
interface ILogin {
  string Usuario { get; }
  string Senha { get; }
}
```

- A interface ILogin define as propriedades somente-leitura Usuario e Senha que podem ser usadas por uma biblioteca de autenticação
- Tipos que implementem a interface ILogin devem definir as duas propriedades

Interfaces com Propriedades

- ILogin e Autenticação podem ser empacotadas para prover controle de usuários para várias aplicações



Interfaces de Iteração

- As interfaces IEnumerable e IEnumerator definem um padrão de iteração em coleções de objetos
- Classes que implementam IEnumerable podem utilizar a instrução *foreach* para percorrer os objetos da coleção
- A interface IEnumerabler define o método GetEnumerator que deve retornar um iterador para a coleção

```
interface IEnumerable {
  IEnumerator GetEnumerator();
}
```

Interface IEnumerator

- IEnumerator define o comportamento do iterador usado para percorrer a coleção, utilizado pelo *foreach*.
  - A propriedade Current retorna o objeto atualmente selecionado
  - MoveNext move para o próximo objeto da coleção
  - Reset reinicia a iteração.

```
interface IEnumerator {
  object Current { get; }
  bool MoveNext();
  void Reset();
}
```

Implementação de IEnumerable

- A classe Turma implementa a interface IEnumerable

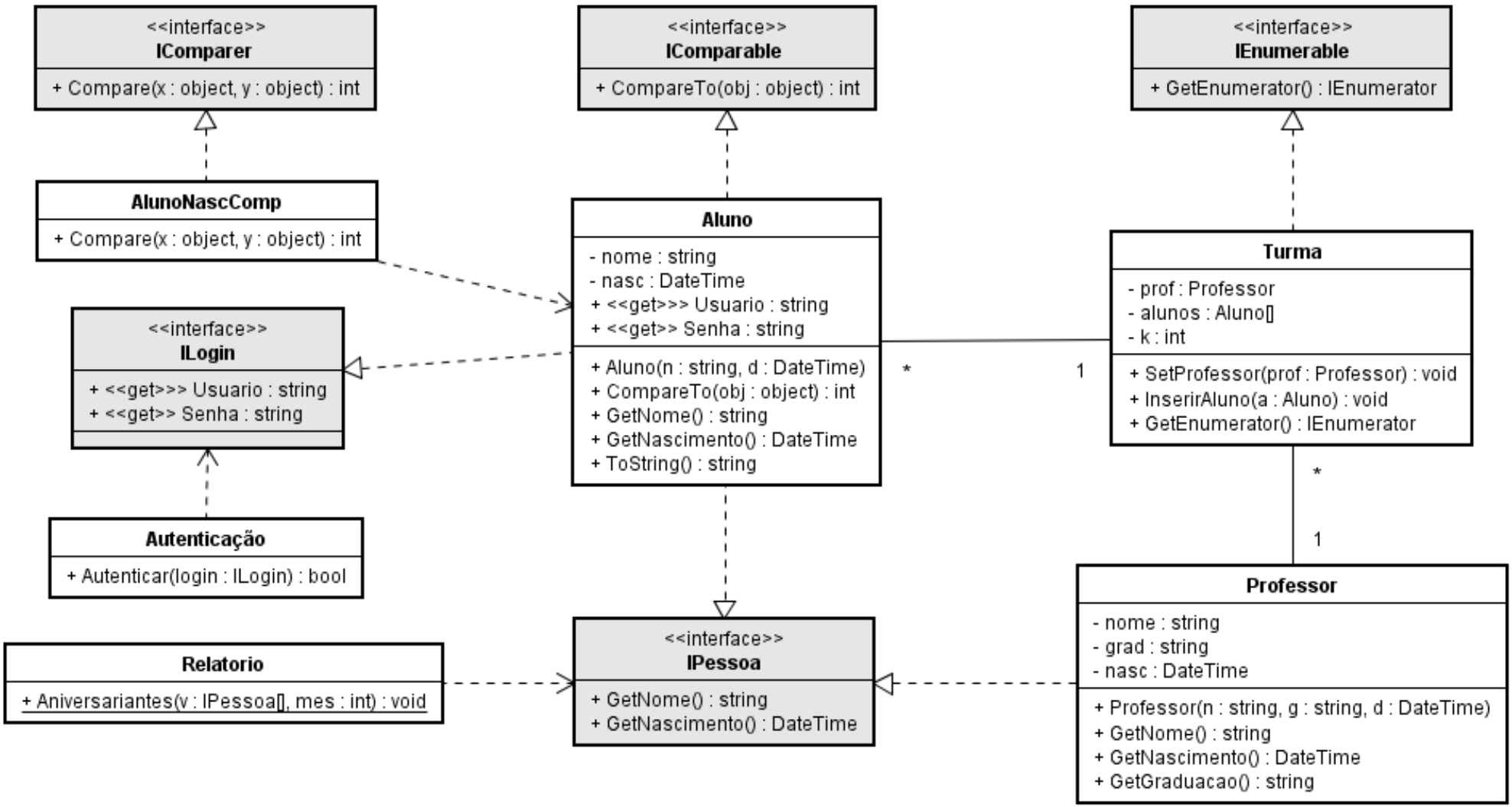
```
class Turma : IEnumerable {
  private Professor prof;
  private Aluno[] alunos = new Aluno[40];
  private int k;
  public void SetProfessor(Professor prof) { this.prof = prof; }
  public void InserirAluno(Aluno a) { alunos[k++] = a; }
  public IEnumerator GetEnumerator() {
    Aluno[] v = new Aluno[k];
    Array.Copy(alunos, v, k);
    return v.GetEnumerator();
  }
}
```

Foreach em Objeto Turma

- Como a classe Turma implementa IEnumerable é possível utilizar um *foreach* para acessar os alunos da turma

```
public static void Main (string[] args) {
  Aluno a = new Aluno("Pedro", DateTime.Parse("01/01/1992"));
  Aluno b = new Aluno("Maria", DateTime.Parse("01/01/1993"));
  Aluno c = new Aluno("Paulo", DateTime.Parse("01/01/1991"));
  Professor p = new Professor("Alice", "Engenharia",
    DateTime.Parse("01/01/1970")) ;
  Turma t = new Turma();
  t.SetProfessor(p);  t.InserirAluno(a);
  t.InserirAluno(b);  t.InserirAluno(c);
  foreach(Aluno aluno in t) Console.WriteLine(aluno);
}
```

Diagrama de Classes Final



Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- UML – Uma Abordagem Prática, Gilleanes T. A. Guedes, Novatec, 2004
- Interfaces (Guia de Programação em C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/interfaces/index