Padrão Básico de Projeto: Interfaces e Polimorfismo

Herança de implementação versus herança de interface

- Há uma diferença grande entre uma classe e seu tipo
  - A classe define ambos um tipo e uma implementação desse tipo
  - O tipo define apenas a interface oferecida para acessar objetos da classe
  - Um objeto pode ter muitos tipos
  - Classes diferentes podem implementar o mesmo tipo
- Herança de classe ("extends") significa herança de implementação
  - A sub-classe herda a implementação da super-classe
  - É um mecanismo para compartilhar código e representação
- Herança de interface (ou subtipos) descreve quando um objeto pode ser usado em vez de outro
- Ao fazer herança de classe, automaticamente faz também herança de interface
- Algumas linguagens não permitem definir tipos separadamente de classes
  - Mas, neste caso, a classe puramente abstrata serve

"Program to an interface, not an implementation"

- O fato de que a herança de implementação permite facilmente reusar a funcionalidade de uma classe é interessante mas não é o aspecto mais importante a ser considerado
- Herança oferece a habilidade de definir famílias de objetos com interfaces idênticas

- Isso é extremamente importante pois permite desacoplar um objeto de seus clientes através do polimorfismo
- A herança de interface corretamente usada (sem eliminar partes da interface nas sub-classes) acaba criando subtipos, permitindo o polimorfismo
- Programar em função de uma interface e não em função de uma implementação (uma classe particular) permite o polimorfismo e fornece as seguintes vantagens:
  - Clientes permanecem sem conhecimento do tipo de objetos que eles usam, desde que os objetos obedeçam a interface
  - Clientes permanecem sem conhecimento das classes que implementam tais objetos
- A interface é o que há de comum entre as várias situações (os vários objetos usados pelos clientes)
  - A flexibilidade vem da possibilidade de mudar a implementação da interface, até em tempo de execução, já que o polimorfismo é implementado com "late binding" feito em tempo de execução
- Em java, uma classe pode implementar várias interfaces
  - Isso permite ter mais polimorfismo mesmo sem que as classes pertençam a uma mesma hierarquia

Exemplo no uso de interfaces

- Temos vários tipos de composites (coleções) que não pertencem a uma mesma hierarquia
  - ColeçãoDeAlunos
  - ColeçãoDeProfessores
  - ColeçãoDeDisciplinas
- Temos um cliente comum dessas coleções
  - Digamos um selecionador de objetos usado numa interface gráfica para abrir uma list box para selecionar objetos com um determinado nome
  - Exemplo:

- Quero listar todos os alunos com nome "João" e exibí-los numa list box para escolha pelo usuário
- Idem para listar professores com nome "Alfredo"
- Idem para listar disciplinas com nome "Programação"

- Queremos fazer um único cliente para qualquer uma das coleções
- O exemplo abaixo tem polimorfismo em dois lugares

```
// Como queremos usar o resultado final que contruiremos n
ComponenteDeSelecao cds =
      new ComponenteDeSelecao(umaColeçãoDeAlunos, "João");
response.out.println(cds.geraListBox());

cds = new ComponenteDeSelecao(umaColeçãoDeDisciplinas, "Pr
response.out.println(cds.geraListBox());
```

```
interface SelecionavelPorNome {
  Iterator getIteradorPorNome(String nome);
}

interface Nomeavel {
  String getNome();
}

class ColecaoDeAlunos implements SelecionavelPorNome {
  // ...
  Iterator getIteradorPorNome(String nome) {
    // ...
  }
}

class Aluno implements Nomeavel {
  // ...
  String getNome() { ... }
}

class ColeCAoDeProfessores implements SelecionAvelPorNome {
  // ...
  Iterator getIteradorPorNome(String nome) {
    // ...
  }
}
```

```
class Professor implements Nomeavel {
  // ...
  String getNome() { ... }
}

class ColecaoDeDisciplinas implements SelecionavelPorNome {
  // ...
  Iterator getIteradorPorNome(String nome) {
    // ...
  }
}

class Disciplina implements Nomeavel {
  // ...
  String getNome() { ... }
}

class ComponenteDeSelecao {
  Iterator it;
  // observe o tipo do parâmetro (uma interface)
  public ComponenteDeSelecao(SelecionavelPorNome coleção, St
    it = coleção.getIteradorPorNome(nome); // chamada polimó
  }
  // ...
  String geraListBox() {
    StringBuffer resposta = new StringBuffer();
    resposta.append("<select name=\"nome\" size=\"1\">");
    while(it.hasNext()) {
      int i = 1;
      // observe o tipo do objeto
      Nomeavel obj = (Nomeavel)it.next();
      resposta.append("<option value=\"escolha" + i + "\">"
                      obj.getNome() + // chamada polimórfica
                      "</option>");
    }
    resposta.append("</select>");
    return resposta.toString();
  }
}
```

Como achar interfaces

- Procure assinaturas repetidas
  - Exemplo: várias classes que representam coisas que podem ser vendidas indicam o uso de uma

interface VendávelIF

- Onde há delegação, um objeto se esconde atrás de outro: deve haver uma interface comum
- Procure métodos que poderiam ser usadas em aplicações semelhantes e use interfaces para que a reusabilidade das classes clientes seja maior
  - Exemplo: muitas coisas poderiam ser reserváveis, não só passagens de avião
  - Exemplo: muitas coisas pode ser alugadas, não só fitas de vídeo
  - Exemplo: muitos objetos são clonáveis
- Procure mudanças futuras (novos objetos que poderiam aparecer) e coloque as semelhanças sob controle de interfaces
- Há, entretanto, pessoas que não concordam em "pensar muito na frente"
  - Vide "Extreme Programming" ...

Exercícios para casa

- Neste exemplo, interfaces e polimorfismo poderiam ajudar a limpar o código? De que forma?
- Leia este artigo e forme sua opinião sobre o assunto: você concorda ou não com o autor? Por quê?
  - Maximize your code reusability

programa