Herança e Polimorfismo de Inclusão

Sérgio Queiroz de Medeiros
sergio@ufs.br

29 de maio de 2012

Polimorfismo

- Ad-hoc
  - Coerção
  - Sobrecarga (*Overloading*)
- Universal
  - Paramétrico
  - Inclusão

Herança

- Herança é uma maneira de reusar código de outros objetos, ou de estabelecer um subtipo a partir de um objeto existente
- Exemplo de herança em C++:

```
class derived : protected base { ...
```

- O que acontece?

Visibilidade em C++

- Qualquer classe pode limitar a visibilidade de seus membros
  - *public*
  - *protected*
  - *private*
- Uma classe derivada pode restringir a visibilidade de membros de uma classe base, mas não aumentá-la
- Restrições de visibilidade em uma classe derivada podem ser definidas individualmente para cada membro da classe

Visibilidade

- **Eiffel**: classes derivadas podem tanto restringir quando aumentar a visibilidade de mebros da classe base
- **Java**: um membro *protected* também é visível dentro do pacote da classe.
  - Possui visibilidade de pacote
- **C#**: modificador *internal* permite um membro da classe ser visível no *assembly* (equivalente ao *.jar*) na qual a classe aparece

Polimorfismo

- Uma consequência da herança é que um objeto de uma classe derivada possui todos os membros de sua classe base C
  - Podemos usar objetos da classe D onde objetos da classe C são esperados (desde que D não restrinja a visibilidade dos membros públicos de C)
- A habilidade de usar uma classe derivada onde a classe base é esperada é chamada de polimorfismo de inclusão ou polimorfismo de subtipo

Polimorfismo

```
class person { ...
class student : public person { ...
class professor : public person { ...

student s;
professor p;
...
person *x = &s;
person *y = &p;

s.f();
p.f();
x->f();
y->f();
```

Ligação de Método Dinâmica

- Como saber em `x->f()` de qual classe virá o método `f`?
  - A escolha é com base no tipo da variável `x`?
  - A escolha é com base na classe do objeto `s` referenciado por `x`?
- Ligação de Método Estática x Ligação de Método Dinâmica

Ligação de Método Dinâmica

- A ligação de método dinâmica é mais custosa do que a estática
  - Embora o custo seja pequeno, ele possui impacto em métodos curtos
- Smalltalk, Objective-C, Python, Ruby e Lua usam ligação dinâmica para todos os métodos
- Java usa ligação dinâmica por padrão, mas permite que classes e métodos sejam marcados como `final`

Ligação de Método Dinâmica

- C++ e C# usam ligação de método estática por padrão, mas permitem especificar ligação de método dinâmica
  - Overriding x Redefining

```
class person {
  public:
    virtual void f();
    ...
```

Classes Abstratas

- Na maioria das linguagens OO é possíver omitir o corpo de métodos virtuais em uma classe base
- **Java**:

```
abstract class person {
   public abstract void f();
```

- **C++**:

```
class person {
   public:
      virtual void f() = 0;
```

- Em C++ métodos abstratos são chamados de métodos virtuais puros

Que método usar?

- Na ligação de método estática o compilador pode determinar que método usar com base no tipo da variável
- Na ligação de método dinâmica o método é escolhido em tempo de execução
  - Tabela de métodos virtuais (*vtable*)
  - Cada classe possui uma *vtable*

Tabela de Métodos Virtuais

```
class foo {
  int a;
  double b;
  char c;
public:
  virtual void k(...
  virtual int l(...
  virtual void m();
  virtual double n(...
} F;
```

```
class bar : public foo {
  int w;
public:
  void m();
  virtual double s(...
  virtual char *t(...
  ...

};  B;
```

Tabela de Métodos Virtuais

```
foo F;
bar B;
foo *q;
bar *s;
...
q = &B;  //ok
s = &F;  //erro semântico estático
s = dynamic_cast<bar *>(q);
s = (bar *) q;
```

Polimorfismo de Inclusão x Polimorfismo Paramétrico

- Polimorfismo paramétrico é útil para construir abstrações sobre tipos não relacioados
  - Pilhas
  - Árvores
  - Conjuntos
- Polimorfismo de inclusão é útil para construir abstrações sobre tipos relacionados
  - Uso de herança

Herança Múltipla

- Algumas linguagens, como C++ e Python, permitem herança múltipla
- Outras linguagens, como Java, C# e Ruby, permitem uma forma limitada de herança múltipla (interfaces)
- Herança múltipla (como implementado em C++ e Python) adiciona um grau de complexidade à linguagem que parece ultrapassar os seus benefícios

Herança Múltipla

- As classes `X` e `Y` possuem o método `f`. A classe `Z` herda de `X` e `Y`. De quem `Z` herda `f`?
- As classes `X` e `Y` possuem `A` como pai. A classe `Z` herda de `X` e `Y`. Quantas cópias de `A` a classe `Z` possui?
- A implementação de herança simples assumia que a representação de um objeto da classe pai era um prefixo da representação de um objeto de uma classe derivada. Com herança múltipla, como pode cada pai ser um prefixo do filho?

Referências

- Programming Language Pragmatics (Michael Schott)
  - Seções 9.2, 9.4 e 9.5
- Concetps of Programming Languages (Robert Sebesta)
  - Seções 9.8, 11.5 a 11.7 e Capítulo 12