Universidade Federal do Ceará
Centro de Ciências/Departamento de Computação
Código da Disciplina: CK0084
Professor: Ismayle de Sousa Santos

Sistemas de Informações e Banco de Dados

Herança, Interfaces e Classes Abstratas

IsmayleSantos
ismayle@ufc.br
@IsmayleSantos

Hoje aprenderemos sobre ...

- Programação Orientada a Objetos
  - Associações entre classes
  - Herança
  - Hierarquia de classe
  - Polimorfismo
  - Interface
  - Classes abstratas

Relembrando...

Relembrando POO ...

- Em orientação a objeto, uma classe é uma estrutura que abstrai um conjunto de objetos com características similares
  - A classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar
- Classes não são diretamente suportadas em todas as linguagens, e são necessárias para que uma linguagem seja orientada a objetos
- A programação orientada a objeto tem como pilares: herança, polimorfismo e encapsulamento

Encapsulamento

- Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, o mais isolado possível
- A idéia é tornar o software mais flexível, fácil de modificar e de criar novas implementações
- O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe
- Não devemos permitir o acesso público aos membros, exceto em caso de ser constantes

Encapsulamento

- Os dados contidos em um objeto somente poderão ser acessados e/ou modificados através de seus métodos
  - Dessa forma não é possível alterar os dados diretamente, somente através de métodos definidos no objeto
  - Sempre usamos private, a menos que tenhamos um bom motivo para deixá-lo com outro nível de acesso

Encapsulamento

- Para se ter acesso a algum atributo ou método que esteja encapsulado utiliza-se o conceito de get e set
  - Com **SET** é feita uma atribuição a algum atributo, ou seja, define, diz o valor que algum atributo deve ter
  - Com **GET** é possível recuperar esse valor
- Exemplo: Considerando um atributo "double raio"
  - **public void setRaio(double novoRaio)**
    - método que altera o valor de raio
  - **public double getRaio()**
    - método que retornar o valor de raio

Encapsulamento

- O encapsulamento assegura que toda a comunicação com o objeto seja realizada por um conjunto pré-definido de operações
- O encapsulamento facilita as mudanças, visto que os objetos são isolados uns dos outros, reduzindo desta forma o acoplamento
- Além disso o encapsulamento facilita a manutenção de classes, bem como, garante a integridade dos atributos de um objeto em um determinado instante

Associação entre Classes

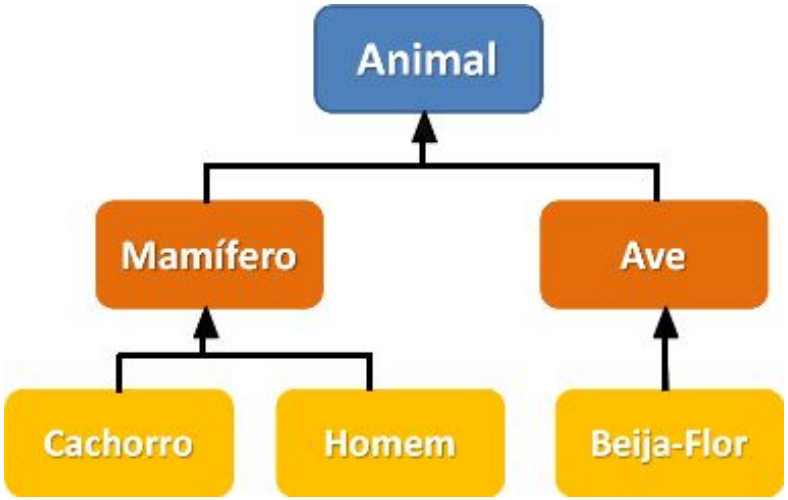
- Uma associação define que os objetos de uma classe são conectados a objetos de outra classe
  - ocorre quando uma classe possui atributos do tipo de outra classe
- Existe uma associação entre duas classes se uma instância de uma classe deve conhecer sobre a existência da outra de modo a realizar seu trabalho

- ***Classe Cliente***
- ***Classe Ordem Pedido***

Herança!

Introdução à Herança ...

- A herança é uma das características primárias e é um dos principais pilares da orientação a objetos
- A herança pode diminuir a quantidade de códigos escrito no início do desenvolvimento do projeto



Qualquer linguagem orientada a objetos possui herança!

Introdução à Herança ...

- Na programação orientada a objetos, o sistema é modelado usando objetos que são criadas usando uma classe
  - Uma classe é um modelo ou uma descrição para criar um objeto
    - Serve para Instanciação de objetos

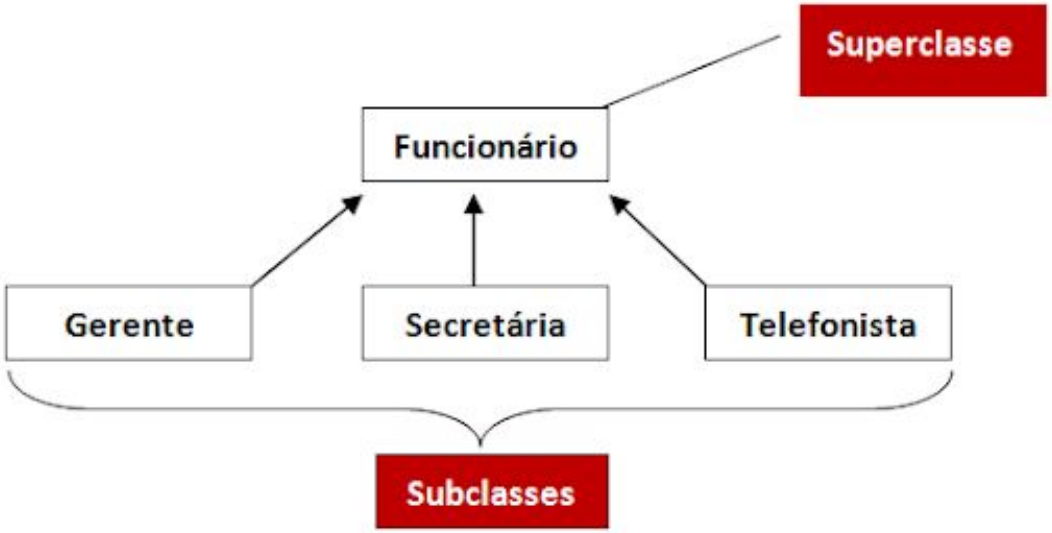
- A herança permite o uso de propriedades e métodos de uma classe já existente ao invés de implementar um programa do zero

O que é Herança?

- É o mecanismo para expressar a similaridade entre Classes, simplificando a definição de classes iguais que já foram definidas
- Ou ainda, uma maneira de reutilizar código à medida que podemos aproveitar os atributos e métodos de classes já existentes para gerar novas classes mais específicas que aproveitarão os recursos da classe "hierarquicamente superior"

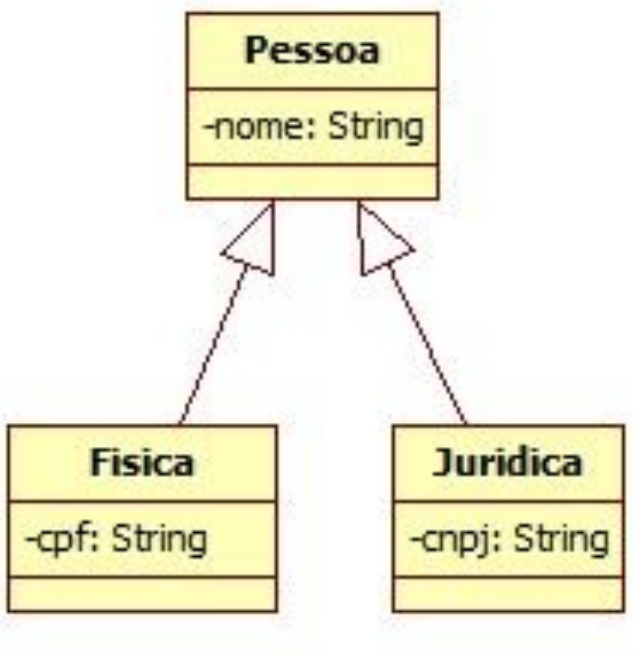
O que é uma Superclasse e Subclasse?

- É chamada de Subclasse uma classe que herda os membros de uma classe superior à ela, sendo ela a Superclasse
  - Superclasse - Fornece membros a outras classes
  - Subclasse - Herda membros da subclasse



Exemplo de Herança

- A classe Pessoa é mais genérica que a classe Física e Jurídica, mas semanticamente representa "é um tipo" ou "é do tipo" pessoa



Herança é a capacidade de uma subclasse de ter acesso às propriedades da superclasse a ela relacionada!

Extends e sua Sintaxe

- A palavra reservada **extends** é usada para indicar que a classe herda (estende) de outra classe:
  - Mecanismo para definição de herança e subtipos
  - A subclasse herda todos os atributos e métodos que a superclasse possuir
  - Subclasse é uma derivação, um subtipo, uma extensão da superclasse

```
class Subclasse extends Superclasse {
   /* ... */
}
```

- ***Herança (Classe Pessoa)***

O que é Superclasse?

- Em Herança, a classe existente a partir da qual as novas classes são derivadas é conhecida como Superclasse
  - Superclasse direta - É a superclasse a qual a subclasse herda explicitamente seus objetos
  - Superclasse indireta- É qualquer superclasse acima da classe direta na hierarquia de classe
- Objetos de superclasse não podem ser tratados como objetos de suas subclasses

Exemplo de Superclasse

- Todos os carros são veículos, mas nem todos os veículos são carros

O que é Subclasse?

- Uma subclasse tem seus próprios métodos e classes
  - É mais específica que a superclasse
- Normalmente, uma subclasse adiciona os seus próprios atributos e métodos ao comportamento da superclasse
  - Representa um grupo mais especializado de objetos
- Possui comportamentos da superclasse mais os adicionais específicos a ela
- Uma subclasse também pode vir a ser uma superclasse

Exemplo de Subclasse

- Há uma superclasse veículos e duas subclasses: passeio e caminhão

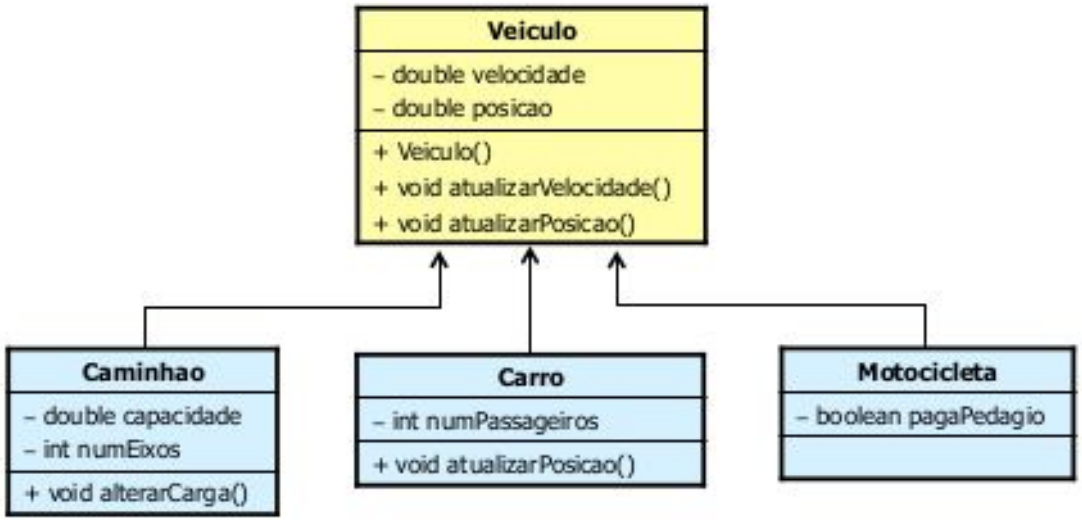
Atenção! Caminhão não pode ser subclasse de Veiculo e também subclasse de uma outra classe Frota, por exemplo. Isso seria herança múltipla ...

Subclasse

- As novas classes serão subclasses da classe Veiculo, por exemplo
- A construção de classes tendo como base uma outra classe permite construir uma hierarquia de classes
- Numa hierarquia de classes, a classe mais geral (superclasse) serve de base para a construção de classes mais específicas (subclasses)

Subclasses e Comportamento

- As subclasses herdam os campos e métodos públicos de sua superclasse, que por sua vez, pode ter herdado campos e métodos de outras superclasses
- Em Java, uma classe pode ter apenas uma superclasse imediata, herança simples



Herança Simples e Herança Múltipla

- Em herança simples um objeto herda características somente de uma superclasse
  - Uma classe pode ter muitas subclasses
  - Uma classe só pode ter uma superclasse
- Em herança múltipla uma classe é derivada de mais de uma superclasse direta

Java não suporta herança múltipla!

O que é Hierarquia de Classe?

- A hierarquia de classes em POO é uma hierarquia de especialização, pelo que uma subclasse de uma dada classe é uma extensão, refinamento ou especialização desta
- Hierarquia simples é uma classe derivada de uma superclasse direta

- ***Hierarquia de Classes***

Class Object

- Toda classe Java estende ("herda de") direta ou indiretamente a classe Object
  - Todas as outras classes herdam (ou estendem) direta ou indiretamente a partir da classe Object, mesmo que não seja definido explicitamente
- A Class Object define um construtor e 11 métodos e não possui atributos

Métodos da classe Object	
clone()	*getclass()*
equals()	*hashCode()*
finalize()	*notify(), notifyAll()*
toString()	*wait() – 3 versões*

Class Object - toString()

- Retorna a representação do objeto que o invocou em formato de string
- A implementação padrão retorna os nomes do pacote ou da classe, seguido pela representação em hexadecimal do valor retornado pelo método hashCode()
- É recomendado que todas as subclasses subscrevam este método
- Pode ser utilizado para substituir o método print()

Redefinição de Métodos

- Métodos herdados de uma superclasse podem ser redefinidos na subclasse
- A redefinição é uma nova implementação para o método herdado, específica para a subclasse.
- Lembrando: a combinação do nome de um método com os tipos de sua lista de parâmetros é conhecida como assinatura do método
- Para a redefinição, o método da subclasse deve ter a mesma assinatura do método herdado da superclasse

Redefinição de Métodos

- O tipo do valor de retorno (embora não sendo parte da assinatura) também deve ser o mesmo
- Se os métodos herdados da superclasse forem definidos na subclasse com assinaturas diferentes haverá a sobrecarga e não a redefinição dos métodos

- ***Redefinição de Métodos***

Interfaces

Interfaces

- A interface é um contrato que quando assumido por uma classe deve ser implementado
  - Separa o contrato da implementação

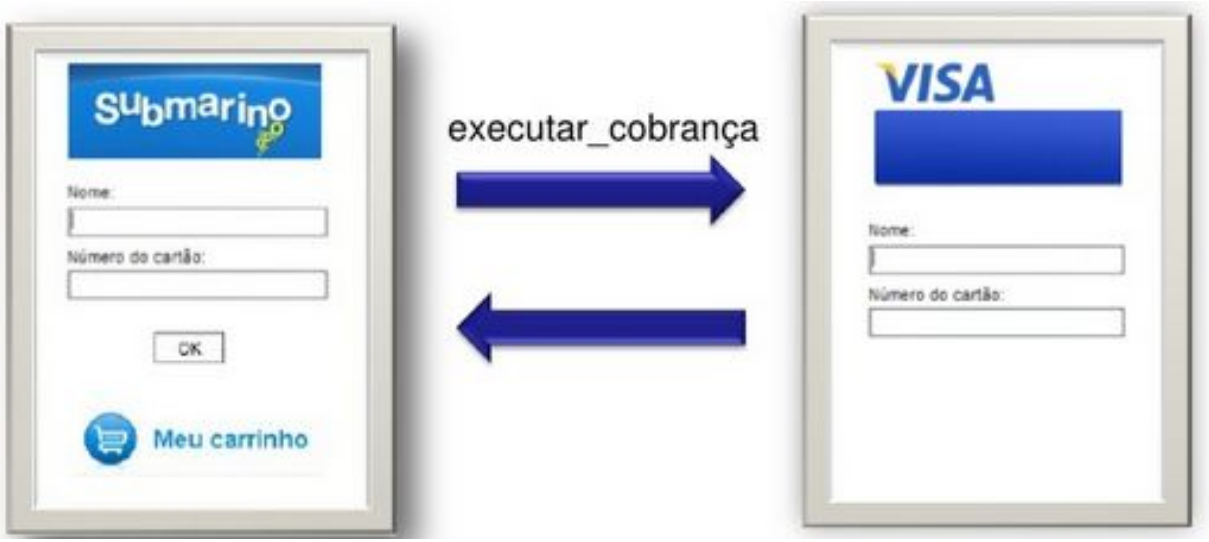
Uma interface nada mais é do que um bloco de código definindo um tipo e os métodos e atributos que esse tipo deve possuir

Interfaces

- A interface não contém nenhum código de implementação, apenas assinaturas de métodos e/ou atributos que devem ter seu código implementado nas classes que "chamar" essa interface
  - os atributos dela são públicos e finais (constantes)
- A Interface define um padrão para especificação do comportamento de classes
- Uma classe pode implementar várias interfaces, mas pode ter apenas uma superclasse

Interfaces

- As interfaces estabelecem as mensagens que podem ser trocadas entre os componentes de software e ocultam os detalhes de implementação



Interfaces

- Uma interface é definida através da palavra reservada interface
  - [public] interface B
- Para uma classe implementar uma interface é usada a palavra implements
  - public class nomeClasse implements nomeInterface
- Interfaces admitem apenas os níveis de acesso public e default
- As classes que forem implementar uma interface terão de adicionar todos os métodos da interface ou se transformar em uma classe abstrata

Exemplo do Uso de Interfaces

Para entendermos melhor, vamos codificar um exemplo de uma interface.

Classes Abstratas

Classes Abstratas

- É um tipo de classe especial que não pode ser instanciada, apenas herdada
  - Uma classe abstrata não pode ter um objeto criado a partir de sua instanciação
  - Possibilita herança de código preservando comportamento (semântica)
- Uma classe abstrata nada mais é do que uma especificação conceitual para outras classes

Classes Abstratas

- Para ter um objeto de uma classe abstrata é necessário criar uma classe mais especializada herdando dela e então instanciar essa nova classe
- Uma classe abstrata pode conter métodos abstratos que as classes que irão estendê-la devem implementar
- Diz o que deve ter a subclasse, mas não diz como
- Os métodos abstratos estão presentes somente em classes abstratas, e são aqueles que não possuem implementação

Diferença entre Interfaces e Classes Abstratas

- Classes abstratas podem conter métodos não-abstratos, isto é, que contêm implementação e que podem ser herdados e utilizados por instâncias das subclasses
- As interfaces não podem conter nenhum método com implementação, todos os seus métodos são implicitamente abstract e public e não possuem corpo
- Os modificadores public e abstract podem ser omitidos sem qualquer efeito colateral

Diferença entre Interfaces e Classes Abstratas

- Se existirem campos (atributos) nas interfaces, eles serão implicitamente considerados public, static e final, isto é, **constantes públicas**, devendo, portanto, ser inicializados na sua declaração
- Se uma classe abstrata contiver apenas métodos abstratos, então, ela pode ser criada como uma interface
    - Terá o mesmo propósito de determinar um comportamento padrão que deve ser apresentado por todas as classes que a implementa

Diferença entre Interfaces e Classes Abstratas

- A diferença essencial entre classes abstratas e interfaces em Java é que uma subclasse somente pode herdar de uma única classe (abstrata ou não), enquanto qualquer classe pode implementar várias interfaces simultaneamente
- Interfaces são, portanto, um mecanismo simplificado de implementação de "herança múltipla" em Java, que possibilita que mais de uma interface determine os métodos que uma classe herdeira deve implementar

Exemplo do Uso de Interfaces

Para entendermos melhor, vamos codificar um exemplo com classe abstrata

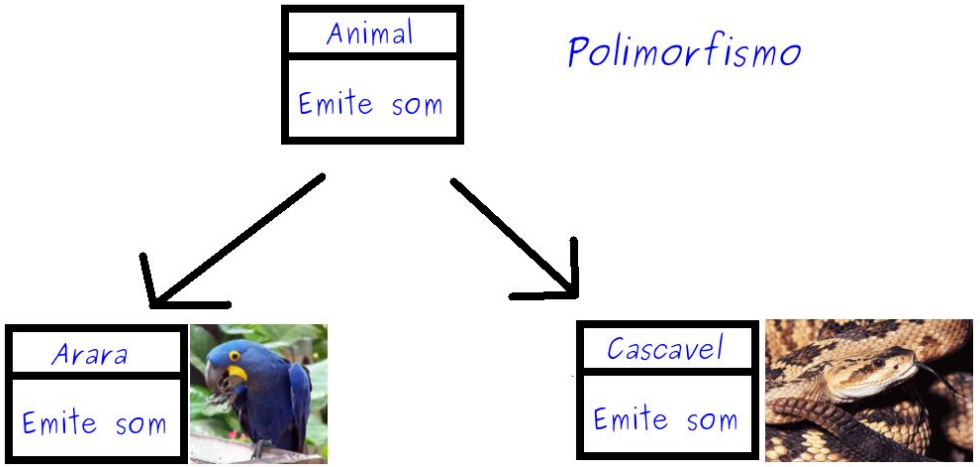
Polimorfismo

O que é Polimorfismo?

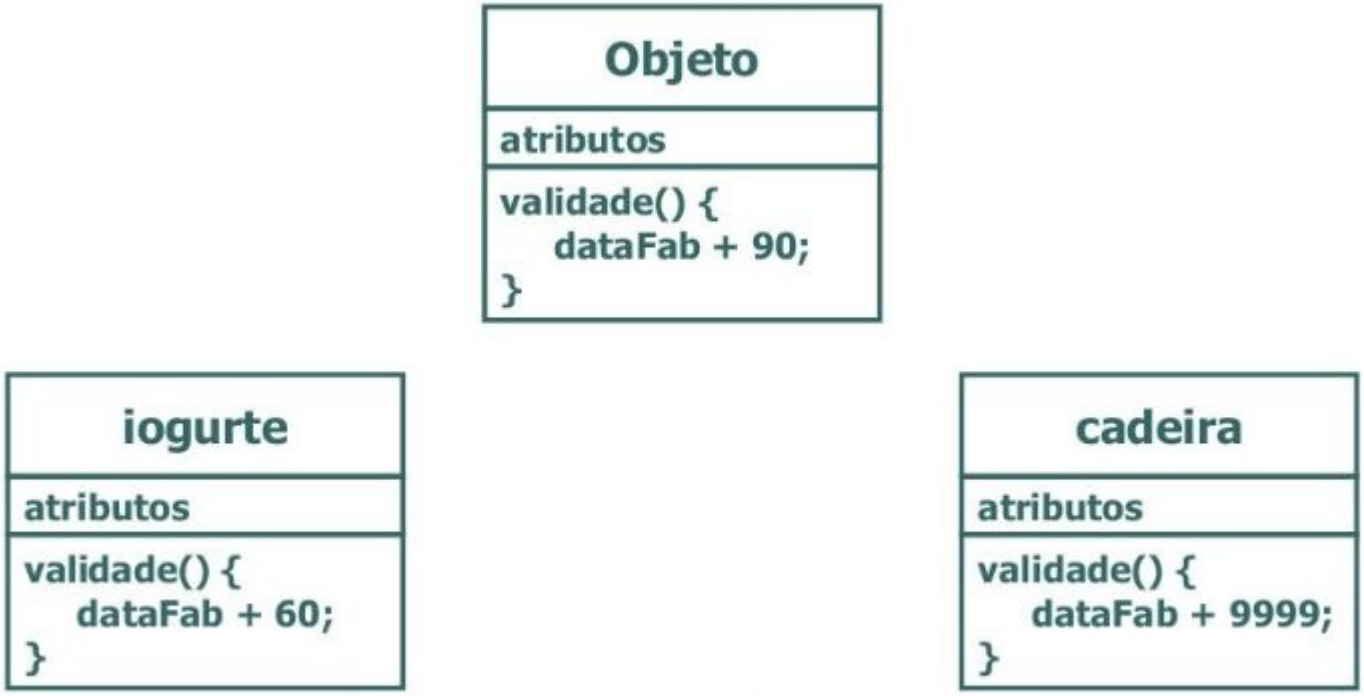
- Polimorfismo é o princípio pelo qual duas ou mais classes derivadas de uma mesma superclasse podem invocar métodos que têm a **mesma assinatura, mas comportamentos distintos**, especializados para cada classe derivada, usando uma referência a um objeto do tipo da superclasse
- O polimorfismo existe como decorrência da hierarquia de classes porque uma variável de referência para um objeto de uma superclasse pode ser usado também como referência para um objeto da subclasse

O que é Polimorfismo?

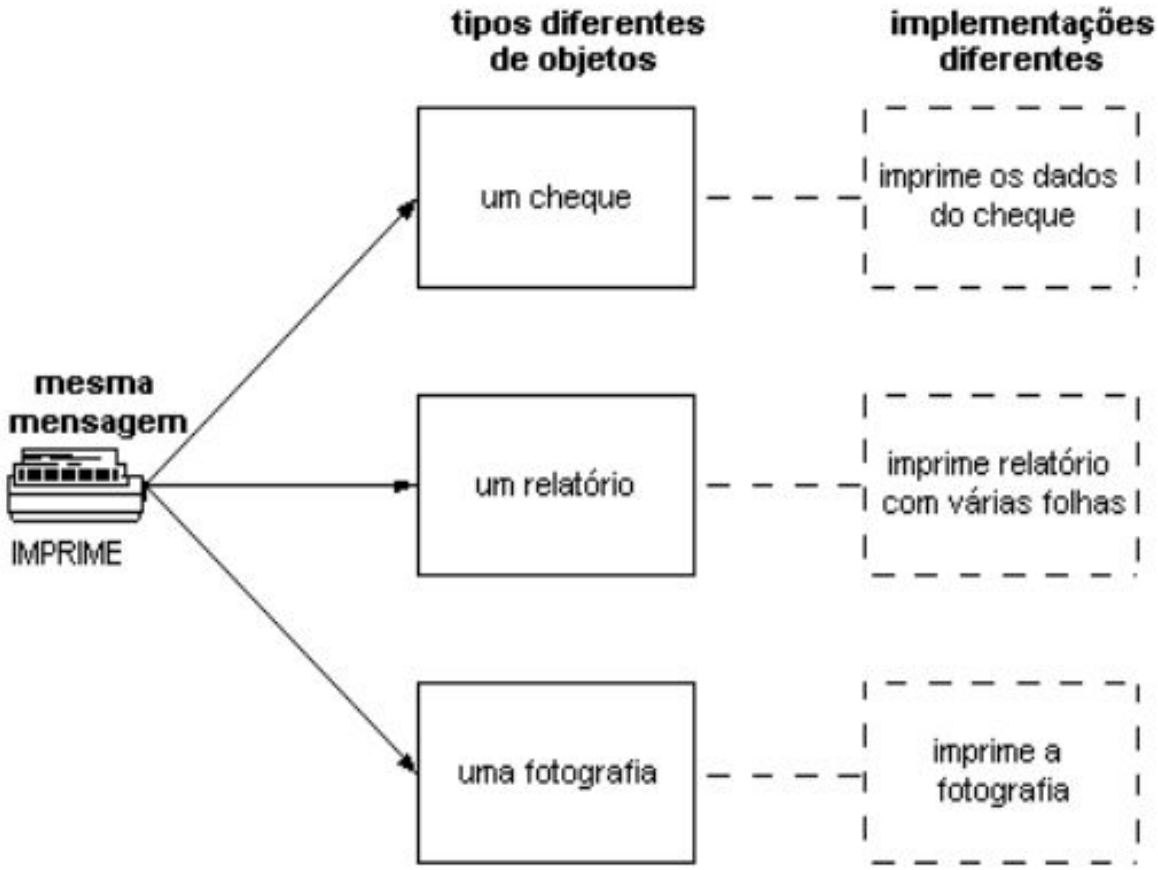
- Significa que variáveis podem referenciar mais do que um tipo
  - Poli = várias e Morfos = formas
- É a característica única de linguagens orientadas a objetos que **permite que diferentes objetos respondam à mesma mensagem, cada um ao seu modo**



Exemplo de Polimorfismo



Exemplo de Método Polimórfico



O que é Sobrecarga de Métodos?

- Permite que um nome de função seja utilizado mais do que uma vez com diferentes tipos de parâmetros
  - Exemplo: Uma função soma pode ser sobrecarregada para operar com dois parâmetros inteiros e dois reais

O que é Sobrescrita de Métodos?

- Pode-se definir métodos com o mesmo nome, recebendo os mesmos argumentos, porém a sua implementação é realizada de forma diferente e originadas de uma classe pai
  - Chamamos isso de sobrescrita
- Isso acontece para classes diferentes

A Palavra-chave Final

- Algumas vezes pode ser interessante que um elemento de um programa (classe, campo ou método) não possa ser modificado
  - Nestes casos, a palavra-chave final deve ser usada
- Classe final:
  - uma classe especificada como final não pode ter subclasses
  - A classe String é final: garante-se que essa classe não será alterada

A Palavra-chave Final

- Método final:
  - um método definido como final não pode ser redefinido
  - Métodos são definidos como final quando sua implementação não puder ser alterada por alguma razão
  - Métodos declarados como static ou como private são implicitamente final
  - Todos os métodos de uma classe final são implicitamente final

A Palavra-chave Final

- Atributo final:
  - um atributo definido como final não pode ser modificado, ou seja, é uma constante
  - Uma vez atribuído um valor ao campo, este valor não pode ser alterado
  - Uma classe pode ter um campo final que não é inicializado na definição, mas uma vez que a atribuição seja feita, o valor não pode mudar

E para fechar, Desvantagens da Herança..

- Uma desvantagem da herança é que uma subclasse pode herdar métodos que não precisa ou até mesmo que não deveria ter
  - Ainda pode haver um método necessário, mas inadequado
  - Às vezes a subclasse precisa de uma versão personalizada do método

Obrigado!

Por hoje é só pessoal...

Dúvidas?

IsmayleSantos

ismayle@.ufc.br

@IsmayleSantos