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

Herança

PUC GOIÁS

Agenda:

- Importação ***static***
- Modificador ***final*** (constantes)
- Acesso de pacote (nota)
- Herança
  - Subclasse e Superclasse
  - Superclasse direta e indireta
  - Java e suporte a herança única
  - Relacionamento “é um” vs “tem um”
  - Modificador de acesso ***protected***
- Exercício: implementação
- *Downcasting*
- Exemplo de *downcasting*

PUC GOIÁS

Importação Static

Permite importar os membros static de uma interface ou classe para que você possa acessá-los por meio dos nomes não qualificados na sua classe — isto é, um ponto (.) e o nome da classe não são necessários ao usar um membro static importado.

- *Importação Static Simples*

```
import static nomeDoPacote.NomeDaClasse.nomeDoMembroStatic;
```

- *Importação Static por Demanda*

```
import static nomeDoPacote.NomeDaClasse.*;
```

PUC GOIÁS

Importação Static

```
1   // Figura 8.14: StaticImportTest.java
2   // Importação static dos métodos da classe Math.
3   import static java.lang.Math.*;
4
5   public class StaticImportTest
6   {
7      public static void main(String[] args)
8      {
9         System.out.printf("sqrt(900.0) = %.1f%n", sqrt(900.0));
10        System.out.printf("ceil(-9.8) = %.1f%n", ceil(-9.8));
11        System.out.printf("E = %f%n", E);
12        System.out.printf("PI = %f%n", PI);
13     }
14  } // fim da classe StaticImportTest
```

Teste sem *static*...

PUC GOIÁS

Constantes: *final*

UItilizar a palavra-chave ***final*** para especificar o fato de que uma variável não é modificável (isto é, é uma constante). Essas variáveis podem ser inicializadas quando elas são declaradas. Se não forem, elas devem ser inicializadas em cada construtor da classe.

- *Exemplo*

private final int INCREMENT;

```
public class JavaApplication17 {

    static final int INC=0;

    public static void main(String[] args) {

        System.out.printf("INC = %d%n", INC);

    }

}
```

PUC GOIÁS

NOTA – Acesso de pacote

Se nenhum modificador de acesso (**public**, **protected** ou **private**) for especificado para um método ou variável quando esse método ou variável é declarado em uma classe, o método ou variável será considerado como tendo acesso de pacote. Em um programa que consiste em uma declaração de classe, isso não tem nenhum efeito específico. Entretanto, se um programa utilizar múltiplas classes no mesmo pacote (isto é, um grupo de classes relacionadas), essas classes poderão acessar diretamente os membros de acesso de pacote de outras classes por meio de referências a objetos das classes apropriadas,
ou no caso de membros **static**, por meio do nome de classe. O acesso de pacote é raramente usado.

PUC GOIÁS

Herança

Ao criar uma classe, em vez de declarar membros completamente novos, você pode designar que a nova classe, chamada de **subclasse**, deva herdar membros de uma classe existente, que é chamada de **superclasse**. (A linguagem de programação C++ refere-se à superclasse como a **classe básica** e a subclasse como a **classe derivada**.) Uma subclasse pode tornar-se uma superclasse para futuras subclasses.

Uma subclasse pode adicionar seus próprios campos e métodos. Portanto, ela é mais específica que sua superclasse e representa um grupo especializado de objetos. A subclasse exibe os comportamentos da superclasse e pode modificá-los de modo que eles operem adequadamente para a subclasse. É por isso que a herança é às vezes chamada **especialização**.

PUC GOIÁS

Herança

- A **superclasse direta** é aquela a partir da qual a subclasse herda explicitamente.
- Uma **superclasse indireta** é qualquer classe acima da superclasse direta na hierarquia de classes, que define os relacionamentos de herança entre classes
- No Java, a **hierarquia de classes** inicia com a **classe Object** (no pacote java.lang), da qual toda classe em Java direta ou indiretamente estende (ou “herda de”).
- O **Java só suporta herança única**, na qual cada classe é derivada exatamente de uma superclasse direta. Ao contrário de C++, o Java não suporta herança múltipla, que ocorre quando uma classe é derivada de mais de uma superclasse direta.
- Entretanto, Java pode usar **interfaces** para obter muitos dos benefícios da **herança múltipla** e, ao mesmo tempo, evitar os problemas associados.

PUC GOIÁS

Herança

- Distinguimos entre o **relacionamento é um** e o **relacionamento tem um**, em que "**é um"** representa a herança.

- Em um **relacionamento é um** (herança), um objeto de uma subclasse também pode ser tratado como um objeto da superclasse — por exemplo, um carro é um veículo

- Em um **relacionamento tem um** (associação), um objeto contém referências como membros a outros objetos — por exemplo, um carro tem um volante (um objeto carro tem uma referência a um objeto volante).

PUC GOIÁS

Herança

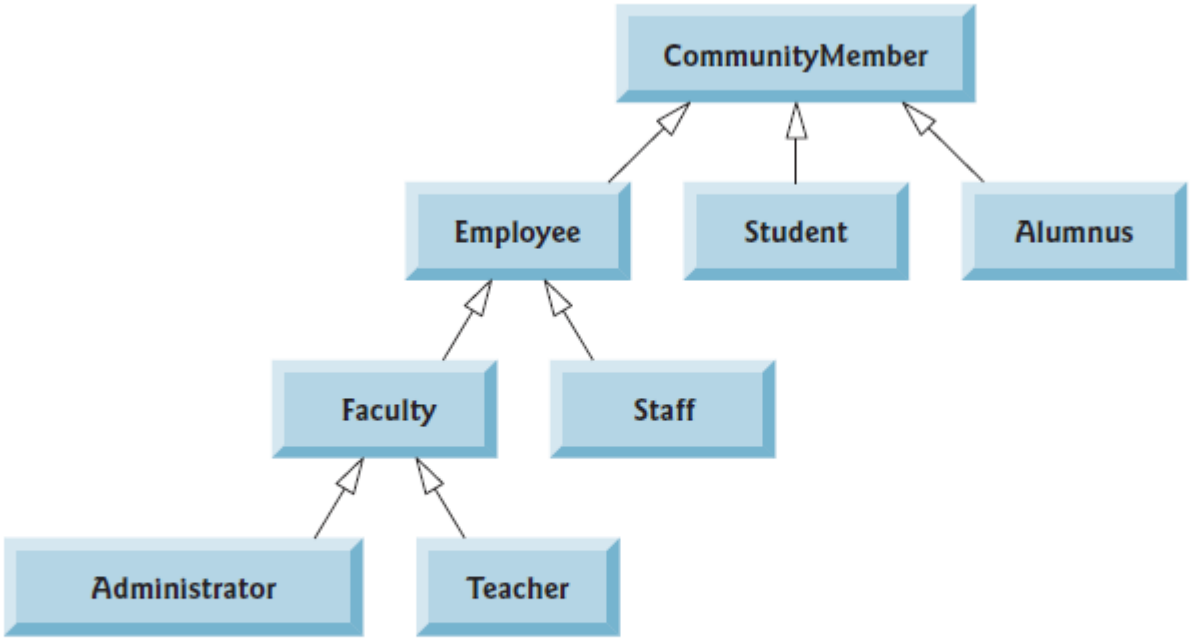


Figura 9.2 | Diagrama de classes UML da hierarquia de herança para `CommunityMembers` universitários.

PUC GOIÁS

Herança

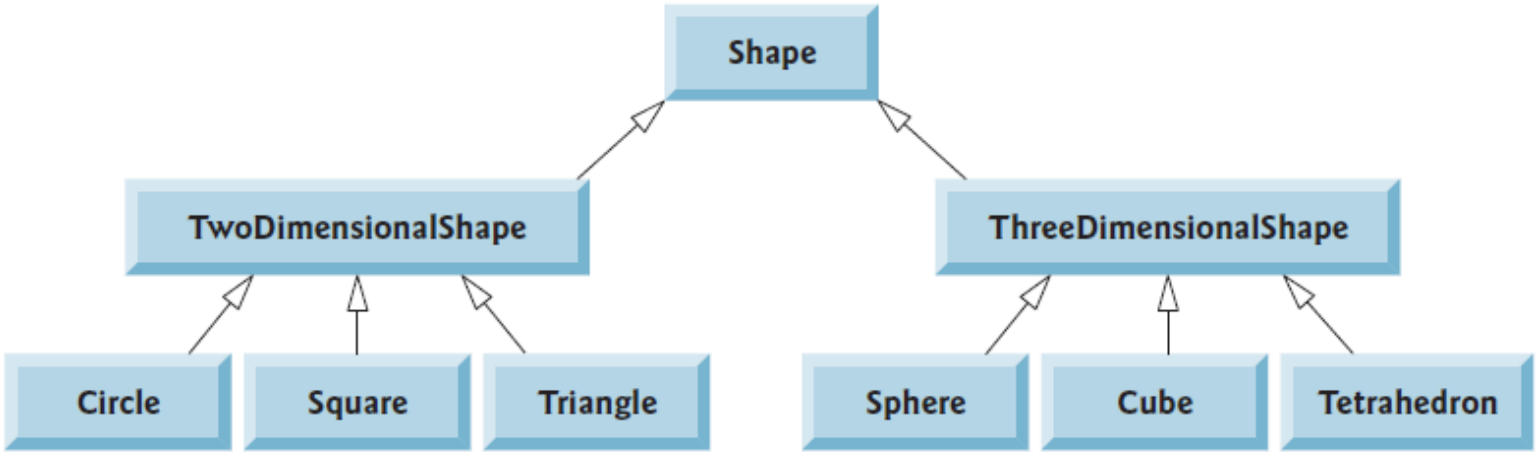


Figura 9.3 | Diagrama de classes UML da hierarquia de herança para Shapes.

PUC GOIÁS

Herança – métodos e atributos Protected

- Os membros ***protected*** de uma superclasse podem ser acessados por membros dessa superclasse, de suas subclasses e de outras classes no mesmo pacote — membros ***protected*** também têm acesso de pacote.

- Todos os membros de superclasse ***public*** e ***protected*** retêm seu modificador de acesso original quando se tornam membros da subclasse — membros ***public*** da superclasse tornam-se membros ***public*** da subclasse, e membros ***protected*** da superclasse tornam-se membros ***protected*** da subclasse.

- Membros ***private*** de uma superclasse **não são acessíveis fora da própria classe**. Em vez disso, eles permanecem ocultos de suas subclasses e só podem ser acessados por meio dos métodos ***public*** ou ***protected*** herdados da superclasse.

PUC GOIÁS

Herança – métodos e atributos Protected

- Os métodos de subclasse podem referir-se a membros ***public*** e ***protected*** herdados da superclasse simplesmente utilizando os nomes de membro.

- Quando um método de subclasse sobrescrever um método de superclasse herdado, a versão superclasse do método pode ser acessada a partir da subclasse, precedendo o nome do método de superclasse com a palavra-chave ***super*** e um **separador de ponto** (.).

PUC GOIÁS

Herança – Exemplo em Sala

```
public class Aluno
 {
   private String nome;
   private int idade;
   private String matricula;

    public Aluno(String nome, int idade, String matricula)
    {
      this.nome=nome;
      this.idade = idade;
      this.matricula= matricula;
    }

   // getter e setters ...
}
```

PUC GOIÁS

Herança – Exemplo em Sala

```
public class Professor
 {

   private String nome;
   private int idade;
   private String materia;

   public Professor(String nome, int idade, String materia)
   {
      this.nome=nome;
      this.idade = idade;
      this.materia= materia;

   }

    // getter e setters ...
}
```

PUC GOIÁS

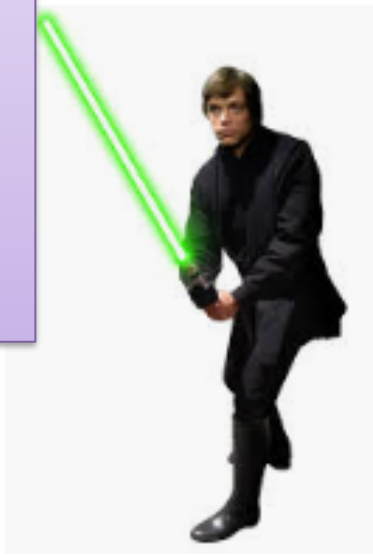
Herança – Exemplo em Sala

```
public static void main (String[] args) {
      Aluno objAluno = new Aluno ("Goku",40,"1009");
     Professor objProfessor = new Professor ("Mestre Kame", 90,
"Kamehamaha");

      System.out.println("Aluno    : " + objAluno.getNome());
      System.out.println("Idade    : " + objAluno.getIdade());
      System.out.println("Matricula: " + objAluno.getMatricula());

      System.out.println("Professor: " + objProfessor.getNome());
      System.out.println("Idade    : " + objProfessor.getIdade());
      System.out.println("Materia  : " + objProfessor.getMateria());

   }
```



PUC GOIÁS

Herança – Exemplo em Sala

Generalização/Especialização

Herança simples: private / public

PUC GOIÁS

Herança – Exemplo em Sala

```
public class Pessoa {
   private String nome;
   private int idade;

   public Pessoa(String nome, int idade)
  {
      this.nome=nome;
      this.idade = idade;
   }

    // getters e setters...
    }
}
```

Herança simples: private / public

PUC GOIÁS

Herança – Exemplo em Sala

```
public class Aluno extends Pessoa{
   private String matricula;

   public Aluno(String nome, int idade, String matricula) {
     super(nome,idade);
     this.matricula= matricula;
   }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
}
```

Herança simples: private / public

PUC GOIÁS

Herança – Exemplo em Sala

```
public class Professor extends Pessoa{
   private String materia;

   public Professor(String nome, int idade, String materia) {
     super(nome,idade);
     this.materia= materia;
   }

    //getters e setters...

}
```

Herança simples: private / public

PUC GOIÁS

Herança – Exemplo

Herança simples: protected/public/private

PUC GOIÁS

Herança – Exemplo

```
public class Pessoa {
   private String nome;
   private int idade;

    public Pessoa(String nome, int idade) {
      this.nome=nome;
      this.idade = idade;
     }
       // getters e setters...

 protected void alteraNome(String nome)
    {
                this.nome=nome;
    }
}
```

Herança simples: protected/public/private

PUC GOIÁS

Herança – Exemplo

```
public class Aluno extends Pessoa{
   private String matricula;

   public Aluno(String nome, int idade, String matricula) {
     super(nome,idade);
     this.matricula= matricula;
   }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
}
```

Herança simples: protected/public/private

PUC GOIÁS

Herança – Exemplo

```
public class Professor extends Pessoa{
  private String materia;

  public Professor(String nome, int idade, String materia) {
    super(nome,idade);
    this.materia= materia;
  }

   //getters e setters...

}
```

Herança simples: protected/public/private

PUC GOIÁS

Herança – Exemplo

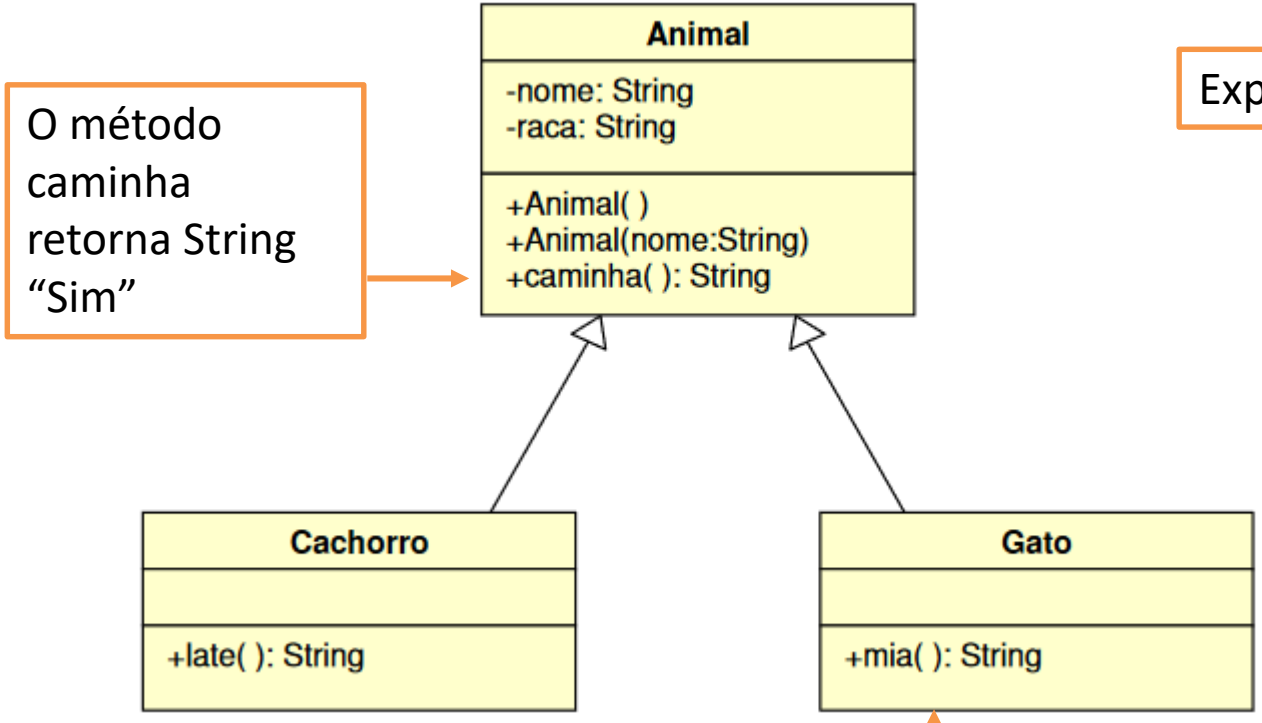
```
public static void main(String[] args) {
        Aluno objAluno = new Aluno ("Goku",40,"1009");
        Professor objProfessor = new Professor ("Mestre Kame", 90,
"Kamehamaha");
        Pessoa objPessoa = new Pessoa("Teste",10);

        System.out.println ("Professor: " + objProfessor.getNome( ) );
        System.out.println ("Idade    : " + objProfessor.getIdade( ) );
        System.out.println ("Materia  : " + objProfessor.getMateria( ) );
        objProfessor.alteraNome("Gohan");
        System.out.println ("Professor: " + objProfessor.getNome( ) );
        System.out.println ("Idade    : " + objProfessor.getIdade( ) );
        System.out.println ("Materia  : " + objProfessor.getMateria( ) );
}
```

Herança simples: protected/public/private

PUC GOIÁS

Exercício: implemente essas Classes



O método caminha retorna String "Sim"

Explicação no próximo slide!

O método late imprime "au au"

O método mia imprime "miau"

PUC GOIÁS

Exercício: implemente essas Classes

- Para a classe Animal, crie os métodos GET/SET protected (apesar de não estarem explicitamente descritos no diagrama).
- Para ambas as classes cachorro e gato, crie construtores com parâmetros.
- Crie os objetos:
  - Cachorro(Fred)
  - Gato(John)
- Fred tem raça Boxer e John tem raça Viralatas
- No programa principal, imprima o nome, a raça e as respectivas “falas” de cada animal.

PUC GOIÁS

Nota sobre Herança: Compatibilidade de Tipos

- Qualquer subclasse é compatível com a sua superclasse

```
Animal cao2 = new Cachorro();

        System.out.println("Cachorro 2:");
        System.out.println("-->Nome:" + cao2.getNome());
        System.out.println("-->Peso:" + cao2.getPeso() + "Kg");
```

- Contudo, a reciproca não é verdadeira

```
Cachorro cao2 = new Animal();

        System.out.println("Cachorro 2:");
        System.out.println("-->Nome:" + cao2.getNome());
        System.out.println("-->Peso:" + cao2.getPeso() + "Kg");
```

PUC GOIÁS

Polimorfismo - Downcasting

- Java não permite a atribuição de uma referência de superclasse a uma variável de subclasse, se a referência da superclasse for convertida explicitamente para o tipo da subclasse.
- Uma referência de superclasse somente pode ser utilizada para invocar os métodos declarados na superclasse — tentativas de invocar métodos somente de subclasse por meio de uma referência de superclasse resultam em erros de compilação.
- Se um programa precisar realizar uma operação específica na subclasse em um objeto de subclasse referenciado por uma variável de superclasse, o programa deverá primeiro fazer uma coerção (cast) da referência de superclasse para uma referência de subclasse por meio de uma técnica conhecida como **downcasting**. Isso permite ao programa invocar métodos de subclasse que não estão na superclasse.

Observação de engenharia de software 10.3

Embora seja permitido, você geralmente deve evitar o downcasting.

PUC GOIÁS

Polimorfismo – Downcasting – Exemplo Animais

```
// Downcasting 1
    if (Cachorro.class.isInstance(animal)) {
      Cachorro cao2 = Cachorro.class.cast(animal);
      cao2.late();
    }

// Downcasting2
    if (Cachorro.class.isInstance(animal)) {
      Cachorro cao3 = (Cachorro)animal;
      cao3.late();
    }

// Downcasting 3
    Cachorro cao4 = (Cachorro)animal;
    cao4.late();
```

PUC GOIÁS

BOAS PRÁTICAS DE PROGRAMAÇÃO

Erro comum de programação 8.7

Um erro de compilação ocorre se um programa tentar importar métodos `static` que têm a mesma assinatura ou campos `static` que têm o mesmo nome proveniente de duas ou mais classes.

Observação de engenharia de software 8.11

Declarar uma variável de instância como `final` ajuda a impor o princípio do menor privilégio. Se uma variável de instância não deve ser modificada, declare-a como `final` para evitar modificação. Por exemplo, na Figura 8.8, as variáveis de instância `firstName`, `lastName`, `birthDate` e `hireDate` nunca são modificadas depois que elas são inicializadas, então elas devem ser declaradas `final`. Vamos aplicar essa prática em todos os programas daqui para a frente. Veremos os benefícios adicionais de `final` no Capítulo 23, "Concorrência".

Erro comum de programação 8.8

Tentar modificar uma variável de instância `final` depois que é ela inicializada é um erro de compilação.

PUC GOIÁS

BOAS PRÁTICAS DE PROGRAMAÇÃO

Dica de prevenção de erro 8.5

Tentativas de modificar uma variável de instância `final` são capturadas em tempo de compilação em vez de causarem erros em tempo de execução. Sempre é preferível retirar bugs em tempo de compilação, se possível, em vez de permitir que passem para o tempo de execução (onde experiências descobriram que o reparo é frequentemente muito mais caro).

Observação de engenharia de software 8.12

Um campo `final` também deve ser declarado `static` se ele for inicializado na sua declaração para um valor que é o mesmo para todos os objetos da classe. Após essa inicialização, seu valor nunca pode mudar. Portanto, não precisamos de uma cópia separada do campo para cada objeto da classe. Criar o campo `static` permite que todos os objetos da classe compartilhem o campo `final`.

Observação de engenharia de software 9.1

Os métodos de uma subclasse não conseguem acessar diretamente os membros `private` de sua superclasse. Uma subclasse pode alterar o estado de variáveis de instância `private` da superclasse somente por meio de métodos não `private` fornecidos na superclasse e herdados pela subclasse.

PUC GOIÁS

BOAS PRÁTICAS DE PROGRAMAÇÃO

Observação de engenharia de software 9.2

Declarar variáveis de instância `private` ajuda você a testar, depurar e modificar sistemas corretamente. Se uma subclasse pudesse acessar variáveis de instância `private` da sua superclasse, classes que herdam dessa subclasse também poderiam acessar as variáveis de instância. Isso propagaria acesso ao que devem ser variáveis de instância `private`, e os benefícios do ocultamento de informações seriam perdidos.

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 09. 2015.