Linguagem de Programação 3

Aula 6
Programação Orientada a Objetos com C#

l.bertholdo@ifsp.edu.br

Conteúdo

- Programação Orientada a Objetos – POO
  - Classes Abstratas
  - Interfaces

l.bertholdo@ifsp.edu.br

Classes Abstratas

- Tipo de polimorfismo onde um método de uma classe mãe pode ter diferentes implementações em suas classes filhas, sem que exista uma implementação genérica deste método na classe mãe.
- Para implementar este tipo de polimorfismo, é necessário criar uma classe mãe abstrata, cujos métodos abstratos poderão ser utilizados de diferentes formas por suas classes filhas.
- Uma classe abstrata pode ter tanto métodos **abstratos** (sem implementação) quanto **concretos** (com implementação).
- Ao herdar de uma classe abstrata, a classe filha necessariamente deve implementar todos os métodos abstratos desta classe.

l.bertholdo@ifsp.edu.br

Classes Abstratas

- A ideia é que um método abstrato seja sempre herdado. Portanto, então ele nunca poderá ser privado.
- Um método **abstrato** é semelhante a um método **virtual**, com a diferença de que não contém implementação. Seu código é escrito de diferentes formas nas classes filhas.
- Para indicar que uma classe é abstrata, deve-se usar a palavra reservada **abstract** antes da palavra **class**.
- Para indicar que um método é abstrato, deve-se usar a palavra reservada **abstract** antes do tipo de retorno do método.

```
abstract class ClasseAbstrata{

    protected abstract void MetodoAbstrato();

}
```

l.bertholdo@ifsp.edu.br

Classes Abstratas

- O método da classe filha que implementa o método abstrato da classe mãe deve ter em sua declaração a palavra **override** para indicar a sobreposição. Exemplo:

```
class ClasseFilha : ClasseAbstrata{
        protected override void Metodo(){
                // Código do método
        }
}
```

- As assinaturas do método **abstrato** e do seu respectivo método **implementado na classe filha** devem ser idênticas, ou seja, devem ter o mesmo tipo de retorno, nome, número e tipos de argumentos.

- Além disso, o nível de proteção do método não pode ser alterado na classe filha.

l.bertholdo@ifsp.edu.br

Classes Abstratas

- Não é possível criar um objeto do tipo de uma classe abstrata que instancie esta mesma classe abstrata.

  Exemplo: ClasseAbstrata objeto = new ClasseAbstrata()

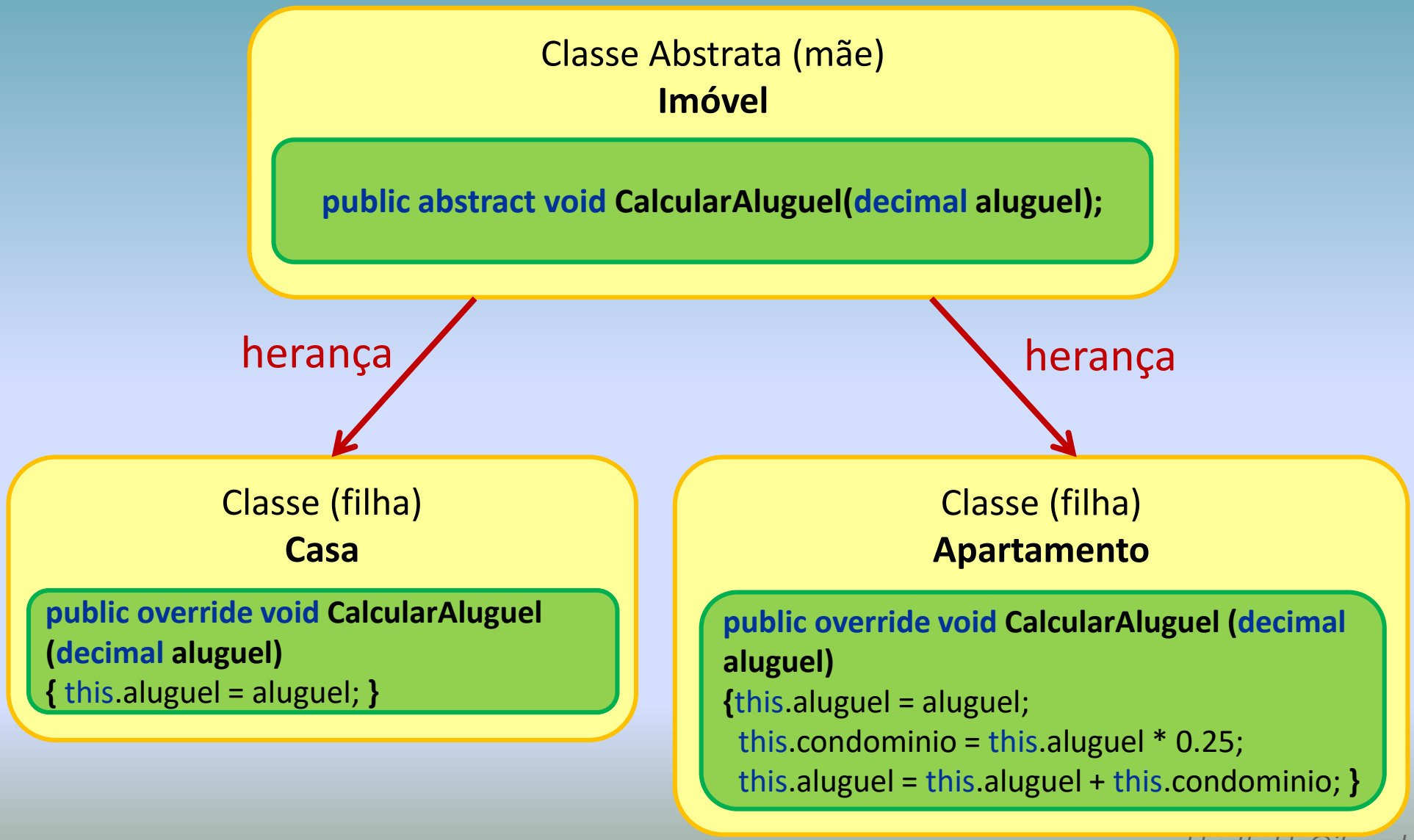
- Mas é possível criar um objeto do tipo de uma classe abstrata que instancie uma de suas classes filhas.

  Exemplo: ClasseAbstrata objeto = new ClasseFilha();

- Com isso, o objeto do tipo da classe abstrata pode chamar os **métodos concretos** da própria classe abstrata, e também os **métodos abstratos** implementados nas filhas da classe abstrata.

l.bertholdo@ifsp.edu.br

Classes Abstratas



l.bertholdo@ifsp.edu.br

Classes Abstratas

```
class Inicio{
    static void Main(string[] args){
        new Casa(); // Chama o construtor da classe Casa.
        new Apartamento(); // Chama o construtor da classe Apartamento.
        Console.ReadKey();
    }
}
```

```
abstract class Imovel{
    private string endereco;
    private decimal valor;

    protected void SetEndereco(string endereco){
        this.endereco = endereco;
        Console.WriteLine("Endereço cadastrado: " + this.endereco);
    }

    protected void SetValor(decimal valor) {
        this.valor = valor;
        Console.WriteLine("Valor cadastrado: " + this.valor);
    }

    public abstract void CalcularAluguel(decimal aluguel);
}
```

l.bertholdo@ifsp.edu.br

Classes Abstratas

```
class Apartamento : Imovel{
    private int andar;
    private decimal aluguel, condominio;

    public Apartamento(){
        base.SetEndereco("Rua Itu, 301");
        base.SetValor(300000);
        this.SetAndar(7);
        this.CalcularAluguel(1000);
    }

    public void SetAndar(int andar){
        this.andar = andar;
        Console.WriteLine("Andar cadastrado: " + this.andar);
    }

    // Sobrescreve o método CalcularAluguel da classe Imovel.
    public override void CalcularAluguel(decimal aluguel){
        this.aluguel = aluguel;
        this.condominio = this.aluguel * Convert.ToDecimal(0.25);
        this.aluguel = this.aluguel + this.condominio;
        Console.WriteLine("Aluguel do apto: " + this.aluguel);
    }
}
```

```
class Casa : Imovel{
    private int metroQuadrado;
    private decimal aluguel;

    public Casa(){
        base.SetEndereco("Rua Manaus, 220");
        base.SetValor(450000);
        this.SetAreaTerreno(250);
        this.CalcularAluguel(1500);
    }

    public void SetAreaTerreno(int metroQuadrado){
        this.metroQuadrado = metroQuadrado;
        Console.WriteLine("Área cadastrada: " + this.metroQuadrado);
    }

    // Sobrescreve o método CalcularAluguel da classe Imovel.
    public override void CalcularAluguel(decimal aluguel){
        this.aluguel = aluguel;
        Console.WriteLine("Aluguel da casa: " + this.aluguel);
    }
}
```

l.bertholdo@ifsp.edu.br

Interfaces

- Interfaces são semelhantes a classes abstratas, porém há algumas diferenças:
  - Interfaces não podem ter atributos, somente métodos.
  - Nas interfaces todos os métodos são obrigatoriamente **públicos** e **abstratos** (sem implementação). Assim, torna-se desnecessário usar as palavras reservadas **public** e **abstract**.
- Em resumo, a interface indica apenas **o que o método deve receber e retornar** (por exemplo: receber um valor inteiro e retornar um valor booleano) e não **o que ele deve fazer**. Isso é definido nos métodos das classes que implementam a interface.

l.bertholdo@ifsp.edu.br

Interfaces

- A declaração de interfaces é similar à das classes, porém usa-se a palavra reservada **interface** ao invés de **class**. A documentação do .NET recomenda que os nomes de interfaces iniciem com a letra "I".

```
interface IFuncionario{
        void CalcularVencimento();
}
```

- Para indicar que uma classe implementa uma interface, após o nome da classe, deve-se incluir dois pontos (:) seguido pelo nome da interface. Exemplo:

```
class Balconista : IFuncionario {
        public void CalcularVencimento(){
                // Código do método
        }
}
```

l.bertholdo@ifsp.edu.br

Interfaces

- Não é possível criar um objeto do tipo de uma interface que instancie esta mesma interface.

  Exemplo: Interface objeto = new Interface();

- Mas é possível criar um objeto do tipo de uma interface que instancie uma classe que implemente esta interface.

  Exemplo: Interface objeto = new ClasseQueImplementaInterface();

- Com isso, o objeto do tipo da inteface pode chamar os **métodos abstratos** codificados nas classes que implementam a interface.

Interfaces

- Um classe pode implementar mais de uma interface. Nesse caso, é necessário separá-las por vírgula na declaração da classe. Exemplo: class Classe : Interface1, Interface2, Interface3
- Ao implementar uma interface, a classe precisa obrigatoriamente conter todos os métodos desta interface, ainda que sem implementação.
- O uso de interfaces visa deixar o código mais flexível e possibilitar alterações na implementação sem maiores dificuldades.

l.bertholdo@ifsp.edu.br

Interfaces

```
class AuxAdm:IFuncionario{
    private double vencimento;
    private double salario = 1200;
    private double valeAlimentacao = 200;

    public double CalcularVencimento() {
        vencimento = salario + valeAlimentacao;
        return vencimento;
    }
}
```

```
class Vendedor:IFuncionario{
    private double vencimento;
    private double salario = 1200;
    private double valeAlimentacao = 200;
    private double comissao = 300;

    public double CalcularVencimento(){
        vencimento = salario + valeAlimentacao + comissao;
        return vencimento;
    }
}
```

```
class Gerente:IFuncionario{
    private double vencimento;
    private double salario = 3000;
    private double valeAlimentacao = 200;
    private double descPlanoSaude = 500;

    public double CalcularVencimento(){
        vencimento = salario + valeAlimentacao - descPlanoSaude;
        return vencimento;
    }
}
```

```
interface IFuncionario{
    double CalcularVencimento();
}
```

Interfaces

Para o método **EfetuarPagto** não importa de que tipo é o funcionário, ele apenas tem que chamar o método **CalcularVencimento**.

```
class Pagamento{
    public void EfetuarPagto(IFuncionario f){
        // Chama o método CalcularVencimento da classe para a qual objeto IFuncionario está apontando.
        Console.WriteLine("O vencimento do funcionário é: " + f.CalcularVencimento());
        Console.ReadKey();
    }
}
```

```
class Inicio{
    static void Main(string[] args){
        Pagamento p = new Pagamento();  // Cria um objeto Pagamento para chamar o método EfetuarPagto.

        IFuncionario f = new AuxAdm();  // Cria um objeto IFuncionario que instancia a classe AuxAdm.
        p.EfetuarPagto(f);

        f = new Vendedor();  // O objeto IFuncionario passa a instanciar a classe Vendedor.
        p.EfetuarPagto(f);

        f = new Gerente();  // O objeto IFuncionario passa a instanciar a classe Gerente.
        p.EfetuarPagto(f);
    }
}
```

Ao criar um objeto do tipo **IFuncionario**, pode-se instanciar qualquer classe que implemente a interface **IFuncionario** (AuxAdm, Vendedor ou Gerente).

Se um dia surgir um novo tipo de funcionário, basta criar uma classe para ele (que implemente a interface **IFuncionario**) e um objeto **IFuncionario** (que instancie esta nova classe). As classes Pagamento, AuxAdm, Vendedor, Gerente e a interface IFuncionario não precisarão sofrer qualquer alteração.

Referências

- Henrique Loureiro; C# 6.0 com Visual Studio – Curso Completo. FCA, 2015.
- John Sharp; Microsoft Visual C# 2013: Passo a Passo. Bookman, 2014.
- http://www.caelum.com.br/apostila-java-orientacao-objetos/interfaces/#10-2-interfaces