Polimorfismo

Prof. Ricardo P. Mesquita

Polimorfismo?

- Permite "programar no geral" em vez de "programar no específico"
- Permite escrever programas que processam objetos que compartilham a mesma superclasse (direta ou indiretamente) como se todos fossem objetos da superclasse
  - *Isso pode simplificar a programação*!
- Podemos projetar e implementar sistemas que são facilmente extensíveis…

Exemplo

- Se a classe Retângulo for derivada da classe Quadrilátero, então um objeto Retângulo é uma versão mais específica de um Quadrilátero.
- Qualquer operação que pode ser realizada em um Quadrilátero também pode ser realizada em um Retângulo
- Essas operações também podem ser realizadas em outros quadriláteros (como quadrados, paralelogramos, trapezóides)
- *O Polimorfismo ocorre quando um programa invoca um método por meio de uma superclasse variável Quadrilátero – em tempo de execução, a versão de subclasse correta do método é chamada, com base no tipo de referência armazenada na variável de superclasse.*

Observações

- O polimorfismo permite tratar as generalidades e deixar que as especificidades sejam tratadas em tempo de execução. Você pode instruir objetos a se comportarem de maneiras apropriadas para esses objetos, sem nem mesmo conhecer seus tipos (contanto que os objetos pertençam à mesma hierarquia de herança).

Observações

- O polimorfismo promove extensibilidade: o software que invoca o comportamento polimórfico é independente dos tipos de objeto para os quais as mensagens são enviadas. Novos tipos de objetos que podem responder a chamadas de método existentes podem ser incorporadas a um sistema sem modificar o sistema básico. Somente o código de cliente que instancia os novos objetos deve ser modificado para acomodar os novos tipos.

Exemplo Prático

- Acesse:
  - (1)BasePlusCommissionEmployee.java
  - (1)CommissionEmployee.java
  - (1)PolymorphismTest.java

Atenção: corrija os nomes dos arquivos!!

Exemplo Prático

- Saída:

```
Call CommissionEmployee's toString with superclass reference to superclass object:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 10000.00
commission rate: 0.06

Call BasePlusCommissionEmployee's toString with subclass reference to
subclass object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00

Call BasePlusCommissionEmployee's toString with superclass reference to
subclass object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 300.00
```

Classes e Métodos Abstratos

- São classes para as quais não são criados objetos
- São usadas apenas como superclasses em hierarquias de herança
- Não podem ser usadas para instanciar objetos (essas classes são incompletas)
  - As subclasses declaram as "partes ausentes" para aí então tornarem-se classes "concretas" – e daí instanciar objetos

Classes e Métodos Abstratos

- O propósito de uma classe abstrata é fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e assim podem compartilhar um design comum.
- Declaração:
  - Use a palavra-chave **abstract**
  - Uma classe abstrata contém, normalmente, um ou mais métodos abstratos (possui a palavra-chave **abstract** em sua declaração)
  - Ex:

```
public abstract void draw();
```

Classes e Métodos Abstratos

- Métodos abstratos não fornecem implementações.
- *Uma classe que contém métodos abstratos deve ser declarada como uma classe abstrata mesmo se essa classe contiver alguns métodos concretos (não abstratos)*
- Cada subclasse concreta de uma superclasse abstrata também deve fornecer implementações concretas de cada um dos métodos abstratos da superclasse.
- Construtores e métodos **static** não podem ser declarados **abstract**
  - *Construtores não são herdados, portanto um construtor **abstract** jamais seria implementado.*

Classes e Métodos Abstratos

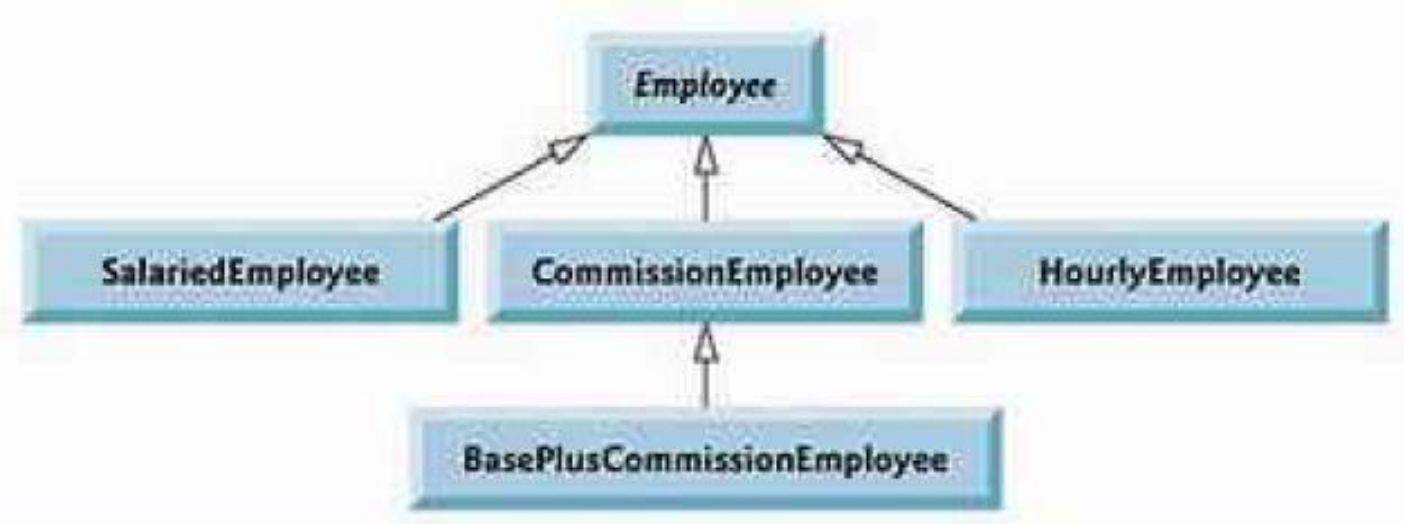
- Embora métodos **static** não **private** sejam herdados, eles não podem ser sobrescritos! Como métodos **abstract** se destinam a serem sobrescritos, não faz sentido declarar um método **static** como **abstract**.

Observações

- Uma classe abstrata declara atributos e comportamentos comuns (abstratos e concretos) das várias classes em uma hierarquia de classes.
- Em geral, uma classe abstrata contém um ou mais métodos abstratos que as subclasses podem sobrescrever.
- Variáveis de instância e métodos concretos de uma classe abstrata estão sujeitos às regra normais de herança.
- Tentar instanciar um objeto de uma classe abstrata gera um erro de compilação.
- A falha em implementar os métodos abstratos de uma superclasse em uma subclasse é um erro de compilação, a menos que a subclasse também seja **abstract**.

Exemplo

- Folha de pagamento usando polimorfismo
- Hierarquia Employee



Exemplo

- Acesse:
  - (2)BasePlusCommissionEmployee.java
  - (2)CommissionEmployee.java
  - (2)Employee.java
  - (2)HourlyEmployee.java
  - (2)PayrollSystemTest.java
  - (2)SalariedEmployee.java

Atenção: corrija os nomes dos arquivos!!

Exemplo

- A saída do programa:

```
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
earned: $500.00

Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned $800.00
```

Exemplo

- Continuação

```
hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00; commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00; commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
```

Atenção

- Atribuir uma variável de superclasse a uma variável de subclasse (sem coerção explícita) gera um erro de compilação.
- Se, em tempo de execução, a referência de um objeto de subclasse tiver sido atribuída a uma variável de uma das suas superclasses diretas ou indiretas, é aceitável fazer *downcast* da referência armazenada nessa variável de superclasse de volta a uma referência do tipo da subclasse.
  - *Antes de realizar essa coerção, utilize o operador* ***instanceof*** *para assegurar que o objeto é de fato um objeto de um tipo de subclasse apropriado!*

Vinculação Dinâmica ou Tardia

- Ocorre quando o Java decide, dinamicamente, em tempo de execução, qual método é adequado para a chamada.
  - "Tardia" ~ depois da compilação

Métodos e Classes Final

- Métodos **final** não podem ser sobrescritos
- Métodos declarados **private** e **static** são implicitamente **final**
  - *Porque não é possível sobrescrevê-los em uma subclasse*.
- Uma declaração de método **final** nunca pode mudar, assim, todas as subclasses utilizam a mesma implementação do método
- Chamadas a métodos **final** são resolvidas em tempo de compilação (***vinculação estática***)

Métodos e Classes Final

- Uma classe **final** não pode ser superclasse (uma classe não pode estender uma classe **final**.
- Todos os métodos de uma classe **final** são implicitamente **final**.
- *Tornar uma classe final impede que programadores criem subclasses que possam driblar as restrições de segurança*.

Interfaces

- Permite que classes não relacionadas implementem um conjunto de métodos comuns (por exemplo, um método que calcula o valor de um pagamento)
- Definem e padronizam como coisas, pessoas e sistemas podem interagir entre si.
- Uma **declaração de interface** inicia com a palavra-chave **interface** e contém somente constantes e métodos **abstract**.
- Todos os membros de uma interface devem ser **public**.
- Interfaces não podem especificar nenhum detalhe de implementação.

Interfaces

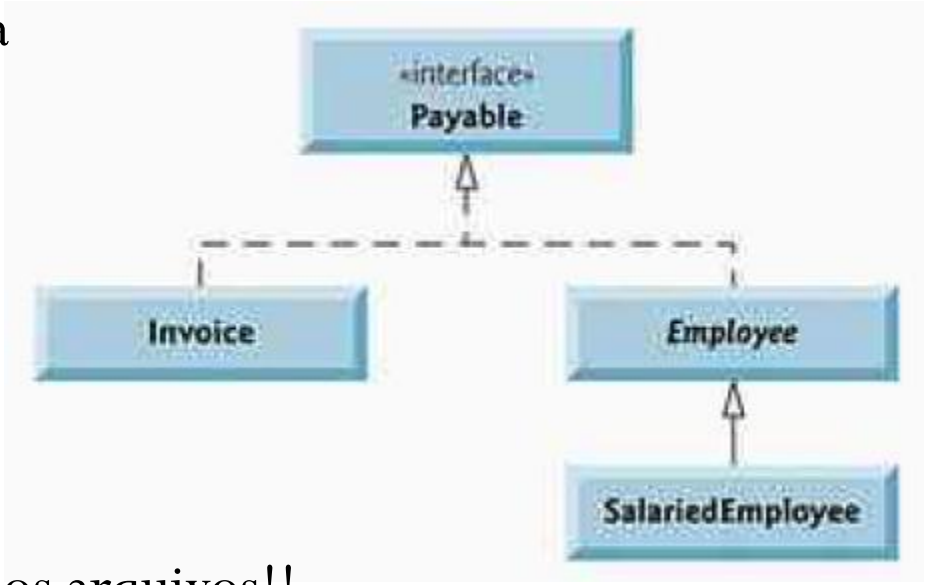
- Todos os métodos de uma interface são implicitamente **public abstract**
- Para utilizar uma interface, uma classe concreta deve especificar que ela **implementa** a interface e deve declarar cada método da interface com a assinatura especificada na declaração de interface.
- **Atenção**: *falhar em implementar qualquer método de uma interface em uma classe concreta que implementa a interface resulta em um erro de compilação indicando que a classe deve ser declarada* ***abstract***.

Interfaces

- Uma interface é geralmente utilizada quando classes díspares precisam compartilhar métodos e constantes comuns. Isso permite que objetos de classes não relacionados sejam processados polimorficamente.
  - *Objetos de classe que implementam a mesma interface podem responder às mesmas chamadas de método.*
- Uma interface costuma ser utilizada no lugar de uma classe **abstract** quando não há nenhuma implementação padrão a herdar
- Relacionamento classe-interface: *realização*

Exemplo

- Acesse:
  - (3)Employee.java
  - (3)Invoice.java
  - (3)Payable.java
  - (3)PayableInterfaceTest.java
  - (3)SalariedEmployee.java



Atenção: corrija os nomes dos arquivos!!