Polimorfismo

Ling. Programação Orientada a Objetos

Prof. Leandro Fernandes

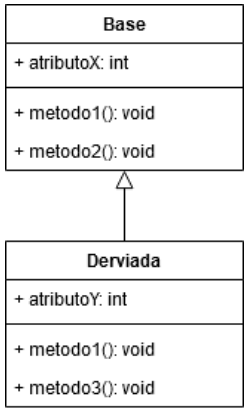
1

O que é Polimorfismo?

- Significado: *Poli* (muitas) + *morfos* (formas)
- Permite que os objetos assumam comportamentos diferentes, dependendo do tipo de instância a quem referenciam.
- Uma subclasse pode redefinir (sobrescrever) um método herdado
- O polimorfismo acontece através da recodificação de um ou mais métodos herdados por uma subclasse
- Em tempo de execução, o Java saberá qual implementação deve ser usada.

2

Conversão entre tipos: *Upcasting*

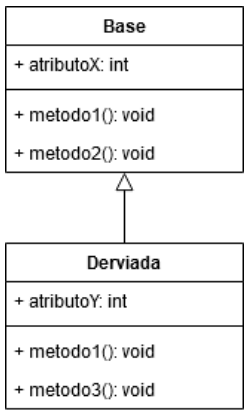


Exemplo:

- Base objB = new Derivada();
  - Declara uma referência para a classe Base, definindo a vocação de objB: referenciar qualquer instância que sejam do tipo Base
  - Em Java podemos atribuir uma instância de subclasse (derivada) a um objeto de superclasse (base). Essa operação é conhecida como *upcasting* (*up type casting*).
  - Note que essa ação é razoável e coerente com a regra "X é um Y" que temos numa relação de herança, permitindo assim atribuir a objB uma instância de Derivada.

3

Conversão entre tipos: *Upcasting*



Exemplo: Base objB = new Derivada();

Chamadas aos métodos:

- objB.metodo1();
  - É executado a versão metodo1() localizada na classe Derivada, assumindo a "forma" da instância a que objB referencia no momento.
- objB.metodo2();
  - Como não há definição de metodo2() em Derivada, será executado a versão do método dado na classe Base (como em qualquer outro caso de herança em que não há reescrita).
- objB.metodo3();
  - Erro: Embora seja uma instância de Derivada, o objeto olha pra ela com a perspectiva de Base e, portanto, não sabe da existência do metodo3().

4

Conversão entre tipos: *Upcasting*

Resumo:

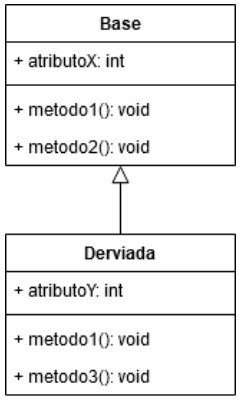
- O *upcasting* é um processo de conversão natural que ocorre entre um objeto de um tipo mais específico para um tipo mais genérico (quando há uma relação de herança que os envolve).
- É feito de maneira implícita através de atribuição de tipos compatíveis entre si (classes derivadas e superclasses).
- Note que apenas os membros mais genéricos estarão visíveis e, assim, podem ser acessados sem demandar artifícios adicionais ou estratagemas que impliquem em novas conversões.

5

E como resolver o caso *objB.metodo3*() ?

- Aqui entra um ponto interessante, é possível fazer *downcasting* ...
- Em Java é proibido atribuir um objeto de superclasse (base) a um objeto de subclasse (derivada), sem que seja feita uma conversão explícita (coerção), uma vez que não se pode assegurar qual é o tipo de instância que o objeto base referencia.
- O *downcasting* é feito utilizando o operador de casting entre classes:

```
Base objB = new ... ;
Derivada objD;
objD = (Derivada) objB;
objD.metodo3();
```



6

Conversão entre tipos: *Downcasting*

Olhando mais atentamente:

(1) Base objB = new OutraDeriv();

(2) Derivada objD;

(3) objD = (Derivada) objB;

(4) objD.metodo3();

Não há como assegurar qual é o tipo de instância que o objeto base referencia. Durante a execução do comando (3), se a instância não for do tipo da subclasse a coerção será inválida e uma *ClassCastException* será lançada.

Há um operador especial que permite verificar qual é o tipo de uma instância: *instanceof*

Sua sintaxe é a seguinte:

objeto *instanceof* Classe

Podemos resolver o problema verificando antes de converter:

(3) ***if*** (objB *instanceof* Derivada) {

(4) objD = (Derivada) objB;

(5) objD.metodo3();

(6) }

7

Considerações importantes ...

- Muitas pessoas usam o *instanceof* para identificar a qual classe fazer *downcasting* por não compreenderem adequadamente o conceito de polimorfismo.
- Observe que se há uma versão reescrita do método nas subclasses, ao invoca-lo usando um objeto base (que pode estar referenciando qualquer instância da hierarquia) a versão que será executada é a da classe correspondente.
  - Isto é, o método assumirá o comportamento que for adequado a instância em questão sem que seja necessário ficar codificando conversões manualmente.
- Veja que o uso de métodos abstratos na classe superclasse obrigam que todas as classes derivadas implementem suas próprias versões daquele método.

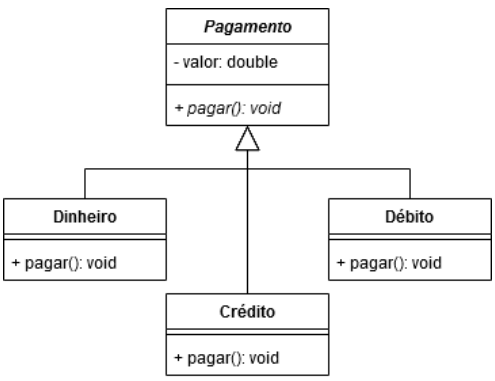
8

Classes Abstratas (relembrando)

- Se uma classe possui algum método sem implementação (abstrato), o modificador **abstract** deve preceder sua declaração.
- Não podem ser usadas para instanciar objetos (pois estão "incompletas" apresentando comportamentos não implementados).
- Devem ser vistas como *templates* para subclasses que irão dar uma implementação aos métodos abstratos
- Exemplo:
  - A classe abstrata Pagamento tem um método abstrato pagar(...)
  - As classes concretas PagamentoVisa, PagamentoCheque, PagamentoDinheiro implementam o comportamento do método pagar(...)

9

Polimorfismo + Classes Abstratas



- A definição do método abstrato pagar() impõe a todas as classes derivadas a necessidade de sua implementação.
- Isso também permite que, tendo um objeto do tipo Pagamento, o método pagar() seja invocado independentemente da instância referenciada por ele.
- Por fim, o fato da classe Pagamento ser abstrata inibe a possibilidade de criarmos uma instância direta do tipo Pagamento.

10