Suporte à POO

Baseado em
"Concepts of Programming Languages- 4rd Ed."
Robert W. Sebesta
(Addison-Wesley, 1998)

Elaborado por: Alberto Castro
DCC/UA, 1999

Categorias das linguagens que suportam POO:

1. Suporte a POO acrescentado a uma linguagem já existente

- **C++ (também suporta programação procedural e orientada a dados)**
- **Ada 95 (também suporta programação procedural e orientada a dados)**
- **CLOS (também suporta programação funcional)**
- **Scheme (também suporta programação funcional)**

2. Suporte a POO, mas com a mesma "aparência" e usam a mesma estrutura básica das linguagens imperativas mais antigas

- **Eiffel (não foi baseada diretamente em nenhuma linguagem anterior)**
- **Java (baseada no C++)**

3. Linguagens de POO puras

- **Smalltalk**

Evolução do Paradigma

1. Procedural - 1950s-1970s (abstração procedural)

2. Orientada a dados - inicio dos anos 80 (orientada a dados)

3. POO - final dos anos 80 (Herança e binding dinâmico)

Origens da Herança

Observações de meados dos anos 80:

- **Aumento da produtividade pode vir da reutilização**
- **TADs são difíceis de reutilizar - quase nunca corretamente**
- **Todos os TADs são independentes e estão num mesmo nível**

Herança resolve ambos: Reutilização de TADs após modificações menores e define classes em uma hierarquia

Definições sobre POO:

- **TADs são chamados *classes***
- **Instâncias de classes são chamadas *objetos***
- **Uma classe que herdeira é chamada *classe derivada* ou uma *subclasse***
- **A classe da qual outra classe é herdeira é chamada classe-pai ou super-classe**
- **Subprogramas que definem operações sobre objetos são chamados *métodos***
- **A coleção completa de métodos de um objeto é chamada de seu *protocolo de mensagens* ou *interface***
- **Mensagens possuem duas partes - o nome do método e o objeto de destino**
- **No caso mais simples, uma classe herda todas as entidades de seu pai**

- Herança pode complicada por controles de acesso para entidades encapsuladas

- Uma classe pode ocultar entidades de suas subclasses
- Uma classe pode ocultar entidades de seus clientes

- Além dos métodos padrão de herança, uma classe pode modificar um método herdado

- O novo método substitui o herdado
- O método na classe pai é sobrepujado

- Existem dois tipos de variáveis em uma classe:

1. Variáveis da classe
2. Variáveis da instância

- Existem dois tipos de métodos em uma classe:

1. Métodos da classe - mensagens para a classe
2. Métodos da instância - mensagens para objetos

- Herança simples × herança múltipla

- Desvantagem de herança por reutilização:

- Cria interdependências entre classes que complicam a manutenção

Polimorfismo em LPOOs

- Uma variável polimórfica pode ser definida em uma classe que é apta a referenciar (ou apontar) a objetos da classe e objetos de qualquer de seus descendentes

- Quando uma hierarquia de classes inclui classes que sobrepõe métodos e tais métodos são chamados através de uma variável polimórfica, o binding para o método corrente deve ser dinâmico

- Polimorfismo simplifica a adição de novos métodos

- Um *método virtual* é aquele que não inclui uma definição (apenas define um protocolo)

- Uma *classe virtual* é aquela que inclui ao menos um método virtual

- Uma classe virtual não pode ser instanciada

Questões de projeto para LPOO

1. A exclusividade de objetos

a. Tudo são objetos
***vantagens* - elegância e pureza**
***desvantagens* - operações lentas em objetos simples (e.g., float)**

b. Adiciona objetos a um sistema de tipagem completa
***Vantagem* - operações rápidas em objetos simples**
***Desvantagens* - resulta em um sistema de tipagem confuso**

c. Inclui um sistema de tipagem de estilo imperativo para primitivas mas considera todo o resto como objetos
***Vantagem* - operações rápidas em objetos simples e um sistema de tipagem relativamente pequeno**
***Desvantagem* - (ainda) alguma confusão devido aos dois sistemas de tipagem**

2. Subclasses são subtipos?

- Um relacionamento "é-um" é mantido entre uma classe pai e um objeto da subclasse?

3. Herança de implementação e de interface

- Se apenas a interface de uma classe-pai está visível para a subclasse, trata-se de *herança de interface*

***Desvantagem* - pode resultar em ineficiências**

- Se tanto a interface quanto a implementação da classe-pai está visível para a subclasse, trata-se de *herança de implementação*

***Desvantagem* - modificações na classe-pai requerem recompilação de subclasses, e algumas vez até a modificação das subclasses**

4. Verificação de tipo e polimorfismo

- Polimorfismo pode requerer verificação dinâmica de tipos de parâmetros e do valor retornado
- Verificação dinâmica de tipos é onerosa e retarda detecção de erros

- Se métodos de sobreposição são restritos a ter os mesmos tipos de parâmetro e de valor de retorno, a verificação pode ser estática

5. Herança simples e múltipla

- Desvantagem de herança múltipla:
- Complexidade da linguagem e implementação
- Ineficiência potencial - binding dinâmico é mais oneroso com herança múltipla (mas não muito)

- Vantagem:
- Algumas vezes é extremamente conveniente e valiosa

6. Alocação e dasalocação de objetos

- A partir de onde os objetos são alocados?
- Se eles todos estiverem no heap, referência a eles são uniformes

- A desalocação é explicita or implicita?

7. Binding dinâmico e

- Todas as mensagens para métodos deveriam ser dinâmicas?

Visão geral do Smalltalk

- Smalltalk é uma linguagem de POO pura
- Tudo são objetos
- Toda "computação" é através de objetos enviando mensagens a objetos
- Ela não adota a aparência das linguagens imperativas

- O ambiente Smalltalk
- O primeiro sistema com interface gráfica (GUI) completo
- Um sistema completo para desenvolvimento de software
- Todo o código fonte dosistema é disponível para o usuário, que pode modifica-lo, caso deseje

Introdução ao Smalltalk

- Expressões
- Quatro tipos:
1. Literais (números, strings, e palavras-chave)

2. Nomes de variáveis (todas as variáveis são referências)

3. Expressões-mensagem

4. Expressões-bloco

- Expressões-mensagem

- Duas partes: o objeto receptor e a própria mensagem
- A (parte) mensagem especifica o método e possivelmente alguns parâmetros
- Respostas a mensagens são objetos

- Mensagens podem ser de três formas:
1. *Unária* (sem parâmetros)
e.g., `myAngle sin`
(envia uma mensagem para o método `sin` do objeto `myAngle`)

2. *Binária* (um parâmetro, um objeto)
e.g., `12 + 17`
(envia a mensagem "`+ 17`" para o objeto `12`; o objeto-parâmetro é "`17`" e o método é "`+`")

3. *Palavra-chave* (usa-se palavras-chave para organizar os parâmetros)
e.g., `myArray at: 1 put: 5`
(envia os objetos "`1`" e "`5`" para o método `at:put:`do objeto `myArray`)

- Mensagens múltiplas para o mesmo objeto podem ser "enfaixadas" todas juntas, se paradas por ponto-e-vírgula

Métodos

- Forma geral:
  padrão_de_mensagem [| temps |] comandos

  - Um padrão de mensagem é como os parâmetros formais de um subprograma
    - Para uma mensagem unária, é só o nome
    - Para outros, ele lista palavras-chave e nomes formais
  - temps são apenas nomes - Smalltalk não possui tipos!

Atribuições

- Forma mais simples:
  nome1 <- nome2

- É simplesmente uma atribuição a ponteiros

- O lado direito da expressão (RHS) pode ser uma expressão-mensagem
  e.g., `index <- index + 1`

Blocos

- Uma sequência de comandos, separados por pontos, delimitados por colchetes
e.g., `[index <- index + 1. sum <- sum + index]`

- Um bloco especifica alguma coisa, mas não a faz (executa)

- Para solicitar a execução de um bloco, envia-se uma mensagem unária, `value`
  e.g., `[…] value`

- Se um bloco é atribuido a uma variável, ele é avaliado enviando-se um `value` para aquela variável
  e.g.,
```
addIndex <- [sum <- sum + index]
   …
addIndex value
```

- Blocos podem possuir parâmetros, como em
  `[:x :y | statements]`

- Se um bloco contem uma expressão relacional, ela retorna um objeto booleano, `true` ou `false`

Iteração

- Os objetos `true` e `false` possuem métodos para construir compostos de controle

- O método `WhileTrue:` de `Block` é usado para loops pré-testados. Ele é definido para todos os blocos que retornam objetos booleanos.

e.g.,
```
[count <= 20]
   whileTrue [sum <- sum + count.
              count <- count + 1]
```

- `timesRepeat:` é definido para inteiros e pode ser utilizado para construir loops de contagem
  e.g.,
```
xCube <- 1.
   3 timesRepeat: [xCube <- xCube * x]
```

Seleção

- Os objetos booleanos possuem os métodos `ifTrue:ifFalse:` , os quais podem ser usados para construir seleção
e.g.,
```
total = 0
  ifTrue: […]
  ifFalse: […]
```

Macro aspectos do Smalltalk

- *Verificação de tipos e polimorfismo*

  - Todos os bindings de mensagens a métodos são dinâmicos

    - O processo é procurar o objeto para o qual a mensagem é enviada para o método; caso não encontrado, procurar a superclasse, e assim por diante.

  - Uma vez que todas as variáveis são sem-tipo, métodos são todos polimórficos

- *Herança*

  - Todas as subclasses são subtipos (nada pode ser ocultado)

  - Toda herança é herança de implementação

  - Sem herança múltipla

  - Métodos podem ser redefinidos, mas os dois não são relacionados

C++

***- Características gerais*:**

- Sistema de tipagem mista
- Construtores e destrutores
- Cuidadoso controle de aceso a entidades de classe

- Herança

- Uma classe não necessita ser subclasse de nenhuma classe

- Controle de acesso para membros são
1. Private (visível apenas na classe e "amigos")
2. Public (visível nas subclasses e clientes)
3. Protected (visível na classe e nas subclasses)

- Em adição, o processo de hierarquização em subclasses pode ser declarado com controles de acessos, os quais definem mudanças potenciais no acesso por subclasses

- Herança múltipla é permitida

- Binding dinâmico

- Um método pode ser definido como `virtual`, o que significa que ele pode ser chamado através de variáveis polimórficas e dinamicamente ligados (binding) a mensagens

- Uma função virtual pura não tem definição alguma
- Uma classe que tem ao menos uma função virtual pura é uma classe abstrata

- Avaluiação

- C++ provê extensivo controle de acesso (diferente do Smalltalk)

- C++ provê herança múltipla

- Em C++, o programador deve decidir em tempo de projeto, quais métodos terão binding estático e quais terão binding dinâmico.

- Binding estático é mais rápido!

- Verificação de tipo em Smalltalk é diâmico (flexível, mas insegura)

Java

- Características gerais

- Todos os dados são objetos, exceto os tipos primitivos

- Todos os tipos primitivos possuem classes "empacotadoras" que armazenam um valor (de dados)

- Todos os objetos são dinâmicos no heap, são referenciados através de variáveis de referência, e a maioria é alocada com `new`

- Herança

- Somente herança simples, mas existe uma categoria de classe abstrata que provê alguns dos benefícios da herança múltipla (interface)

- Uma interface pode incluir apenas declarações de métodos e constantes nomeadas
e.g.,

```
public class Clock extends Applet
                  implements Runnable
```

- Métodos podem ser `final` (não podem ser sobrepostos)

- Binding Dinâmico

- Em Java, todas as mensagens tem binding dinâmico aos métodos, exceto quando o método é `final`

- Encapsulamento

- Dois compostos, classes e pacotes

- Pacotes proveem um "container" para classes que são relacionadas

- Entidades definidas sem um modificador de escopo (access) tem o escopo do pacote, que os faz visíveis através do pacote no qual eles são definidos

- Toda classe num pacote é "amiga" para as entidades no escopo do pacote (que se encontram noutro lugar no pacote)

Ada 95

- Características gerais

- POO foi uma das mais importantes extensões para o Ada 83

- Container de encapsulamento é um pacote que define um tipo etiquetado

- Um tipo etiquetado é aquele no qual todo objeto inclui uma etiqueta para indicar seu tipo durante a execução

- Tipos etiquetados podem ser tipos privados ou records

- Nenhum construtor ou destrutor é chamado implicitamente

- Herança

- Subclasses são derivadas de tipos etiquetados

- Novas entidades numa subclasse são adicionadas num record

Exemplo:

```
with PERSON_PKG; use PERSON_PKG;
package STUDENT_PKG is
  type STUDENT is new PERSON with
    record
      GRADE_POINT_AVERAGE : FLOAT;
      GRADE_LEVEL : INTEGER;
    end record;
  procedure DISPLAY (ST: in STUDENT);
end STUDENT_PKG;
```

- `DISPLAY` é sobreposto a partir de `PERSON_PKG`

- Todas as subclasses são subtipos

- Apenas herança simples, exceto através de "generics"

- Binding dinâmico

- Binding dinâmico é feito usando variáveis polimórficas chamadas tipos com abrangência de classe (*classwide*)
e.g., para o tipo etiquetado `PERSON`, o tipo classwide é `PERSON'class`

- Outros bindings são estáticos
- Qualquer método pode possuir binding dinâmico

Eiffel

- Características gerais

- Possui tipos primitivos e objetos

- Todos os objetos atendem três operações: `copy`, `clone`, e `equal`

- Métodos são chamados *rotinas*

- Instâncias de variáveis são chamados a*tributos*

- As rotinas e atributos de uma classe são (juntas) chamadas de seus *aspectos*

- Criação de objeto é feito com um operador (`!!`)

- Construtores são definidos na cláusula `creation`, e são explicitamente chamados no comando no qual o objeto é criado

- Herança

- O ancestral (pai) de uma classe é especificado com a cláusula `inherit`

- Controle de acesso

- cláusulas `feature` especificam controle de acesso para as entidades nele definidas

- Sem um modificador, as entidades numa declaração `feature` são visíveis para subclasses e clientes

- Com o modificador `child`, entidades são ocultas de clientes mas são visíveis a subclasses

- Com o modificador `none`, entidades são ocultas de clientes e subclasses

- Aspectos herdados podem ser ocultos de subclasses com `undefine`

- Classes abstratas podem ser definidas incluindo-se o modificador `deferred` na definição de classe

- Binding dinâmico

- Quase todas as mensagens tem binding dinâmico

- Um método que se sobrepõe deve ter parâmetros que são atribuições compatíveis com aqueles do método sobreposto.

- Todos os aspectos devem ser definidos numa cláusula `redefine`

- Acesso a aspectos sobrescritos é possível colocando-se seus nomes numa cláusula `rename`

- Avaliação

- Similar ao Java no sentido de não suportar programação procedural e praticamente todo o binding de mensagens ser dinâmico

- Projeto de suporte a POO elegante e "limpo"

Implementando Compostos OO

- Registros de instâncias de classes (CIRs) armazenam o estado de um objeto

- Se uma classe possui um ancestral (pai), as variáveis que são instâncias da classe são adicionadas ao CIR-pai

- Tabelas de métodos virtuais (VMTs) são usadas para binding dinâmicos