
ORACLE
Academy

Java Fundamentals

7-5
Polimorfismo

ORACLE
Academy

Objetivos

- Esta aula abrange os seguintes tópicos:
  - Aplicar referências de superclasses aos objetos de subclasse
  - Escrever código para sobrepor métodos
  - Usar envio do método dinâmico para dar suporte ao polimorfismo
  - Criar métodos e classes abstratos
  - Reconhecer uma sobreposição do método correto

Visão geral

- Esta aula abrange os seguintes tópicos:
  - Usar o modificador final
  - Explicar a finalidade e a importância da classe Object
  - Escrever código para um applet que exiba dois triângulos de cores diferentes
  - Descrever referências do objeto

Revisão de Herança

- Quando uma classe herda de outra, a subclasse “é-um(a)” tipo da superclasse
- Objetos de uma subclasse podem ser referenciados usando uma referência de superclasse, ou tipo

Saiba Mais

- Visite as páginas de tutorial da Oracle para saber mais:
- Herança:
  - http://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html
- Polimorfismo:
  - http://docs.oracle.com/javase/tutorial/java/IandI/Herança.html

Exemplo de Herança

- Se as classes forem criadas para uma classe Bicycle e uma classe RoadBike que estende Bicycle, uma referência do tipo Bicycle pode referenciar um objeto RoadBike (veja abaixo)
  - Como RoadBike “é-um(a)” tipo de Bicycle, é perfeitamente legítimo armazenar um objeto RoadBike como uma referência de Bicycle
  - O tipo de uma variável (ou referência) não determina o tipo real do objeto ao qual ela se refere

Exemplo de Herança

- Portanto, uma referência de Bicycle, ou variável, pode ou não conter um objeto da superclasse Bicycle, pois ela pode conter qualquer subclasse de Bicycle

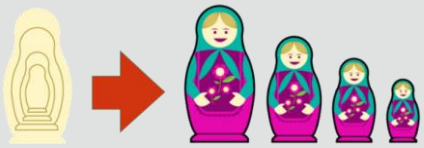
```
Bicycle bike = new RoadBike();
```

Polimorfismo

- Quando uma variável ou referência pode se referir a diferentes tipos de objetos, isso é chamado de polimorfismo
- Polimorfismo é um termo que significa “muitas formas”
- No caso de programação, o polimorfismo permite que variáveis se refiram a muitos diferentes tipos de obetos, ou seja, elas podem ter várias formas
- Por exemplo, como RoadBike “é-um(a)” Bicycle, há duas referências possíveis que definem o tipo de objeto que ela é (Bicycle ou RoadBike)

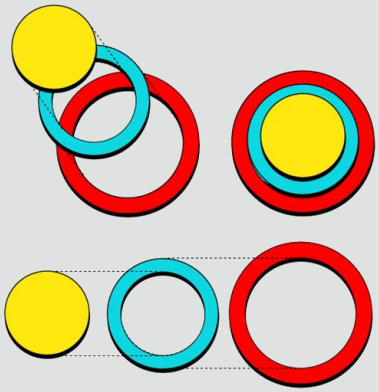
Polimorfismo e Bonecas de Aninhamento

- O polimorfismo pode ser visualizado como um conjunto de bonecas de aninhamento: O conjunto de bonecas compartilham um tipo e aparência, mas são todas únicas de certa forma
- Cada boneca tem a mesma forma, com um tamanho determinado pela boneca onde ela deve se encaixar
- Cada boneca menor é armazenada dentro da próxima boneca maior
- Do lado de fora, você não vê as bonecas menores, mas é possível abrir cada uma para encontrar uma boneca menor



Superclasses e Subclasses

- De forma semelhante, as subclasses podem “caber” dentro do tipo de referência de uma superclasse
- Uma variável de superclasse pode conter, ou armazenar, o objeto de uma subclasse, enquanto se parece e se comporta como a superclasse



ORACLE Academy

Esse conceito nos permite contornar a limitação de que matrizes só podem conter um tipo de dados. Uma matriz de Bicicletas pode conter os objetos Bicicleta e RoadBike, pois RoadBike "É uma" Bicicleta. Levado ao extremo, uma matriz de Objetos pode, na verdade, conter qualquer tipo de objeto, uma vez que Objeto é a superclasse definitiva.

Variáveis de Superclasse

- Se você “abrir” uma variável de superclasse, ou chamar um de seus métodos, descobrirá que você tem na verdade um objeto de subclasse armazenado
- Por exemplo
  - Com as bonecas de aninhamento, não é possível ver a boneca menor até você abrir a boneca maior
  - O tipo dela pode ser ambíguo
  - Quando o código Java é compilado, o Java não verifica o tipo (supertipo ou subtipo) de objeto que está dentro de uma variável
  - Quando o código Java for executado, o Java “abrirá” para ver que tipo de objeto está dentro da referência, e chamará os métodos que forem desse tipo

Incerteza ao Referenciar Objetos

- Existem benefícios na incerteza ao referenciar diferentes objetos na hora da compilação
- Por exemplo:
  - Você escreve um programa que calcula os diferentes comprimentos de tubos do quadro de uma bicicleta, de acordo com as medidas do ciclista e o tipo de bicicleta desejada (RoadBike ou MountainBike)
  - Você deseja uma lista para acompanhar quantos objetos de bicicleta você criou
  - Você quer apenas uma lista, e não duas listas separadas para cada tipo de bicicleta
  - Como você cria essa lista? Um array, talvez?
  - Qual é o problema em usar um array para criar a lista?

Por Que Não Usar um Array de Objetos de Classe?

- Arrays são conjuntos de elementos do mesmo tipo, como um conjunto de números inteiros, duplos ou Bicycles (Bicicletas)
- Embora seja possível ter um array de objetos, eles devem ser do mesmo tipo
- Isso funciona para classes que não são estendidas

Por Que Não Usar um Array de Objetos de Classe?

- O polimorfismo resolve este problema
- Como RoadBike e MountainBike são tipos de objetos Bicycle, use um array de referências de Bicycle para armazenar a lista de bicicletas que você criou
- Qualquer um dos dois tipos de Bicycle pode ser adicionado a este array

```
Bicycle[] bikes = new Bicycle[size];
```

Referências de Objetos

- A classe Object é a maior superclasse em Java, pois ela não estende outra classe
- Consequentemente, qualquer classe pode ser armazenada em uma referência de Object

```
Object[] objects = new Bicycle[size];
```

- Neste exemplo de array, é válido também armazenar nossas bicicletas em um array de referências de Object
- No entanto, isso torna nosso tipo de array ainda mais ambíguo, e deve ser evitado a menos que haja um motivo para isso

Sobrepondo Métodos do Objeto

- Como Object é a superclasse de todas as classes, sabemos que todas as classes herdam métodos de Object
- Dois desses métodos são muito úteis, como o método equals() e o método toString()
- O método equals() permite que verifiquemos se duas referências estão consultando o mesmo objeto
- O método toString() retorna uma String que representa o objeto
- A String fornece informações básicas sobre o objeto, como a classe, o nome e um hashcode único

Substituindo ou Redefinindo Métodos

- Embora os métodos equals() e toString() sejam úteis, eles não possuem funcionalidade para o uso mais específico
- Para a classe Bicycle, devemos gerar uma String contendo o número do modelo, a cor, o tipo de quadro e o preço
- O uso do método de Object não retornará essas informações
- O método toString() na classe Object retorna uma representação de String do local do objeto na memória
- Em vez de criar um método com outro nome, podemos substituir o método toString() e redefini-lo para adequar às nossas necessidades

Substituindo Métodos

- A substituição de métodos é uma forma de redefini-los com o mesmo tipo de retorno e parâmetros adicionando, ou substituindo a lógica existente, em um método de subclasse
- Substituir é diferente de sobrecarregar um método
- Sobrecarregar um método significa que o programador mantém o mesmo nome (por exemplo, toString()), mas altera os parâmetros de entrada (assinatura do método)
- A substituição basicamente oculta o método da classe pai com a mesma assinatura, e não será chamado em um objeto de subclasse, a menos que a subclasse use a palavra-chave super

Substituindo Métodos

- A substituição não altera os parâmetros
- Altera somente a lógica dentro do método definido na superclasse

Tecnicamente, um método de substituição na subclasse pode, com algumas limitações, ter um tipo de retorno diferente, desde que os tipos e o número de parâmetros sejam os mesmos. Essa prática é desencorajada.

Tutoriais Java sobre Substituição de Métodos

- Visite os tutoriais Java da Oracle para obter mais informações sobre como substituir métodos:
  - http://docs.oracle.com/javase/tutorial/java/IandI/override.html

Substituindo toString()

- Podemos substituir toString() para retornar uma String que forneça informações sobre o objeto em vez do local do objeto na memória
  - Primeiro, comece com o protótipo:

```
public String toString()
```

  - Não há motivo para alterar o tipo de retorno ou os parâmetros; portanto, substituiremos toString()
  - De acordo com nossos dados privados (número do modelo, cor, tipo de quadro e preço), podemos retornar a seguinte String:

```
return "Model: " + modelNum + " Color: " + color +
       " Frame Type: " + frameType + " Price: " + price;
```

Substituindo toString()

- O resultado do nosso método toString() substituído:

```
public String toString(){
  return "Model      : " + modelNum +
         "Color      : " + color +
         "Frame Type : " + frameType +
         "Price      : " + price;
}//fim do método toString
```

- É muito comum e muito útil ao criar classes Java para substituir o método toString() para testar os método e dados

Noções Básicas sobre o Modelo do Objeto

- Polimorfismo, assim como a herança, é essencial para o modelo do objeto e para a programação orientada por objetos
- O polimorfismo fornece versatilidade ao trabalhar com objetos e referências, mantendo os objetos discretos ou distintos
- No âmago da filosofia, o modelo de objeto transforma programas em um conjunto de objetos em relação a um conjunto de tarefas, encapsulando os dados e criando peças menores de um programa, em vez de um único grupo grande de códigos

Metas do Modelo do Objeto

- O modelo do objeto tem várias metas:
  - Abstração de dados
  - Proteger informações e limitar a capacidade de outras classes de alterar ou corromper dados
  - Ocultar implementação
  - Fornecer código modular que pode ser reutilizado por outros programas ou classes

Polimorfismo e Métodos

- Como os métodos da subclasse são afetados pelo polimorfismo?
- Lembre-se, as subclasses podem herdar métodos de suas superclasses
- Se uma variável de Bicicleta puder conter o tipo de objeto de uma subclasse, como o Java sabe quais métodos chamar quando um método substituído é chamado?

Polimorfismo e Métodos

- Os métodos chamados em uma referência (bike) sempre se referirão a métodos dentro do tipo do objeto (RoadBike)

```
Bicycle bike = new RoadBike();
```

- Imagine que nossa classe Bicicleta contenha um método setColor(Color color) para definir a cor da bicicleta que o ciclista deseja.
- RoadBike herda esse método
- O que acontece quando fazemos o seguinte?

```
bike.setColor(new Color(0, 26, 150));
```


A primeira linha de código não é muito prática, mas ilustra o conceito. A aplicação prática seria com uma matriz de objetos Bicicleta, sendo alguns RoadBikes.

Envio do método dinâmico

- Java é capaz de determinar qual método deve ser chamado com base no tipo do objeto que está sendo referido no momento em que o método é chamado
- Envio do método dinâmico, também conhecido como Ligação Dinâmica, permite que o Java determine correta e automaticamente qual método será chamado com base no tipo de referência e no tipo de objeto

Isso é feito no tempo de execução.

Classes Abstratas

- É realmente necessário definir uma classe Bicycle se só vamos criar objetos de suas subclasses:
  - roadBikes
  - mountainBikes?
- As classes abstratas são uma alternativa que trata desta questão
- Uma classe abstrata é aquela que não pode ser instanciada:
  - Isso significa que não é possível criar objetos desse tipo
  - É possível criar variáveis ou referências desse tipo

Pode ser difícil para os alunos iniciantes em Programação Orientada a Objetos ou OOP entender o valor das classes abstratas, classes finais e outros tópicos relacionados. Eles dirão: "Mas eu não vou criar nenhuma Bicicleta. Por que tornar a classe abstrata?" Não há resposta fácil, mas o resumo é que reforça o design geral que os arquitetos de Java criaram. Ao tornar a Bicicleta abstrata, outro programador não fará objetos Bicicleta puros e "quebrará" a intenção do designer do programa original.

Classes Abstratas

- Se declararmos a classe Bicicleta como abstrata, poderemos ainda usar a sintaxe abaixo, mas não poderemos criar um objeto de Bicycle
- Isso significa que todas as referências do tipo Bicycle farão referência a objetos MountainBike ou RoadBike da subclasse

```
Bicycle bike  = new RoadBike();
```

```
Bicycle bike2 = new mountainBike();
```

Classes Abstratas

- As classes abstratas podem conter métodos totalmente implementados que elas “repassam” para qualquer classe que os estenda
- Torne uma classe abstrata usando a palavra-chave abstract

```
public abstract class Bicycle
```

Classes Abstratas

- As classes abstratas também podem declarar pelo menos um método abstrato (método que não contém nenhuma implementação)
- Isso significa que as subclasses devem usar o protótipo do método (perfil) e devem implementar esses métodos
- Métodos abstratos são declarados com a palavra-chave abstract

```
abstract public void setPrice();
```

Declare como público abstrato. Não usar {}.

Métodos Abstratos

- Métodos abstratos:
  - Não podem ter um corpo de método
  - Devem ser declarados em uma classe abstrata
  - Devem ser substituídos em uma subclasse
- Isso força os programadores a implementarem e redefinirem métodos
- Normalmente, as classes abstratas contêm métodos abstratos, métodos parcialmente implementados ou métodos totalmente implementados

Métodos Parcialmente Implementados

- Lembre-se de que as subclasses podem chamar o construtor e os métodos da superclasse usando a palavra-chave super
- Com classes abstratas, as subclasses também podem usar 'super' para usar o método da superclasse
- Normalmente, isso é feito primeiro substituindo o método da superclasse, chamando o método super ou substituído e, em seguida, adicionando o código
- Por exemplo
  - Vamos substituir o método equals() da classe abstrata Bicycle, que é parcialmente implementada
  - Isso significa que o método equals() na Bicycle não é abstrato

Métodos Parcialmente Implementados

- Isso compara dois objetos de Bicycle com base no preço e no número do modelo
  - Note que este método substitui o método equals() de Object porque ele tem os mesmos parâmetros e tipo de retorno

```
public boolean equals(Object obj) {
   if(this.price == obj.price &&
      this.modelNum == obj.modelNum) {
      return true;
   }
   else {
      return false;
   }//fim if
}//fim do método equals
```

Métodos Parcialmente Implementados

- Podemos substituir o método da nossa subclasse MountainBike e procurar equivalência em outros atributos

```
public boolean equals(Object obj) {
    if(super.equals(obj)) {
       if(this.suspension == obj.suspension)
       return true;
    }//fim if
    return false;
}//fim do método equals
```

Transformando Classes Abstratas em Subclasses

- Ao herdar de uma classe abstrata, você deve executar um dos seguintes procedimentos:
  - Declarar a classe filha como abstrata
  - Substituir todos os métodos abstratos herdados da classe pai
  - Caso não execute esses procedimentos, ocorrerá um erro no momento da compilação

Usando Final

- Embora seja bom ter a opção, em alguns casos, talvez você não queira que alguns métodos sejam substituídos ou tenham sua classe estendida
- O Java fornece uma ferramenta para evitar que os programadores substituam métodos ou criem subclasses:
  - a palavra-chave final

Usando Final

- Um bom exemplo é a classe String
- É declarado que:

```
public final class String {}
```

- Programadores vão se referir a classes como essa como imodificáveis, ou seja, ninguém pode estender String nem modificar ou substituir seus métodos

Usando Final

- O modificador final pode ser aplicado a variáveis
- As variáveis de final não podem alterar os valores após os mesmos serem inicializados

Essas são constantes. Os objetos podem ser finais também, mas provavelmente não da maneira que os alunos intuirão. Consulte o slide 43.

Usando Final

- As variáveis de final podem ser:
  - Campos de classes
    - Campos finais com expressões constantes do tempo de compilação são variáveis constantes
    - Static pode ser combinada com final para criar uma variável sempre disponível e nunca alterada
  - Parâmetros do método
  - Variáveis locais

Ao lidar com valores como PI e outras constantes matemáticas e científicas, etc., é uma boa ideia atribuí-los como variáveis finais estáticas.

Usando Final

- As referências de final devem sempre fazer referência ao mesmo objeto
- O objeto ao qual a variável está fazendo referência não pode ser alterado
- O conteúdo desse objeto pode ser modificado
- Visite o tutorial Java da Oracle para obter mais informações sobre como usar a palavra-chave final:
  - http://docs.oracle.com/javase/tutorial/java/IandI/final.html

Código de Applet de Triângulo

- O código a seguir mostra as etapas envolvidas ao escrever um applet com dois triângulos de cores diferentes

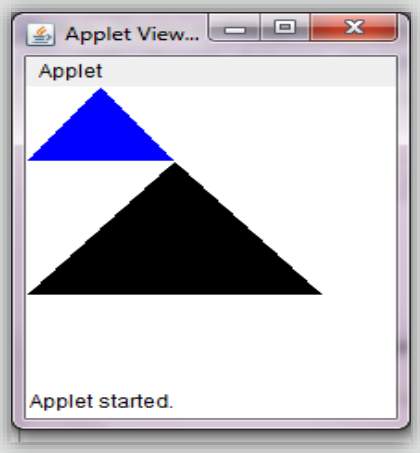
```
public class TrianglesApplet extends Applet{
   public void paint(Graphics g){
      int[] xPoints = {0, 40, 80};
      int[] yPoints = {50, 0, 50};
      g.setColor(Color.blue);
      g.fillPolygon(xPoints, yPoints, 3);
      int[] x2Points = {80, 160, 0};
      int[] y2Points = {50, 140, 140};
      g.setColor(Color.black);
      g.fillPolygon(x2Points, y2Points, 3);
   }//fim do método paint
}//fim da classe TrianglesApplet
```

Código de Applet de Triângulo Explicado

- Etapa 1:
  - Estenda a classe Applet para herdar todos os métodos, incluindo o paint
- Etapa 2:
  - Substitua o método paint para incluir os triângulos
- Etapa 3:
  - Desenhe o triângulo usando o método fillPolygon herdado
- Etapa 4:
  - Desenhe o 2º triângulo usando o método fillPolygon herdado
- Etapa 5:
  - Execute e compile o código

Imagem do Applet de Triângulo

- O código do Applet de Triângulo exibe a seguinte imagem:



Terminologia

- Os principais termos usados nesta lição foram:
  - abstract
  - Envio do método dinâmico
  - final
  - Imutável
  - Sobrecarregando métodos
  - Substituindo métodos
  - Polimorfismo

Resumo

- Nesta aula, você deverá ter aprendido a:
  - Aplicar referências de superclasses aos objetos de subclasse
  - Escrever código para sobrepor métodos
  - Usar envio do método dinâmico para dar suporte ao polimorfismo
  - Criar métodos e classes abstratos
  - Reconhecer uma sobreposição do método correto
  - Usar o modificador final
  - Explicar a finalidade e a importância da classe Object
  - Escrever código para um applet que exiba dois triângulos de cores diferentes
  - Descrever referências do objeto


ORACLE
Academy