Interfaces

Fabio Takeshi Matsunaga

Universidade Estadual do Paraná

15 de setembro de 2015

Missão de hoje

- Revisão de algumas atividades sobre relacionamento de classes;
- Introdução a interface;
- Diferença com relação à herança;

Atividade

- Acrescente dois atributos na classe Funcionario: uma para representar a hora de entrada e outra a hora de saída. A partir desses atributos, faça um método para calcular a quantidade de horas trabalhadas por um funcionário, de acordo com a hora de entrada e a hora de saída. Utilize como atributo um objeto da classe Hora, implementada na última aula.

Voltando à classe Funcionario

- Vamos supor que na classe Funcionario, herdamos alguns Funcionários específicos, como o Diretor e o Gerente;
- Através dessas subclasses, desejamos fazer um método de autenticação de usuários em um sistema;
- Esse método de autenticação é restrito a alguns funcionários específicos;

Voltando à classe Funcionario

```
1  public class Diretor extends Funcionario {
2
3    public boolean autentica(int senha) {
4      // verifica aqui se a senha confere com a
           recebida como parametro
5
6    }
7  }
```

```
1  public class Gerente extends Funcionario {
2
3    public boolean autentica(int senha) {
4      // verifica aqui se a senha confere com a
           recebida como parametro
5      // no caso do gerente verifica tamb m se o
           departamento dele tem acesso
6    }
7  }
```

Problema . . .

Desejamos fazer um método de entrada no sistema para cada funcionário. Mas não é todo funcionário que possui.

```
1  public class SistemaInterno {
2    void login(Funcionario funcionario) {
3       // Invocar o m todo autentica
4       funcionario.autentica();
5       // Nao vai compilar!
6    }
7  }
8
9  ...
10
11 // no main:
12 SistemaInterno s = new SistemaInterno();
13 s.login(new Gerente()); // OK
14 s.login(new Funcionario()); // ???
```

Solução 1

Criar um método de login específico para cada tipo de funcionário que tiver acesso ao sistema.

```
1  public class SistemaInterno {
2
3    void loginGerente(Gerente g) {
4       g.autentica();
5    }
6
7    void loginGerente(Diretor d) {
8       d.autentica();
9    }
10
11 }
```

Porém, precisamos criar um método de login novo para cada tipo de funcionário novo?

Solução 2

Criar uma classe chamada FuncionarioAutenticavel, que herda do Funcionario, para que o Gerente e o Diretor possam obter o método de autenticação da classe herdeira. Neste caso, o FuncionarioAutenticavel irá possuir somente o método de autenticação.

```
1 abstract class FuncionarioAutenticavel extends
      Funcionario {
2     // Herda atributos de funcion rio , nao
          precisa reimplementar
3
4     // Novo metodo
5     public void autentica() {
6       // ...
7     }
8 }
```

Então, neste caso, a classe Gerente e Diretor extenderiam desta classe. Simples assim, né?

Solução 2

```
1  public class Gerente extends
       FuncionarioAutenticavel {
2      // Herda atributos de funcion rio , nao
           precisa reimplementar
3
4      // Novo metodo
5      public void autentica() {
6        // ...
7      }
8  }
```

Solução 2

```
1  public class SistemaInterno {
2    void login(FuncionarioAutenticavel fa) {
3      int senha = //pega senha de um lugar, ou de
           um leitor biometrico
4      // aqui eu posso chamar o autentica!
5      // Pois todo FuncionarioAutenticavel tem
6      boolean autentic = fa.autentica(senha);
7    }
8  }
```

Isso resolve o problema se considerarmos que somente funcionários tenham acesso ao sistema interno da empresa. Porém, se considerarmos que outros objetos possam ter acesso ao mesmo? Por exemplo, um Cliente?

Solução 3

```
1  public class Cliente extends
       FuncionarioAutenticavel {
2    void login(FuncionarioAutenticavel fa) {
3      int senha = //pega senha de um lugar, ou de
           um leitor biometrico
4      // aqui eu posso chamar o autentica!
5      // Pois todo FuncionarioAutenticavel tem
6      boolean autentic = fa.autentica(senha);
7    }
8  }
```

Isso é uma herança sem sentido! Um Cliente não é um FuncionárioAutenticável. Apenas queremos que ele tenha acesso ao sistema.

Problema geral

- Diferenciar as classes Diretor, Gerente e Cliente, de modo que eles tenham acesso ao sistema interno;
- Solução: como referenciar essas três classes da mesma maneira, pois Cliente é diferente de um Diretor e Gerente;
- Solução do problema: criar interfaces em Java;

Classes abstratas X Interfaces

Interfaces

- Uma Interface não é uma classe propriamente dita, e sim uma entidade;
- Os métodos não possuem implementaçao, são todos **abstratos** e possuem somente uma **assinatura**;
- Não se instancia uma interface e muito menos implementa construtores;
- Não possuem atributos;
- Funciona como um tipo de contrato, em que uma interface fornece métodos que toda classe que a implementa deve definir;
- Funciona como uma possível solução para heranças múltiplas;

Classes abstratas X Interfaces

Classes abstratas

- Utilizadas para herança de classes (podem somente ser herdados);
- Deve conter métodos abstratos, que devem ser reimplementados pelas subclasses;
- Podem conter atributos e construtores;
- Não podem ser instanciados diretamente, somente pode ocorrer a instância das classes herdeiras da classe abstrata;
- O contrato das classes abstratas é que as classes herdeiras devem complementar o que existe na classe abstrata;
  - Na classe abstrata pode conter métodos implementados, porém deve também conter métodos abstratos;
  - Na interface, todos os métodos são abstratos, não se implementa nenhum método;

Classes abstratas X Interfaces

- Em Java, não é possível realizar herança múltipla;
- É possível fazer com que uma classe herde um método abstrato e implemente uma interface simultaneamente;
- Dessa forma, uma classe pode ser constituído de duas classes: uma abstrata e outra interface.

Classes abstratas X Interfaces

Característica	**Classes abstratas**	**Interfaces**
Herança	Simples	Múltipla
Métodos	Métodos completos e abstratos	Somente abstratos
Velocidade	Rápido	Lento
Funcionalidade	Uma alteração na classe fornece uma implementação padrão	Uma alteração na interface requer a alteração em todas as implementações

Atividade em sala

- Implementar uma classe (pode ser abstrata) para representar uma TV. Considere que uma TV possui alguns atributos de características como o tamanho em polegadas e atributos de estado como canal, volume e se está ligada ou não;
- Em seguida, implemente uma interface para representar um controle remoto. Essa classe (interface) conterá alguns comandos (métodos) para acionar as ações da TV, como mudar canal, aumentar/diminuir o volume, ligar/desligar, etc;
- Teste uma TV que implemente a interface do Controle Remoto. Imaginamos uma situação para representar uma TV que possui controle remoto;