Programação Orientada a Objetos

Aula 13 - Atributos e Métodos Estáticos, Classes Abstratas e Interfaces

Apresentação

Até agora, para efeitos de projeto de sistemas, você viu classes (concretas) e objetos, unicamente. Porém, é importante que você saiba que há recursos mais sofisticados na linguagem Java que permitem que a gente projete de forma mais genérica e aproximada o mundo real diante de situações que ainda não foram apresentadas nos exemplos anteriores, mas que farão sentido à medida que formos observando as sutilezas que envolvem a arte de desenvolver soluções em Tecnologias de Informação (TI)!

Para tanto, teremos a apresentação de atributos e métodos comuns a todos objetos de uma classe, os chamados atributos e métodos estáticos. Veremos também classes da qual não podemos instanciar objetos diretamente, as chamadas classes abstratas.

Finalmente, conheceremos o conceito de interface em Java, que representa um "contrato" assumido por classes e que as obrigam a implementar um determinado conjunto de métodos de uma determinada forma. Lembre que você vai realizar práticas através dos exercícios apresentados.

Vídeo 01 - Apresentação

Objetivos

Ao final desta aula, você será capaz de:

- Compreender os conceitos que motivam a utilização das técnicas de: atributos e métodos estáticos, classes abstratas e interfaces.
- Saber a aplicação e contextualização desses conceitos.

Atributos Estáticos

Vídeo 02 - Atributos e Métodos Estáticos

Primeiramente, vejamos essa denominação: “atributo estático”. Nas aulas anteriores, abordamos de forma exaustiva o conceito de atributos, os quais são variáveis declarados nas classes, e que possuem tipo e nome. Você viu também que os valores dos atributos são definidos pelos próprios objetos instanciados de uma dada classe. Essa atribuição pode ocorrer de forma direta (caso o atributo seja público), ou através de métodos *setters* que permitem acessá-los (que é o caso dos atributos que são privados), ou mesmo através dos próprios construtores da classe.

Até aí, nenhuma novidade, correto?

Agora, vejamos a denominação “estático”. O que você compreende por esse termo? Algo imóvel, imutável, paralisado?

Pois bem, é aí que “mora o perigo”, pois o termo escolhido para tal variável não foi tão adequado, e dá tipicamente a ideia de uma constante.

> Atributo Estático ≠ Atributo Constante

A ideia por trás do conceito de atributos e métodos estáticos é a de que objetos de uma mesma classe podem e, em algumas situações devem, compartilhar valores em comum, caso contrário, teríamos que criar atributos que precisariam ser atualizados todos ao mesmo tempo, em cada objeto criado.

Vejamos o seguinte exemplo. Um sistema que controle relógios de ponto de várias filiais com o mesmo fuso horário, nessa situação, quando tivermos que fazer um ajuste ou atualizar a hora, minutos e segundos, teremos que fazê-lo em todos os relógios. Porém, se esse atributo não “pertencer” aos objetos, e sim à própria classe, todos saberão a hora certa, pois se baseiam no mesmo valor. Observe na Listagem 1 o código da classe RelogioPonto.

```
1  public class RelogioPonto{
2     public static int hora; // atributo estático
3     public static int minuto; // atributo estático
4     public static int segundo; // atributo estático
5
6     private int id;
7     private String nomeFilial;
8
9     public RelogioPonto(){
10    }
11
12    public RelogioPonto(int ident, String filial){
13       this.id = ident;
14       this.nomeFilial = filial;
15    }
16
17    public int getId(){
18       Return id;
19    }
20
21    public void setId(int id){
22       this.id = id;
23    }
24
25    public String getNomeFilial(){
26       return nomeFilial;
27    }
28
29    public void setNomeFilial(String nomeFilial){
30       this.nomeFilial = nomeFilial;
31    }
32 }
```

Listagem 1 - Classe RelogioPonto

Veja o uso da palavra reservada *static* na declaração dos atributos hora, minuto e segundo. Essa faz com que o compilador entenda que não será necessário reservar um espaço de memória em cada objeto da classeRelogioPonto para armazenar o valor da hora, minuto e segundo, pois esses valores serão

compartilhados por todos os objetos dessa classe. Logo, será necessário apenas um único espaço para armazenar esse valor, juntamente com os códigos dos métodos da classe.

Vamos criar uma aplicação simples para exemplificar o uso dos atributos estáticos hora, minuto e segundo. A Listagem 2 apresenta o código de tal aplicação.

```
1  public class RelogioPontoMain{
2      public static void main(String[] args){
3          RelogioPonto filialCentro = new RelogioPonto(1, "Centro");
4          RelogioPonto filialAlecrim = new RelogioPonto(2, "Alecrim");
5          RelogioPonto filialRocas = new RelogioPonto(3, "Rocas");
6
7          filialAlecrim.hora = 7;
8          filialAlecrim.minuto = 45;
9          filialAlecrim.segundo = 30;
10
11         System.out.println(
12             "Filial Alecrim: " +filialAlecrim.hora+ ":" +filialAlecrim.minuto+":" + filialAlecrim.segundo);
13         System.out.println(
14             "Filial Centro: " +filialCentro.hora+ ":" +filialCentro.minuto+":" + filialCentro.segundo);
15         System.out.println(
16             "Filial Rocas: " +filialRocas.hora+ ":" +filialRocas.minuto+":" + filialRocas.segundo);
17
18     }
19 }
```

Listagem 2 - Aplicação usando a classe RelogioPonto

Observe que criamos 3 (três) objetos RelogioPonto e na sequência pegamos um desses (filialAlecrim) para atribuir os valores de hora, minuto e segundo. Logo, em seguida, imprimimos os valores dos três objetos e teremos o seguinte resultado:

```
1  filial Alecrim: 7:45:30
2  filial Centro: 7:45:30
3  filial Rocas: 7:45:30
```

Isso ocorre, pois, quando atribuímos valores a variáveis estáticas através de um objeto, estamos atribuindo o valor para todos os objetos da mesma classe. Por isso, alguns chamam os atributos estáticos de **atributos de classe**. Para não sermos redundantes, faremos uma substituição da instrução.

De...

```
filialAlecrim.hora = 7;
filialAlecrim.minuto = 45;
filialAlecrim.segundo = 30;
```

Para...

```
RelogioPonto.hora = 7;
RelogioPonto.minuto = 45;
RelogioPonto.segundo = 30;
```

Pois, quando estamos tratando de atributos de classe (atributos estáticos) em Java, é comum e é uma boa prática usarmos o seu valor utilizando o nome da classe seguido do operador ponto "." e o atributo que desejarmos.

Observe que, com essa mudança, mostramos que não é necessário criar um objeto da classe RelogioPonto para que se possa acessar seus atributos estáticos. Quando estáticos, eles estão disponíveis antes mesmo de se criar o objeto.

Com isso, temos como análise geral o seguinte quadro:

> Atributo Estático = Atributo Comum ou Compartilhado

Métodos Estáticos

A mesma ideia dos atributos estáticos pode ser ampliada e aplicada para os métodos, pois, se quisermos alterar o modo de acesso dos atributos hora, minuto e segundo, tornando-os privados (private), por exemplo, teremos que criar métodos que sejam capazes de alterá-los.

Não é uma boa prática modificar valores de atributos estáticos através de referências para seus objetos. Imagine 10 objetos tentando alterar os valores compartilhados, a probabilidade do sistema se tornar caótico (ou dar algum erro) é muito grande. Os métodos estáticos surgiram basicamente para operarem sobre os atributos estáticos, ou que não realizam operação alguma sobre os atributos dos objetos. Ou seja, não é do contexto do objeto, mas, sim, de sua classe.

Alterando o exemplo apresentado anteriormente (Listagem 1) para trabalhar com métodos estáticos, chegaremos ao exemplo apresentado na Listagem 3. Observe o uso da palavra reservada static na declaração dos métodos que operam sobre os atributos estáticos.

```
1  public class RelogioPonto{
2  public static int hora; // atributo estático
3  public static int minuto; // atributo estático
4  public static int segundo; // atributo estático
5
6  private int id;
7  private String nomeFilial;
8
9  public RelogioPonto(){
10 }
11
12 public RelogioPonto(int ident, String filial){
13    this.id = ident;
14    this.nomeFilial = filial;
15 }
16
17 public static int getHora(){
18    return hora;
19 }
20
21 public static void setHora(int hora){
22    RelogioPonto.hora = hora;
23 }
24
25 public static int getMinuto(){
26    return minuto;
27 }
28
29 public static void setMinuto(int minuto){
30    RelogioPonto.minuto = minuto;
31 }
32
33 public static int getSegundo(){
34    return segundo;
35 }
```

```
    public static void setSegundo(int segundo){
        RelogioPonto.segundo = segundo;
    }
    public int getId(){
        Return id;
    }
    public void setId(int id){
        this.id = id;
    }
    public String getNomeFilial(){
        return nomeFilial;
    }
    public void setNomeFilial(String nomeFilial){
        this.nomeFilial = nomeFilial;
    }
}
```

Listagem 3 - Classe RelogioPonto com métodos estáticos

Agora, mais uma alteração deve ser feita na **Listagem 2** , de...

```
RelogioPonto.hora = 7;
RelogioPonto.minuto = 45;
RelogioPonto.segundo = 30;
```

Para...

```
RelogioPonto.setHora(7);
RelogioPonto.setMinuto(45);
RelogioPonto.setSegundo(30);
```

Com isso, temos a utilização dos métodos estáticos diretamente da classe RelogioPonto. Assim como no caso de atributos, também não é necessário criar o objeto da classe para que se possa usar o método.

Atividade 01

1. Crie uma classe para representar Livros em uma biblioteca, com os seguintes campos: título, autor, gênero, editora e edição. Assuma que o atributo gênero pode ter um dos seguintes tipos: Ficção, Ciências Naturais, Literatura ou Ciências Humanas. Assuma que tais tipos de gêneros são valores já conhecidos pela classe Livro, e que todos os objetos devem compartilhá-los.

Para você saber mais...

A seguir, você verá algumas sugestões de artigos da web sobre atributos e métodos estáticos:

- http://www.ic.unicamp.br/~cmrubira/aacesta/java/javatut12.html
- http://labs.sadjow.com/2008/08/10/palavra-chave-static/
- http://homepages.dcc.ufmg.br/~rodolfo/aeds-1-05/metodosEstaticos/metodosEstaticos.html

Classes Abstratas

Ao longo dos nossos estudos sobre a programação orientada a objetos em Java, vimos os conceitos e vários exemplos de implementações de classes e objetos. Você viu que as classes são como “moldes” que geram os objetos segundo a sua definição.

Todas as classes vistas até agora são o que chamamos de classes concretas, pois são utilizadas para gerar objetos diretamente. Ou, ainda, estão “prontas” para serem usadas para gerar seus objetos, como formas de bolo ou biscoitos prontas para receberem a massa e darem objetos quentinhos! Mas, existem classes das quais não é possível gerar (instanciar) objetos, são as chamadas classes abstratas.

Primeiro, vejamos o que é uma classe abstrata.

Saiba Mais!

Classe abstrata é aquela declarada como tal, através da palavra reservada *abstract*, e que pode definir pelo menos um método abstrato.

Vídeo 03 - Classes Concretas x Classes Abstratas

A Listagem 4 apresenta um exemplo de uma classe abstrata, chamada FiguraAbstrata. Uma figura abstrata representa uma figura qualquer, por isso não sabe como ser desenhada e, de fato, não pode ser instanciada, pois ela define um método abstrato (desenha) para se autodesenhar, que ainda está indefinido.

No nosso exemplo, a classe FiguraAbstrata indica que métodos as figuras derivadas dela (subclasses ou classes filha da mesma) devem realizar ou implementar. Esses métodos são os chamados métodos abstratos. Uma classe abstrata não pode instanciar objetos, porém, pode ser derivada por outras classes que, por sua vez, deverão implementar seus métodos ou declarar-se também abstratas.

```
1 public abstract class FiguraAbstrata{
2    /**
3    * Método abstrato de desenho de uma figura
4    */
5    abstract public void desenha();
6 }
```

Listagem 4 - Classe abstrata FiguraAbstrata

A Listagem 5 apresenta a classe derivada Quadrado (subclasse) da classe FiguraAbstrata.

A classe Quadrado realiza o que precisa para ser desenhada, implementando o método desenha(). É possível criar instâncias (objetos) da classe Quadrado, porque ela não é abstrata nem é declarada como tal, e também define uma implementação concreta para o método desenha().

```
1 public class Quadrado extendes FiguraAbstrata{
2    @Override
3    public void desenha()
4       System.out.println(" ---");
5       System.out.println("|   |");
6       System.out.println(" ---");
7    }
8 }
```

Listagem 5 - Classe Quadrado

A importância das classes abstratas dar-se principalmente em tempo de projeto do sistema, cria um nível a mais de abstração e torna o projeto mais reutilizável.

Para você saber mais...

A seguir, você verá algumas sugestões de artigos da web sobre classes abstratas:

- http://java.sun.com/docs/books/tutorial/java/IandI/abstract.html
- http://www.tiexpert.net/programacao/java/classes-abstratas.php

Atividade 02

1. Para exercitar o conceito de classes abstratas, crie outras classes derivadas (subclasses) de FiguraAbstrata como Trapezio, Triangulo etc. Em seguida, implemente o método desenha().
2. Finalmente, crie também uma classe com método main() para tentar instanciar objetos tanto da classe FiguraAbstrata quanto das subclasses que você criou. Não deixe de fazer chamada ao método desenha() para ver o que acontece.

Interface

Imagine uma grande equipe de desenvolvimento de software, onde há subdivisões de acordo com a parte que o software irá atuar, logo, é necessária uma espécie de contrato para garantir “independência” entre as equipes. Assim, saberão o que deverão fazer sem que saibam exatamente como está sendo implementado pela outra equipe.

Imaginemos que fabricamos softwares para serem implantados em brinquedos como robôs, carrinhos e aviões que possuem como característica principal a capacidade de se locomoverem segundo os comandos de um controle remoto. Cada brinquedo entende o que precisa fazer para mover-se para frente ou para trás, parar e emitir um sinal de localização.

Os brinquedos são fabricados por uma companhia e os controles por uma empresa parceira em produção de eletrônicos. Ambos são implementados na linguagem Java. O contrato estabelecido entre os desenvolvedores é de que todos os produtos que necessitam ser controlados por um controle remoto deverão implementar a interface Mobilidade, cujo código é apresentado na Listagem 6.

```
1 public interface Mobilidade{
2     public void andarFrente();
3     public void andarTras();
4     public void parar();
5     public void virarDireita(int graus);
6     public void virarEsquerda(int graus);
7 }
```

Listagem 6 - Interface mobilidade

De um lado, as classes que representam os produtos a serem controlados irão “inscrever-se” ou declarar-se *implementadoras* dessa interface. De outro lado, a classe controle remoto irá operar sobre esses métodos sem conhecer nada além do que ela precisa saber, ou seja, o conjunto de métodos da interface mobilidade. Por isso, chamados de interface, ou modo de ver os objetos.

A Listagem 7 apresenta a classe Robô que implementa a interface Mobilidade.

```
1  public class Robo implements Mobilidade{
2      public void andarFrente(){
3          System.out.println("[robô]: andando para frente...");
4      }
5      public void andarTras(){
6          System.out.println("[robô]: andando para trás...");
7      }
8      public void parar(){
9          System.out.println("[robô]: parado");
10     }
11     public void virarDireita(int graus){
12         System.out.println("[robô]: virando para direita " + graus + " graus");
13     }
14     public void virarEsquerda(int graus){
15         System.out.println("[robô]: virando para esquerda " + graus + " graus");
16     }
17     public void moveBracoDireito(){
18         System.out.println("[robô]: movendo braço direito");
19     }
20     public void moveBracoEsquerdo(){
21         System.out.println("[robô]: movendo braço esquerdo");
22     }
23 }
```

Listagem 7 - Classe Robô

Veja que a classe Robô se define implementadora da interface Mobilidade através da palavra reservada *implements*, seguida do nome da interface Mobilidade. Outros métodos podem ser definidos livremente pelas classes que implementam interfaces. O que vemos na definição dos métodos moveBracoDireito() e moveBracoEsquerdo(), por exemplo, são típicos de funções de um robô, mas que certamente não se aplicaria a um veículo de brinquedo.

A Listagem 8 apresenta a classe Trator, que também implementa a interface Mobilidade. Assim como a classe Robô, a classe Trator implementa a interface Mobilidade e declara outro método chamado de ergueEscavadeira(), que representa uma funcionalidade extra e que não será controlada remotamente.

```
public class Trator implements Mobilidade{
   public void andarFrente(){
      System.out.println("[trator]: andando para frente...");
   }
   public void andarTras(){
      System.out.println("[trator]: andando para trás...");
   }
   public void parar(){
      System.out.println("[trator]: parado");
   }
   public void virarDireita(int graus){
      System.out.println("[trator]: virando para direita " + graus + " graus");
   }
   public void virarEsquerda(int graus){
      System.out.println("[trator]: virando para esquerda " + graus + " graus");
   }
   public void erqueEscavadeira(){
      System.out.println("[trator]: ergue escavadeira");
   }
}
```

Listagem 8 - Classe Trator

Assim, como o método moveBracoDireito() da classe Robô, o método ergueEscavadeira() da classe Trator faz uma atividade específica para o brinquedo. E, por isso, só é definido no próprio brinquedo. Por fim, temos a classe ControleRemoto que realiza operações sobre objetos “Móveis” que implementam a interface Mobilidade. Veja a Listagem 9.

```
public class ControleRemoto{
   public void moverObjetoParaFrente(Mobilidade obj){
      obj.andarFrente();
   }
   public void moverObjetoParaTras(Mobilidade obj){
      obj.andarTras();
   }
   public void dobrarADireita(Mobilidade obj){
      obj.virarDireita(graus);
   }
   public void dobrarAEsquerda(Mobilidade obj){
      obj.virarEsquerda(graus);
   }
}
```

Listagem 9 - Classe ControleRemoto

Observe que a classe ControleRemoto possui métodos que operam sobre objetos de classes que implementam a interface Mobilidade, sejam eles quais forem. Por isso, na declaração dos métodos, não está explícito classes como Trator ou

Robô, pois o controle trata-os de forma transparente. Esse exemplo também representa uma das formas de polimorfismo.

Vídeo 04 - Interfaces

Atividade 03

1. Para praticar os conceitos aprendidos, crie uma classe com um método main(), que teste a classe controle remoto, através da criação de objetos da classe Trator e Robô e veja como são tratados de forma idêntica pela classe ControleRemoto.

Leitura Complementar

Veja a seguir endereços de artigos da web que tratam de interfaces:

- http://labs.sadjow.com/2008/08/23/java-o-que-e-interface/
- http://java.sun.com/docs/books/tutorial/java/IandI/createinterface.html

Resumo

Nesta aula, você viu técnicas sofisticadas para projetar sistemas através de classes abstratas que servem para definir comportamentos os quais deverão ser "materializados" em classes filhas, dando poder e abstração ao projeto de sistemas. Você estudou que classes podem definir atributos e métodos próprios e independentes de seus objetos, chamados de atributos ou métodos de classes, os quais podem ser acessados diretamente através do nome da classe e seu respectivo nome. Você viu também como definir contratos entre classes, através de um tipo especial da linguagem Java, conhecido como interface. Cada interface apenas declara assinaturas de métodos, não oferecendo implementações. As classes que implementam a interface são obrigadas a oferecer uma implementação para cada um dos métodos declarados na interface.

Autoavaliação

1. Defina o que é um atributo estático.
2. Defina o que é um método estático.
3. Exercite a imaginação e crie para cada classe a seguir um atributo estático.
    a. Pessoa
    b. Carro
    c. Livro

    d. ConsultaMédica

    e. HistóricoEscolar

4. Você é capaz de comparar os conceitos de interface e classe abstrata, destacando semelhanças de um lado e diferenças do outro?

5. Defina uma outra interface, cujas classes Trator e Robô irão implementar, que represente a sua manutenção energética, pois ambos precisam recarregar baterias, ligar, desligar etc.

Referências

DEITEL, H. M.; DEITEL, P. J. **Java como programar**. Porto Alegre: Bookman, 2003.

SANTOS, Rafael. **Introdução à programação orientada a objetos usando Java.** Rio de Janeiro: Editora Campus, 2003THE

JAVA tutorials. Disponível em: <http://java.sun.com/docs/books/tutorial/index.html>. Acesso em: 17 maio 2010.