Polimorfismo

- O que é polimorfismo?
  - Significa que variáveis podem referenciar mais do que um tipo.
  - Não é um conceito novo e várias linguagens de programação aplicam.
  - Funções são polimórficas quando seus operandos (parâmetros atuais) podem ter mais do que um tipo.
  - Tipos são ditos polimórficos se suas operações podem ser aplicadas a operandos de mais de um tipo.

Polimorfismo

- Exemplo de função polimórfica:

`comprimento::[A] -> NUM, para todos os tipo A.`

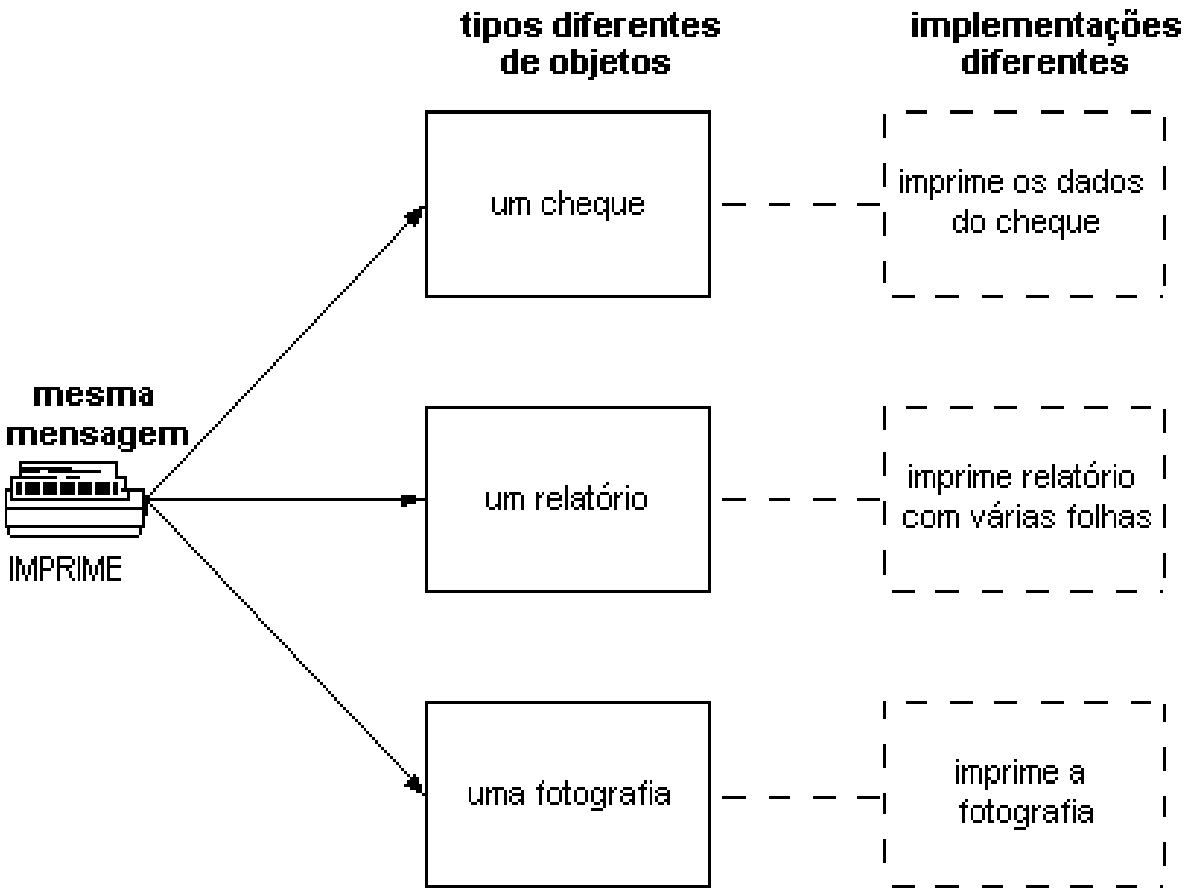
- Portanto:
  - Comprimento é uma função cujo parâmetro de entrada é uma lista (simbolizada pelos colchetes).
  - O tipo do conteúdo da lista não importa (A).
  - A função devolve um inteiro como saída.
- Uma linguagem de tipos monomórficos forçaria o programador a definir diferentes funções para retornar o comprimento de uma lista de inteiros, de uma lista de reais e assim por diante. Exemplo: Pascal e Algo68.

Polimorfismo

- Em particular, no contexto do modelo de objetos, polimorfismo significa que diferentes tipos de objetos podem responder a uma mesma mensagem de maneiras diferentes.

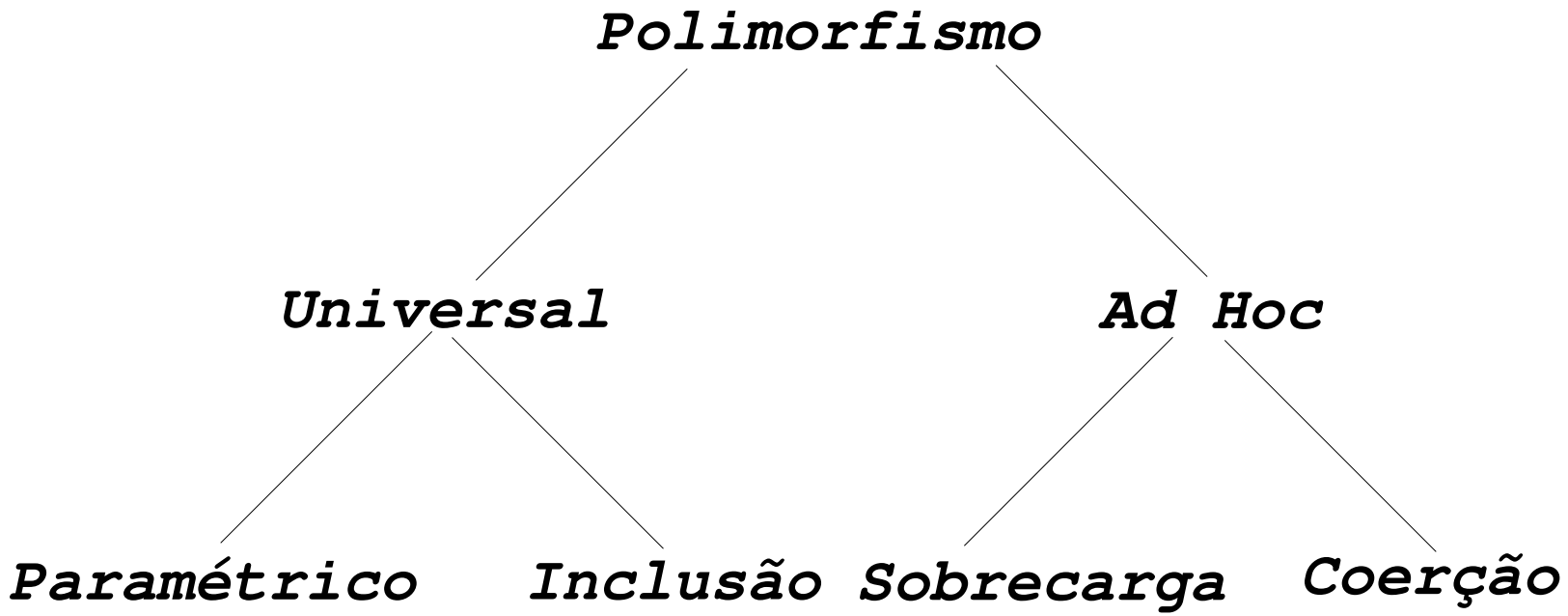
Polimorfismo

- Exemplo de método polimórfio



Polimorfismo

- Formas de polimorfismo
  - Uma taxonomia amplamente utilizada:



Polimorfismo

- Polimorfismo *ad hoc*
  - Não existe um modo único e sistemático de determinar o tipo de resultado de uma função em termos dos tipos dos seus argumentos de entrada.
  - É uma forma limitada de polimorfismo.
  - Possui duas formas: coerção e sobrecarga.
- Polimorfismo universal
  - Trabalha potencialmente num conjunto infinito de tipos de modo disciplinado.
  - Possui duas formas: paramétrico e inclusão.

Polimorfismo

- Coerção
  - Meio para contornar a rigidez dos tipos monomórficos.
  - Existe um mapeamento interno entre tipos.
  - Exemplo:
    - Se o operador soma é definido como tendo 2 parâmetros reais e um inteiro e um real são passados como parâmetros, o inteiro é "coargido" para um real.
- Sobrecarga
  - 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.

Polimorfismo

- Paramétrico
  - Uma única função é codificada e ela trabalhará uniformemente num intervalo de tipos.
  - Funções paramétricas também são chamadas de funções genéricas.
  - Exemplo:
    - A função comprimento apresentada anteriormente.
- Inclusão
  - É encontrada somente em linguagens orientadas a objetos e está relacionada com a noção de subtipo.
  - Exemplo:
    - Hierarquia de classes.

Polimorfismo

- Em C++, por exemplo, temos o polimorfismo parámetrico (através do uso da palavra reservada template), o polimorfismo de sobrecarga e o polimorfismo de inclusão.
- Em C++, o uso de funções membros virtuais numa hierarquia de classes permite que a seleção em tempo de execução da versão mais adequada do método.

Acoplamento Dinâmico

- O que é acoplamento?
  - Acoplamento é uma associação possivelmente entre um atributo e uma entidade.
  - Exemplo:
    - Variáveis são entidades de programas e seus atributos são seu nome, seu tipo e sua área de armazenamento.
  - O tempo em que o acoplamento entre a entidade e seus atributos ocorre é chamado de tempo de acoplamento.
  - Um acoplamento pode ser estático ou dinâmico.

Acoplamento Dinâmico

- Um acoplamento é estático ou adiantado se ocorre antes do tempo de execução e permanece inalterado durante a execução do programa.
- Um acoplamento é dinâmico ou atrasado se ocorre durante o tempo de execução e muda no curso da execução do programa.
- A maior vantagem do acoplamento dinâmico é que as associações podem ser alteradas em tempo de execução.

Acoplamento Dinâmico

- Em programação orientada a objetos, temos o uso de verificação estática de tipos associada com acoplamento dinâmico.
- O acoplamento dinâmico é usado para a associação do nome de uma operação à sua implementação.
- Essa associação é decidida apenas em tempo de execução, pois a implementação a ser executada depende do tipo do objeto que recebe a mensagem.

Acoplamento Dinâmico

- Classes Abstratas vs. Classes Concretas
  - Uma classe abstrata é uma classe que não tem instâncias diretas, mas cujas classes descendentes têm instâncias diretas.
  - Uma classe concreta é uma classe que pode ser instanciada.
  - Existem duas idéias principais envolvidas com a noção de classe abstrata:
    - Presença de operações abstratas.
    - Não possui a habilidade de criar instâncias.

Acoplamento Dinâmico

- Se uma classe contém operações abstratas então a classe não pode ser instanciada porque se um objeto for criado, ele não será capaz de responder a todas as mensagens.
- Portanto, a presença de uma operação abstrata implica na inabilidade de instanciar objetos.
- Entretanto o contrário não é verdadeiro.

Acoplamento Dinâmico

- A conseqüência é que as alterações efetuadas nos métodos e estrutura da classe podem ser automaticamente passadas para todas as suas instâncias.
- Este mecanismo implícito de alteração permite que sistemas possam ser atualizados com base em grupos.
- Neste modelo, as distinções entre classes e objetos são removidas. Na verdade, a noção de classe é eliminada.
- O projetista inicialmente considera um protótipo em particular e depois descobre similaridades e/ou diferenças com outros objetos.
- A idéia é começar com casos particulares e depois generalizá-los ou especializá-los.

Acoplamento Dinâmico

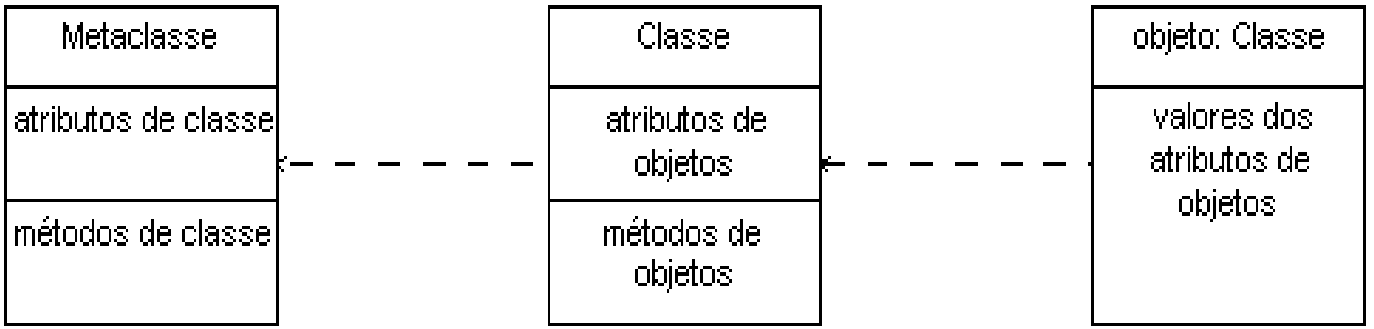
- A principal vantagem de delegação sobre herança é que delegação facilita a mudança de implementação de operações em tempo de execução.

Acoplamento Dinâmico

- Metaclasses
  - Metainformação é um termo genérico aplicado a qualquer dado que descreve outro dado.
  - As classes não são objetos porque elas próprias não são instâncias de classes.
  - Entretanto, em algumas linguagens orientadas a objetos, tal como Smalltalk, existem dois tipos de elementos geradores no modelo de objetos:
    - Metaclasses que geram classes.
    - Classes que geram instâncias terminais.
  - Portanto, nesse caso, todas as entidades do modelo de objetos são objetos, inclusive as classes.

Acoplamento Dinâmico

- Conceito de Metaclasse:



- Mais especificamente, metaclasse é uma classe que descreve outra classe, isto é, ela é uma classe cujas instâncias são classes. Ela guarda metainformação de uma classe.

Acoplamento Dinâmico

- Existem pelo menos dois benefícios da representação de classes como objetos:
  - As informações globais relativas a todos os objetos de uma classe podem ser armazenadas nos atributos de classe.
  - A segunda vantagem é o seu uso para criação/iniciação de novas instâncias da classe.
- Linguagens como C++ e Java não dão apoio direto para a abordagem classes como instâncias de metaclasses.

Acoplamento Dinâmico

- De modo geral, pode-se dizer que o paradigma orientado a objetos dá suporte a pelo menos três níveis de abstrações:
  - Abstração de dados para comunicação de objetos.
  - Super-abstração (herança) para o compartilhamento de comportamento e gerência de objetos.
  - Meta-abstração (metaclasse) como base para a auto-representação do sistema.

Polimorfismo

- Polimorfismo é a capacidade de assumir formas diferentes.
- O polimorfismo permite-nos escrever programas de uma forma geral para tratar uma ampla variedade de classes relacionadas existentes e ainda a serem especificadas.
- Usa-se uma variável de um tipo único (normalmente tipo da superclasse) para referenciar objetos variados do tipo das subclasses.
- Envolve o uso automático do objeto armazenado na super-classe para selecionar um método de uma das sub-classes. O tipo do objeto armazenado não é conhecido até a execução do programa. A escolha do método a ser executado é feita dinamicamente.

Estudo de Caso de Polimorfismo

```
public class Animal {
   private String tipo;
   public Animal(String tipo1){
      tipo = new String(tipo1);
      }
   public void exibir(){
      System.out.println("Eu sou um" + tipo);
      }
   //Método a ser implementado nas sub-classes.
   public void sound(){ }
   }
```

Estudo de Caso de Polimorfismo

```
public class Cachorro extends Animal {
   private String nome, raca;
   public Cachorro(String nome1){
      super(“cachorro”);
      nome = nome1; raca = “SRD”; }
   public Cachorro(String nome1, String raca1){
      super(“cachorro”);
      nome = nome1; raca = raca1; }
   public void sound() {
      System.out.println(“Au, Au”);
      }
   }
```

Estudo de Caso de Polimorfismo

```
public class TestePolimorfismo {

   public static void main(String[] args){

      Animal[] bichos = {new Cachorro("Rintintin",
       "Pastor Alemao"), new Gato("Garfield"), new
       Pato("Donald")};

      Animal mascote;

      for (int i=0; i<5; i++){

         int indice = (int) (bichos.length *
          Math.random() - 0.001);

         mascote = bichos[indice];

         System.out.println("\n Escolha: ");

         mascote.exibir();

         mascote.sound(); }

      }
```

Polimorfismo

- Retorna **true** caso o objeto **a** seja uma instância da classe Gato.
  - Animal a.
  - if (a instanceof Gato)
  - // vacinar

Polimorfismo

- Consideremos o método **sound()** desenvolvido na classe Animal e que não possui nenhuma instrução, não fazendo sentido.
- Este tipo de situação (criar uma superclasse a partir da qual subclasses serão derivadas) acontece muito freqüentemente em programação orientadas a objetos, quando o objetivo é apenas tirar vantagem do polimorfismo.
- Uma outra alternativa em Java é o uso de classes abstratas. Uma classe abstrata é a classe na qual um ou mais métodos são declarados, mas não implementados.