PONTIFÍCIA UNIVERSIDADE CATÓLICA DE GOIÁS
DEPARTAMENTO DE COMPUTAÇÃO
TÉCNICAS DE PROGRAMAÇÃO 1 – CMP 1046
PROF. MSC. ANIBAL SANTOS JUKEMURA

[Polimorfismo]
[Classes e Métodos Abstratos]

PUC GOIÁS

Agenda:

- Polimorfismo
  - Conceitos
  - ***Exemplo***
  - Extensibilidade
- Exercício: implementação
- Classes e métodos abstratos

PUC GOIÁS

Polimorfismo

Polimorfismo significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C# e C++, que permite ao desenvolvedor usar o mesmo elemento de formas diferentes.

Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem. No Polimorfismo temos dois tipos:

- **Polimorfismo Estático ou Sobrecarga**
- **Polimorfismo Dinâmico ou Sobreposição**

PUC GOIÁS

Polimorfismo

- **Polimorfismo Estático** ou **Sobrecarga** (caso do construtor)

  O Polimorfismo Estático se dá quando temos a mesma operação implementada várias vezes na mesma classe. A escolha de qual operação será chamada depende da assinatura dos métodos sobrecarregados.

- **Polimorfismo Dinâmico** ou **Sobreposição**

  O Polimorfismo Dinâmico acontece na **herança**, quando a subclasse sobrepõe o método original. Agora o método escolhido se dá em tempo de execução e não mais em tempo de compilação. A escolha de qual método será chamado depende do tipo do objeto que recebe a mensagem.

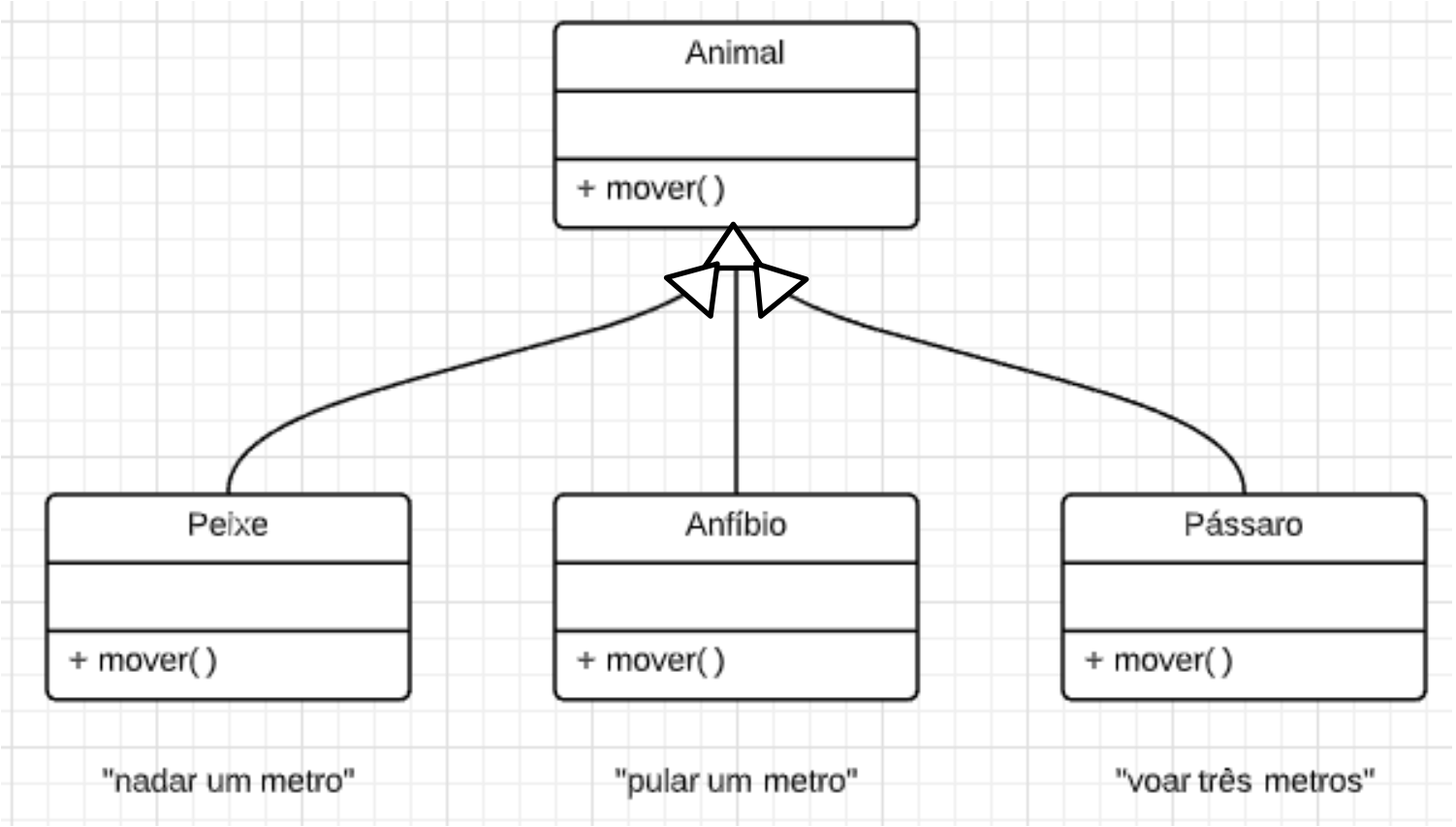
PUC GOIÁS

Polimorfismo

- O polimorfismo permite “programar no *geral*” em vez de “programar no *específico*”.
- Em particular, o polimorfismo permite escrever programas que processam **objetos que compartilham a mesma superclasse**, direta ou indiretamente, **como se todos fossem objetos da superclasse**; isso pode simplificar a programação.

PUC GOIÁS

Polimorfismo



A *mesma* mensagem (nesse caso, mover) enviada a uma *variedade* de objetos tem *muitas formas* de resultados — daí o termo polimorfismo.

PUC GOIÁS

Polimorfismo - Extensibilidade

- Com o polimorfismo, pode-se projetar e implementar sistemas que são facilmente **extensíveis** — novas classes podem ser adicionadas com pouca ou nenhuma modificação a partes gerais do programa, contanto que as novas classes façam parte da hierarquia de herança que o programa processa genericamente.

- As novas classes simplesmente se “encaixam”. As únicas partes de um programa que devem ser alteradas são aquelas que exigem conhecimento direto das novas classes que adicionamos à hierarquia.

- Por exemplo, se ocorre uma extensão da classe Animal para criar a classe Tartaruga (que poderia responder a uma mensagem mover deslizando uma polegada), basta escrever somente a classe Tartaruga e a parte da simulação que instancia um objeto Tartaruga. As partes da simulação que dizem para que cada Animal se mova genericamente podem permanecer as mesmas.

PUC GOIÁS

Polimorfismo

- Até o momento sabe-se que variáveis de superclasse são *concebidas* para referenciar objetos de superclasse e variáveis de subclasse, para referenciar objetos de subclasse.

- Sabe-se também que você não pode tratar um objeto de superclasse como um objeto de subclasse, porque um objeto de superclasse não é um objeto de quaisquer das suas subclasses.

- Por fim, sabe-se que o relacionamento "**é um "** é aplicado somente a partir da parte superior da hierarquia de uma subclasse às suas superclasses diretas (e indiretas), e não vice-versa (isto é, não da parte inferior da hierarquia de uma superclasse às suas subclasses ou subclasses indiretas).

PUC GOIÁS

Polimorfismo

- Entretanto, como você verá mais adiante, outras atribuições são possíveis.

- Existem formas de **invocar um método em um objeto de subclasse por uma referência de superclasse que invoca a funcionalidade da subclasse** — o tipo de objeto referenciado, não o tipo de variável, determina qual método é chamado. Esse exemplo demonstra que um **objeto de uma subclasse pode ser tratado como um objeto da sua superclasse**, permitindo várias manipulações interessantes.

PUC GOIÁS

Polimorfismo Dinâmico - @Override (sobrescrita)

- **@Override**: é uma forma de garantir que você está **sobrescrevendo** um método e não criando um novo.

- Toda linguagem orientada a objetos permite a **sobrescrita** de métodos da superclasse (pai) pela subclasse (filha). Entretanto cada linguagem de programação usa seus próprios meios para lidar com essa sobrescrita.

- O Java optou por usar a notação **@Override** para garantir a segurança no código, entretanto, nada obriga o uso dessa notação.

- Entretanto, em seu não uso, caso você não se lembrar da classe que estendeu a SuperClasse, você na verdade está chamando um método novo, e não sobrescrevendo-o.

PUC GOIÁS

Polimorfismo Dinâmico - @Override (sobrescrita)

```
public class Animal {
    private String Nome;
    private double Peso;

    //GETTERS e SETTERS

     public void mover()
     {
        System.out.println("Animais se movem...");
     }

}
```

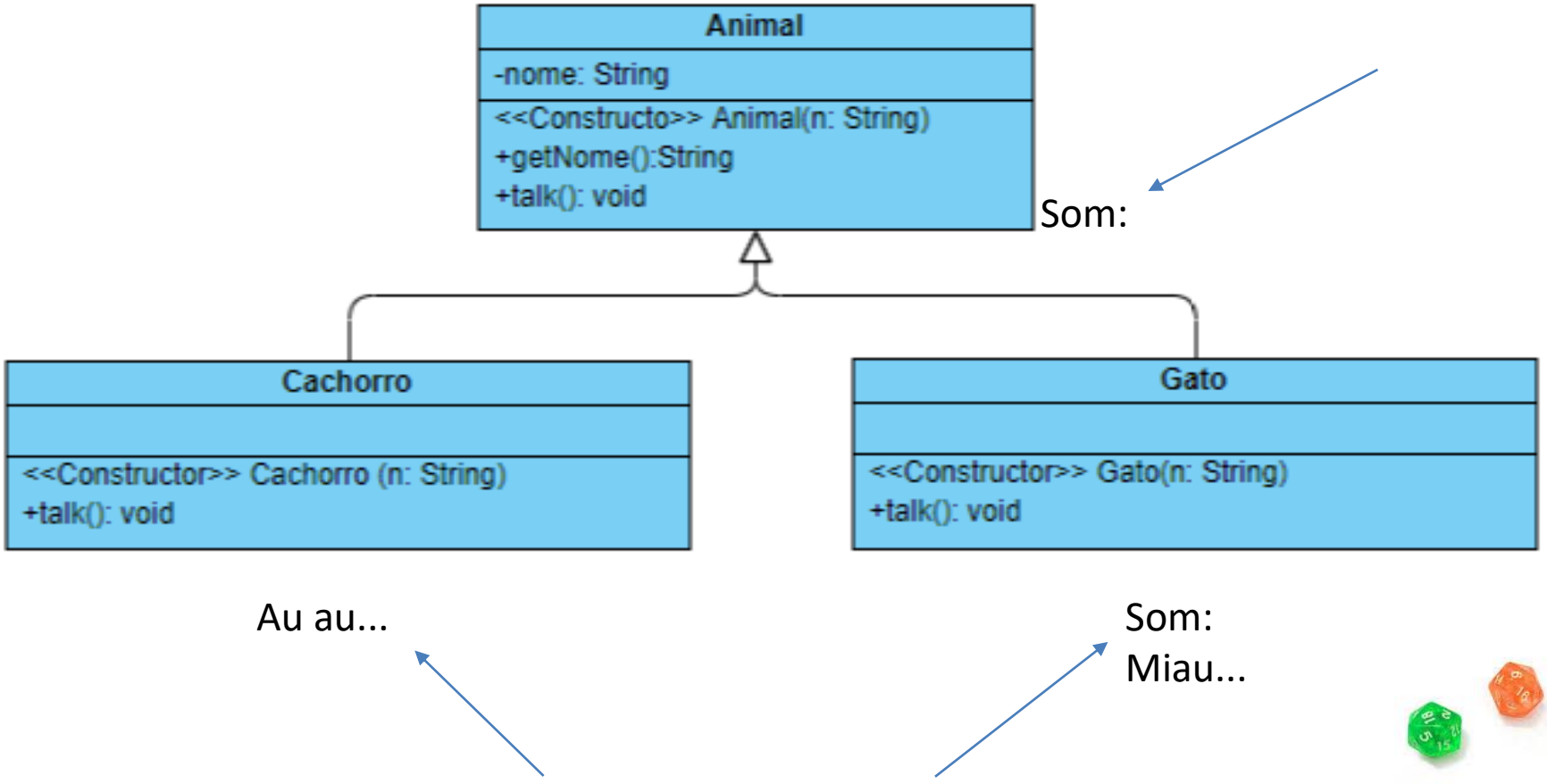
PUC GOIÁS

Polimorfismo Dinâmico - @Override (sobrescrita)

```
public class Cachorro extends Animal{
    private String raca;

//GETTERS e SETTERS

    @Override
    public void mover()
    {
        System.out.println("Corre sobre 4 patas");

    }
}
```

PUC GOIÁS

Exercício Em sala: implemente essas Classes



PUC GOIÁS

Classes e métodos abstratos

- Em Classes Abstratas, *nunca se* pretende criar objetos.
- São classes utilizadas somente como superclasses em hierarquias de herança. Especificamente são chamadas superclasses abstratas. Essas classes não podem ser utilizadas para instanciar objetos, porque classes abstratas são *incompletas*.
- As subclasses devem declarar as “partes ausentes” para que se tornem classes “concretas”, a partir das quais pode-se instanciar objetos.
- O propósito de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim compartilhar um design comum.

PUC GOIÁS

Classes e métodos abstratos

- Cria-se uma classe abstrata declarando-a com a palavra-chave **abstract**. Uma classe abstrata normalmente contém um ou mais métodos abstratos.

- Um método abstrato é um *método de instância* com a palavra-chave abstract na sua declaração, como em:

```
public abstract void draw( ); // método abstrato
```

- Métodos abstratos ***não*** fornecem implementações.

- Uma classe que contém *quaisquer* métodos abstratos deve ser expressamente declarada abstract, mesmo que ela contenha alguns métodos concretos (não abstratos).

PUC GOIÁS

Classes e métodos abstratos

- Cada subclasse concreta de uma superclasse abstrata **também deve fornecer implementações concretas de cada um dos métodos abstratos da superclasse**.

- Os construtores e métodos **static não podem ser declarados abstract**.

- Os construtores *não* são herdados, portanto um construtor abstract nunca seria implementado.

- Embora métodos **não private static** sejam herdados, **eles não podem ser sobrescritos**. Como os métodos abstract devem ser sobrescritos para que possam processar objetos com base em seus tipos, não faria sentido declarar um método **static** como abstract.

PUC GOIÁS

Classes e métodos abstratos

- Embora não seja possível instanciar objetos de superclasses abstratas, é possível **utilizar superclasses abstratas para declarar variáveis que podem conter referências a objetos de *qualquer* classe concreta *derivados dessas* superclasses abstratas**.

- Essas variáveis podem ser utilizadas para manipular objetos da subclasse *polimorficamente.*

- Pode-se também utilizar nomes abstratos de superclasse para invocar métodos **static** declarados nessas superclasses abstratas.

- Uma classe abstrata não precisa necessariamente ter um método abstrato.

PUC GOIÁS

Classes e métodos abstratos: EXEMPLO

Usamos a palavra chave **abstract para impedir que ela possa ser instanciada**. Esse é o efeito direto de se usar o modificador abstract na declaração de uma classe:

```
public abstract class Funcionario {

    protected double salario;

    public double getBonificacao() {
        return this.salario * 1.2;
    }

    // outros atributos e métodos comuns a todos Funcionarios

}
```

E, no meio de um código:

Funcionario f = new Funcionario(); // não compila!!!

PUC GOIÁS

Classes e métodos abstratos: EXEMPLO

- Mas qual é a real vantagem de uma classe abstrata? Poderíamos ter feito isto com uma herança comum. Por enquanto, a única diferença é que não podemos instanciar um objeto do tipo Funcionario, que já é de grande valia, dando mais consistência ao sistema.

- Fique claro que a nossa decisão de transformar Funcionario em uma classe abstrata **dependeu do nosso domínio**. Pode ser que, em um sistema com classes similares, faça sentido que uma classe análoga a Funcionario seja concreta.

- Se ela não pode ser instanciada, para que serve? Serve para o polimorfismo e herança dos atributos e métodos, que são recursos muito poderosos

PUC GOIÁS

Classes e métodos abstratos

```
public abstract class Funcionario {

    protected double salario;

    public double getBonificacao() {
        return this.salario * 1.2;
    }

    // outros atributos e métodos comuns a todos Funcionarios

}
```

Repare que não colocamos o corpo do método e usamos a palavra chave abstract para definir o mesmo. Por que não colocar corpo algum? Porque esse método nunca vai ser chamado, sempre que alguém chamar o método **getBonificacao**, vai cair em uma das suas filhas, que realmente escreveram o método.

```
public abstract class Funcionario {
  protected double salario;

  public abstract double getBonificacao();

   //outros atributos e métodos
}
```

Qualquer classe que estender a classe Funcionario será obrigada a reescrever este método, tornando-o "concreto". Se não reescreverem esse método, **um erro de compilação ocorrerá.**

PUC GOIÁS

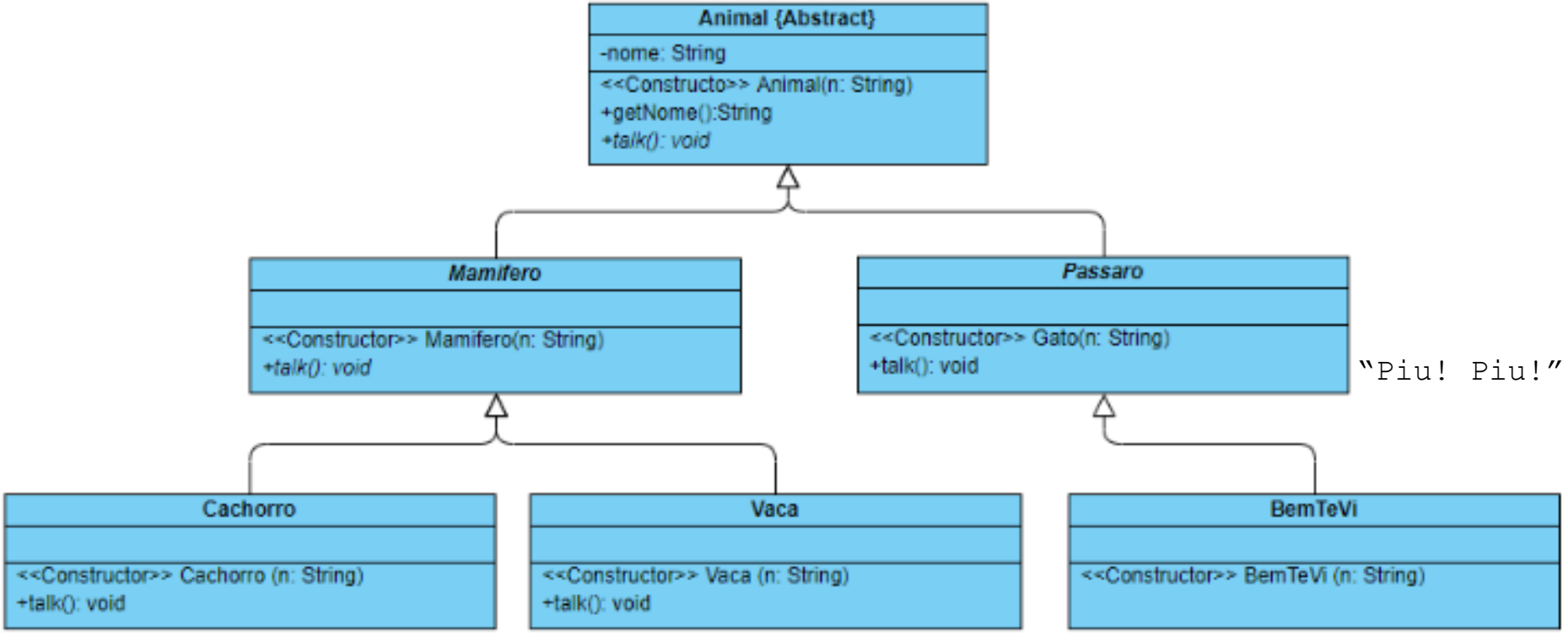
Classes e métodos abstratos

```
public abstract class Funcionario {
  protected double salario;

  public abstract double getBonificacao();

   //outros atributos e métodos
}
```

```
public class Gerente extends Funcionario {

    public double getBonificacao() {
        return this.salario * 1.4 + 1000;
    }
}
```

PUC GOIÁS

Exercício Em sala: implemente essas Classes



PUC GOIÁS

BOAS PRÁTICAS DE PROGRAMAÇÃO

Observação de engenharia de software 10.1

O polimorfismo permite-lhe tratar as generalidades e deixar que o ambiente de tempo de execução trate as especificidades. Você pode instruir objetos a se comportarem de maneiras apropriadas para esses objetos, sem nem mesmo conhecer seus tipos específicos, contanto que os objetos pertençam à mesma hierarquia de herança.

Observação de engenharia de software 10.2

O polimorfismo promove a extensibilidade: o software que invoca o comportamento polimórfico é independente dos tipos de objeto para os quais as mensagens são enviadas. Novos tipos de objeto que podem responder a chamadas de método existentes podem ser incorporados a um sistema sem modificar o sistema básico. Somente o código de cliente que instancia os novos objetos deve ser modificado para acomodar os novos tipos.

PUC GOIÁS

BOAS PRÁTICAS DE PROGRAMAÇÃO

Observação de engenharia de software 10.4

Uma classe abstrata declara atributos e comportamentos comuns (ambos abstratos e concretos) das várias classes em uma hierarquia de classes. Em geral, uma classe abstrata contém um ou mais métodos abstratos que as subclasses devem sobrescrever se elas precisarem ser concretas. Variáveis de instância e métodos concretos de uma classe abstrata estão sujeitos às regras normais da herança.

Erro comum de programação 10.1

Tentar instanciar um objeto de uma classe abstrata é um erro de compilação.

Erro comum de programação 10.2

Falha para implementar os métodos abstratos de uma superclasse em uma subclasse é um erro de compilação, a menos que a subclasse também seja declarada `abstract`.

PUC GOIÁS

Referência Bibliográfica Principal

• DEVMEDIA. Disponível em https://www.devmedia.com.br. Acessado em Julho de 2019.

• DEITEL, Harvey M. Java: Como Programar – 10 ed. Cap 10. 2015.

• CAELUM. Disponível em https://www.caelum.com.br/apostila-java-orientacao-objetos/classes-abstratas/. Acessado em Agosto de 2019.