Polimorfismo em Java

Jose Macedo

Ck112 - Técnicas em Programação

Conceitos de Orientação a Objetos

- Polimorfismo
- Acoplamento dinâmico
- Classes Abstratas X Classes Concretas

Polimorfismo – Características (i/ii)

- Polimorfismo é um termo proveniente do grego e significa "muitas formas“;
- O polimorfismo não é um pensamento novo para o ser humano pois está contido no dia a dia, principalmente na linguagem. Exemplos:
- Seja o termo “abrir” e as seguintes situações:
- Abrir uma porta; Abrir uma caixa; Abrir uma janela; Abrir uma conta bancária.
- Obs.: A ação de abrir vai ser executada de diferentes formas de acordo com a situação.
- O termo Abrir é polimórfico, ou seja, possui várias formas ou “facetas”, cada uma executada de uma maneira.

Polimorfismo – Características (ii/ii)

- No contexto da programação, o polimorfismo permite a existência de métodos de mesmo nome contendo códigos diferentes selecionado por algum mecanismo automático;
- Ideia básica: “Um nome e vários comportamentos”;
- Vantagens:
  - É possível controlar todas as formas de uma maneira mais simples e geral, sem ter que se preocupar com cada objeto especificamente;
  - Independência da implementação, enxerga-se apenas uma; o nome do método;
  - O polimorfismo permite a adaptação e compreensão de um sistema sem existir a necessidade de alterar substancialmente o que já existe.

Polimorfismo

- Polimorfismo significa que variáveis podem referenciar mais do que um tipo.
- Funções são polimórficas quando seus operandos(parâmetros atuais) podem ter mais do que um tipo.
- Tipos são polimórficos se suas operações podem ser aplicadas a operandos de mais de um tipo.

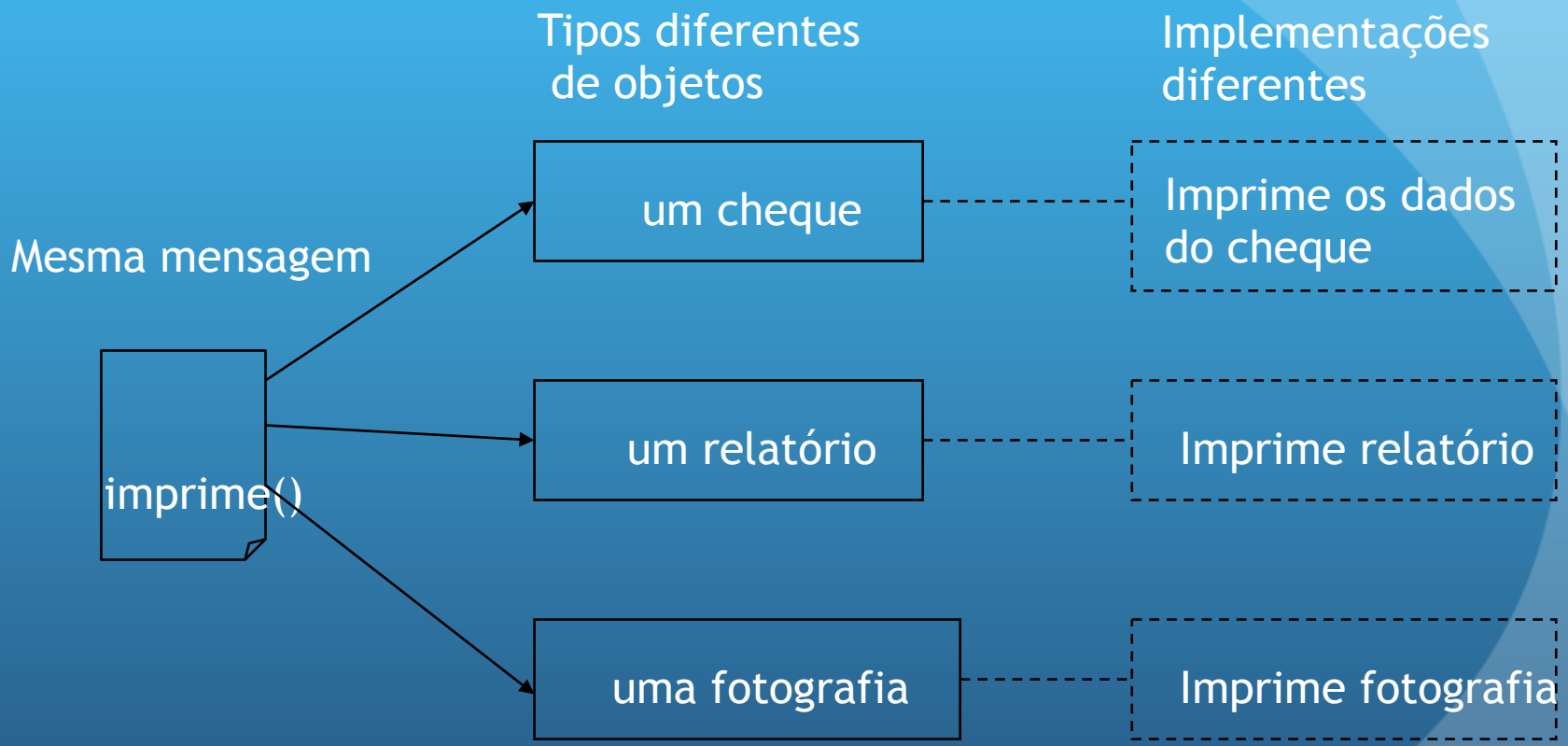
Polimorfismo

- A função definida por:
  - comprimento :: [A] -> NUM, para todos tipos de A
- Informa que:
  - O parâmetro de entrada é uma lista.
  - O tipo do conteúdo da lista (A) não importa.
  - A função devolve um inteiro como saída.
- Em linguagens com tipos monomórficos tem-se que definir diversas funções(inteiros, reais,etc.)

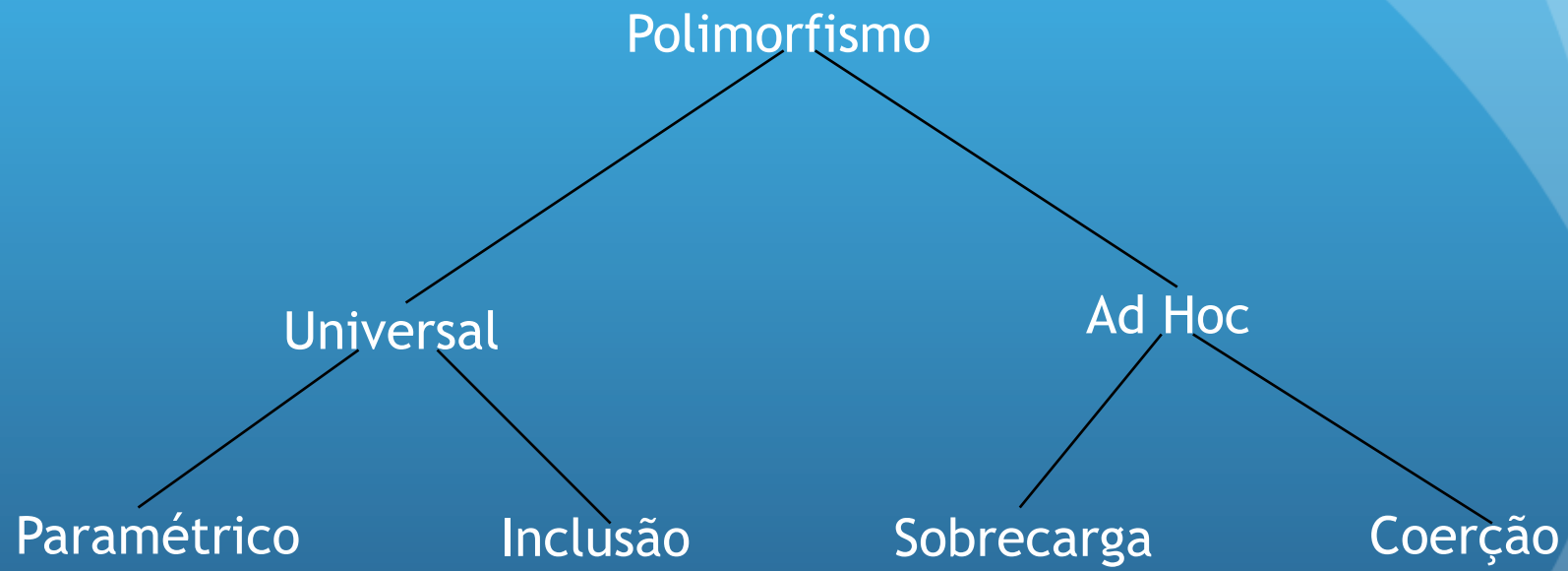
Polimorfismo

- Em OO, polimorfismo significa que diferentes tipos de objetos podem responder a uma mesma mensagem de maneiras diferentes.
- Pode-se definir um método imprime() em diversas classes diferentes.
- Cada versão de imprime() é adaptada para cada tipo de objeto diferente que será impresso.

Polimorfismo



Formas de polimorfismo



Coerção

- Proporciona um meio de contornar a rigidez de tipos monomórficos.
- Se um contexto particular demanda um determinado tipo e um tipo diferente é passado, então a linguagem verifica se há uma coerção adequada.
- Um inteiro pode ser coargido para um real.

Exemplo Coerção

```
…
float Soma (float a, float b) {
        …
}
…
```

- Se o operador *Soma* é definido como tendo 2 parâmetros reais e ocorre a passagem de um inteiro e um real são como parâmetros, o inteiro é “coargido” ou “convertido” para um real;
- Java permite a coerção.

Sobrecarga

- Poliformismo de sobrecarga permite que um nome de função seja usado mais do que uma vez com diferentes tipos de parâmetros.
- Uma função soma pode ser sobrecarregada para operar com parâmetros de tipos diferentes.
- A informação sobre os tipos dos parâmetros é usada para selecionar a função apropriada.

Exemplo Sobrecarga

```
public class Maior{
private int xInt;
private int yInt;
private float xFloat;
private float yFloat;
private double xDouble;
private double yDouble;

public int calcMaior(int x, int y) { ... }
public floa tcalcMaior(float x, float y) { ... }
public double calcMaior(double x, double y) { ... }
public double calcMaior(double a, double b, double c) { ... }
public double calcMaior(double x, int y) { ... }
}
```

Polimorfismo paramétrico

- Uma única função é codificada, e ela trabalhará uniformemente num intervalo de tipos.
- Funções paramétricas são também chamadas de funções genéricas.
- Considere uma classe Pilha(T), onde T é o tipo do elemento que será manipulado.
- Uma classe genérica pode ser escrita independentemente do tipo dos itens armazenados.

Exemplo Polimorfismo Parametrico

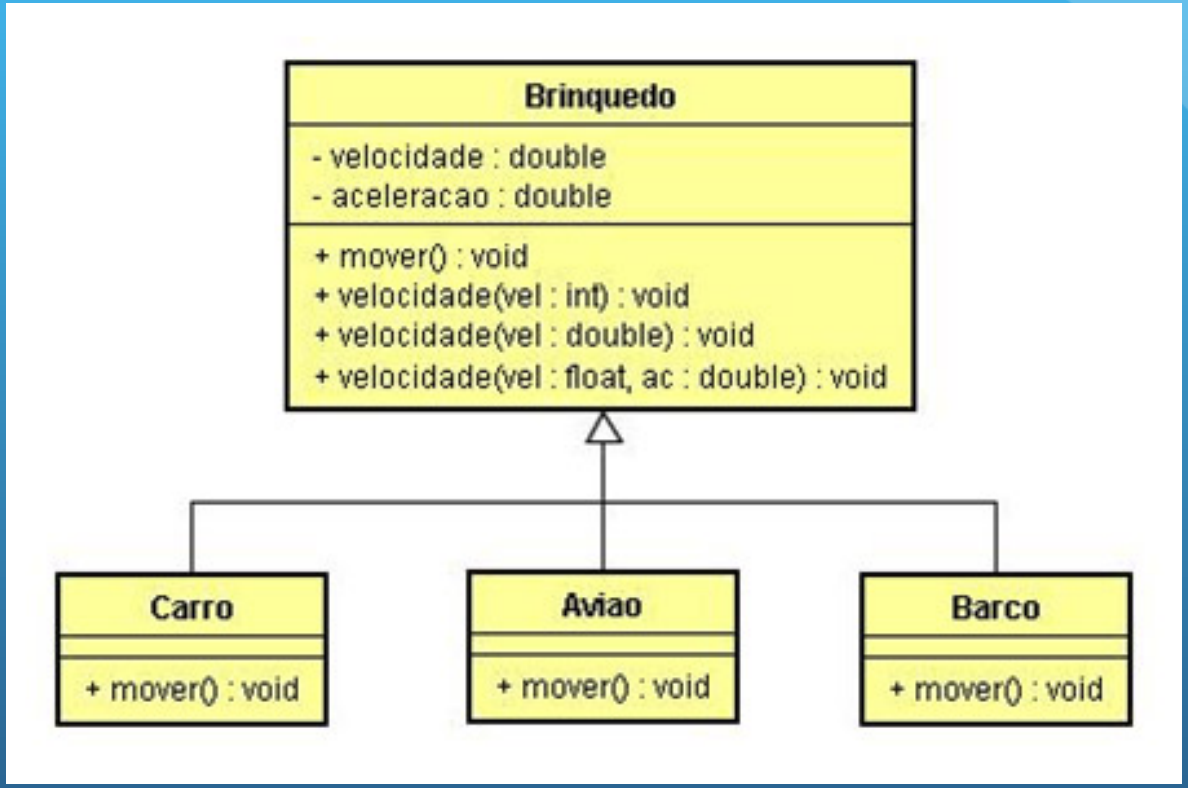
```
Public class List<E> {
    void add(E x){...};
    Iterator<E> iterator() {...};
}
```

- Como no exemplo acima, Java permite tipos genericos;
- A classe *List* está sendo parametrizada com *E* o qual pode receber qualquer tipo;
- O parâmetro *E* vai ser usado no método add e iterator para definir o tipo parâmetro de entrada e saída, respectivamente.

Polimorfismo de inclusão

- Subtipo é uma instância de polimorfismo de inclusão, significando que elementos de um subconjunto também pertencem ao superconjunto.
- Todo o objeto de um subtipo pode ser usado no contexto do supertipo.
- Exemplo da hierarquia de mamíferos.

Exemplo Polimorfismo de Inclusão



Acoplamento dinâmico

- Acoplamento é uma associação possivelmente entre um atributo e uma entidade.
- Variáveis de programa são entidades com atributos de nome, tipo e área de armazenamento.
- Tempo de acoplamento é o tempo em que ocorre o acoplamento entre a entidade e seus atributos.

Acoplamento dinâmico

- Acoplamento estático ocorre antes do tempo de execução, e permanece inalterado durante a execução do programa.
- Acoplamento dinâmico ocorre durante o tempo de execução, e muda no curso da execução do programa.
- No acoplamento dinâmico, as associações podem ser alteradas em tempo de execução.

Acoplamento dinâmico

- Em POO, acoplamento está relacionado com o mapeamento entre o nome de um método e sua implementação.
- Uma mensagem enviada em tempo de execução é dinamicamente acoplada a uma implementação dependendo do tipo do objeto recebedor da mensagem.
- Método calculaBonus() para classe Empregado e classe Diretor.

Exemplo Polimorfismo com Acoplamento Dinâmico

Polimorfismo

- Polimorfismo é uma das ferramentas mais poderosas da programação orientada a objetos. Mais do que isso, é fundamental para o paradigma de OO.
- Exemplo:

```
Animal bicho;
bicho = new Cachorro("Lessie");
bicho = new Gato("Garfield");
bicho.sound();
```

```
public class Animal
{
 private String tipo;

 public Animal(String tipo1)
 {
   tipo = new String(tipo1);
 }

 public void show()
 {
  System.out.println("Eu sou um " + tipo);
 }
}

// Método a ser implementado nas sub-classes
// Cada subclasse irá sobrescrever o método
public void sound() { }
```

```
public class Animal
{
 private String tipo;

 public Animal(String t
 {
   tipo = new String(ti
 }

 public void show()
 {
  System.out.println("E
 }
}

// Método a ser impleme
   classes
public void sound() { }
```

```
public class Cachorro extends Animal
{
 private String nome;     // Nome do cachorro
 private String raça;     // Raça do cachorro

 public Cachorro(String nome1)
 {
  // Chama o construtor da classe base
  super("Cachorro"');
  nome = nome1;              // Nome fornecido
  raça = "Desconhecida";   // Raça default
 }

 public Cachorro(String nome1, String raça1)
 {
  // Chama o construtor da classe base
  super("Cachorro"');
  nome = nome1;            // Nome fornecido
  raça =raça1;             // Raça fornecida
 }

 public void sound()
 {
  System.out.println("Au, au");
 }
}
```

```
public class Animal
{
 private String tipo;

 public Animal(String t
 {
   tipo = new String(ti
 }

 public void show()
 {
  System.out.println("E
 }
}

// Método a ser impleme
   classes
public void sound() { }
```

```
public class Gato extends Animal
{
 private String nome;     // Nome do gato
 private String raça;     // Raça do gato

 public Gato(String nome1)
 {
  // Chama o construtor da classe base
  super("Gato"');
  nome = nome1;              // Nome fornecido
  raça = "Desconhecida";   // Raça default
 }

 public Gato(String nome1, String raça1)
 {
  // Chama o construtor da classe base
  super("Gato"');
  nome = nome1;            // Nome fornecido
  raça =raça1;             // Raça fornecida
 }

 public void sound()
 {
  System.out.println("Miau");
 }
}
```

```
public class Animal
{
 private String tipo;

 public Animal(String tipo1)
 {
   tipo = new String(tipo1);
 }

 public void show()
 {
  System.out.println("Eu
 }
}

// Método a ser implement
   classes
public void sound() { }
```

```
public class Gato e
{
 private String nom
 private String raç

 public Gato(String
 {
  // Chama o constr
  super("Gato"');
  nome = nome1;
  raça = "Desconhec
 }

 public Gato(String nome1, String raça1)
 {
  // Chama o construtor da classe base
  super("Gato"');
  nome = nome1;          // Nome fornecido
  raça =raça1;           // Raça fornecida
 }

public void sound()
{
 System.out.println("Miau");
}
}
```

```
public class Cachorro extends Animal
{
 private String nome;      // Nome do cachorro
 private String raça;      // Raça do cachorro

 public Cachorro(String nome1)
 {
  // Chama o construtor da classe base
  super("Cachorro"');
  nome = nome1;            // Nome fornecido
  raça = "Desconhecida";   // Raça default
 }

 public Cachorro(String nome1, String raça1)
 {
  // Chama o construtor da classe base
  super("Cachorro"');
  nome = nome1;          // Nome fornecido
  raça =raça1;           // Raça fornecida
 }

public void sound()
{
 System.out.println("Au, au");
}
}
```

```
public class TestaPolimorfismo
{
 public static void main(String[] args)
 {
  // Cria um array de animais
  Animal[] bichos = { new Cachorro("Rin
   Alemão"),
                      new Gato("Garfi
  Animal mascote;
  // Escolhe 5 mascotes (sorteados)
  for (int i = 0; i < 5; i++)
  {
   // Cria um índice randômico de 0 a b
   int indice = (int)(bichos.length * M

   mascote = bichos[indice]; // Escolhe

   System.out.println("\nSua escolha:")
   mascote.show();
   mascote.sound();
  }
}
```