Aula 19: Classes abstratas

Professor(a): Virgínia Fernandes Mota
http://www.dcc.ufmg.br/∼virginiaferm

ALGORITMOS E ESTRUTURAS DE DADOS - SETOR DE INFORMÁTICA

Classes abstratas

- Ao término desta aula, você será capaz de:
  - utilizar classes abstratas, quando necessário.

Repetindo mais código?

- Vamos recordar como pode estar nossa classe Funcionario:

```
1  class Funcionario {
2    protected String nome;
3    protected String cpf;
4    protected double salario;
5
6    public double getBonificacao() {
7      return this.salario * 1.2;
8    }
9    // outros métodos aqui
10 }
```

Repetindo mais código?

- Considere o nosso ControleDeBonificacao:

```
1  class ControleDeBonificacoes {
2
3    private double totalDeBonificacoes = 0;
4
5      public void registra(Funcionario f) {
6        System.out.println("Adicionando bonificação do
             funcionario: " + f);
7        this.totalDeBonificacoes += f.getBonificacao();
8      }
9
10     public double getTotalDeBonificacoes() {
11       return this.totalDeBonificacoes;
12     }
13 }
```

Repetindo mais código?

- O método registra recebe qualquer referência do tipo Funcionario: objetos do tipo Funcionario e qualquer de seus subtipos (Gerente, Diretor, ...).
- Estamos utilizando aqui a classe Funcionario para o **polimorfismo**.
- Porém, em alguns sistemas usamos uma classe com apenas esses intuitos: de economizar um pouco código e ganhar polimorfismo para criar métodos mais genéricos, que se encaixem a diversos objetos.

Repetindo mais código?

- Faz sentido ter uma referência do tipo Funcionario? Essa pergunta é diferente de saber se faz sentido ter um objeto do tipo Funcionario.
- ... nesse caso, faz sim e é muito útil.

Repetindo mais código?

- Porém, dar new em Funcionario pode não fazer sentido, isto é, não queremos receber um objeto do tipo Funcionario, mas sim que aquela referência seja ou um Gerente, ou um Diretor, etc.
- Exemplo: imagine a classe Pessoa e duas filhas, PessoaFisica e PessoaJuridica $\rightarrow$ faz sentido permitir instanciar Pessoa? Não!
- Solução: **classes abstratas**.

Classes abstratas

- O que é a nossa classe Funcionário?
- Nossa empresa tem apenas Diretores, Gerentes, Secretárias, etc. Ela é uma classe que apenas idealiza um tipo, define apenas um rascunho.
- Não faz sentido ter um objeto Funcionário apenas!

Classes abstratas

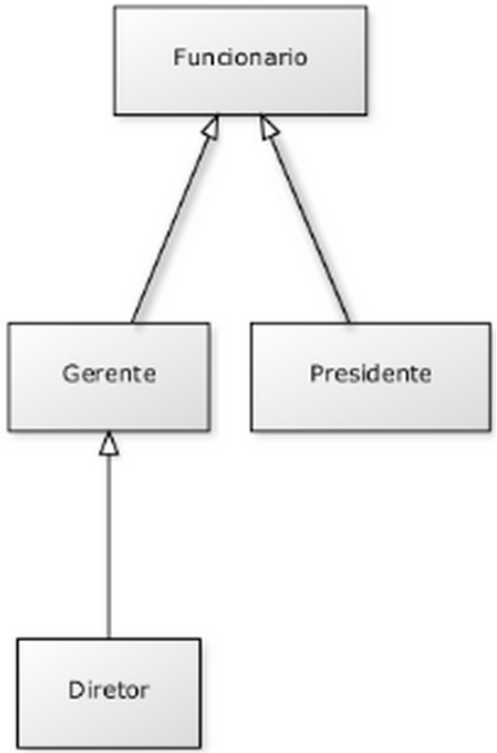
- Usamos a palavra chave **abstract** para impedir que ela possa ser instanciada.

```
1  abstract class Funcionario {
2
3    protected double salario;
4
5    public double getBonificacao() {
6      return this.salario * 1.2;
7    }
8
9    // outros atributos e métodos comuns a todos
         Funcionarios
10
11 }
```

- A classe Funcionário então serve para o polimorfismo e herança dos atributos e métodos!

Classes abstratas

```
1  class Gerente extends Funcionario {
2
3    public double getBonificacao() {
4      return this.salario * 1.4 + 1000;
5    }
6  }
```



Métodos abstratos

- Se o método getBonificacao não fosse reescrito, ele seria herdado da classe mãe, fazendo com que devolvesse o salário mais 20%.
- Faz algum sentido ter esse método na classe Funcionario? Não existe uma regra geral!
- Em uma classe abstrata, podemos escrever que determinado método será sempre escrito pelas classes filhas. Isto é, um **método abstrato**.
  - $\rightarrow$ Todas as classes filhas (concretas) devem reescrever esse método

Métodos abstratos

- Não colocamos o corpo do método e usamos a palavra chave abstract para definir o mesmo!
- As filhas tornarão o método concreto.

```
1  abstract class Funcionario {
2
3    abstract double getBonificacao();
4
5    // outros atributos e métodos
6
7  }
```

Métodos abstratos

- O método do ControleDeBonificacao estava assim:

```
1  public void registra(Funcionario f) {
2    System.out.println("Adicionando bonificação do
         funcionario: " + f);
3    this.totalDeBonificacoes += f.getBonificacao();
4  }
```

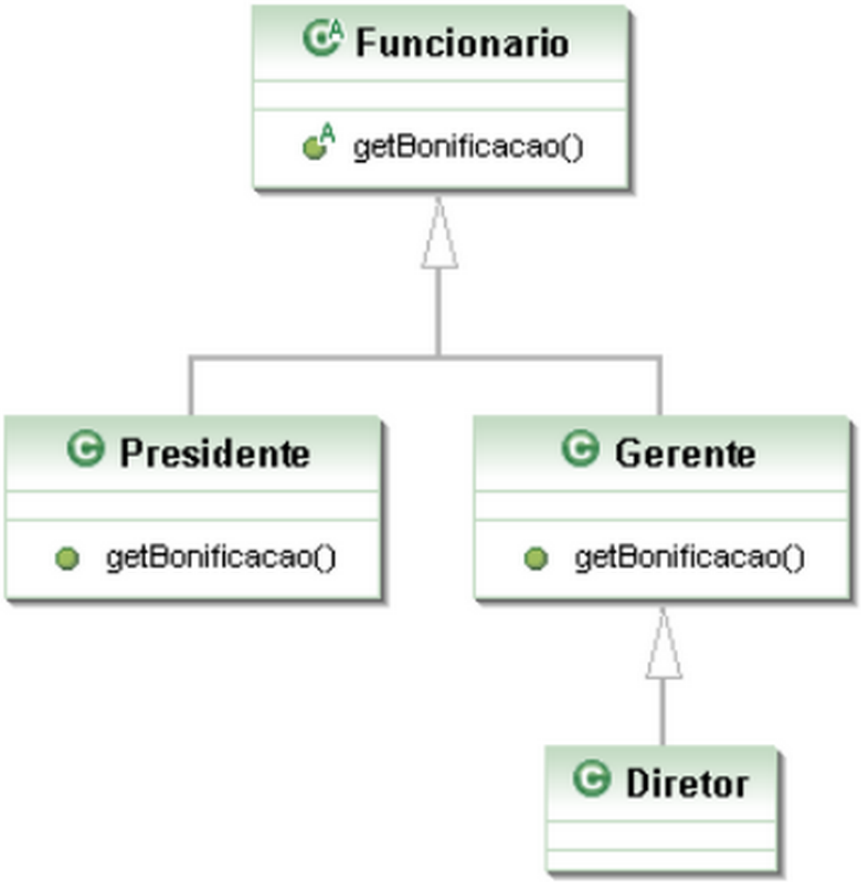
- Como posso acessar o método getBonificacao se ele não existe na classe Funcionario?

Métodos abstratos

- Já que o método é abstrato, com certeza suas subclasses têm esse método, o que garante que essa invocação de método não vai falhar.
- Um método abstrato obriga a classe em que ele se encontra ser abstrata.

Exemplo

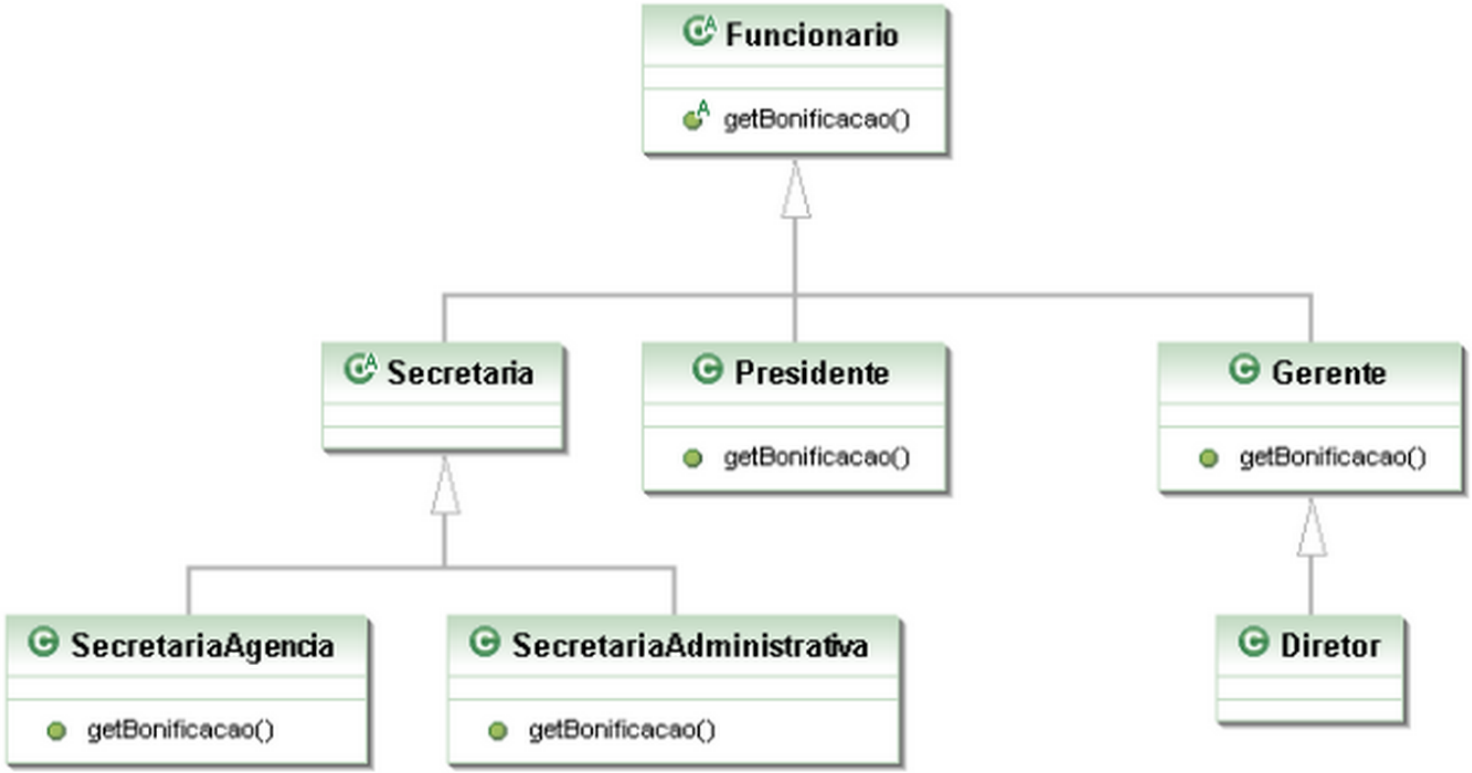
- E se nossa empresa fosse assim?



- Essas classes vão compilar? Vão rodar?

Exemplo

- E se nossa empresa fosse assim?



- Essas classes vão compilar? Vão rodar?

Observações

- Uma classe que estende uma classe normal também pode ser abstrata! Ela não poderá ser instanciada, mas sua classe pai sim!
- Uma classe abstrata não precisa necessariamente ter um método abstrato.

Exercícios

- 1. Repare que a nossa classe Conta é uma excelente candidata para uma classe abstrata. Transforme a classe Conta em abstrata, repare o que acontece no seu main já existente do TestaContas.
- 2. Para que o código do main volte a compilar, troque o new Conta() por new ContaCorrente().
- 3. Transforme o método atualiza() da classe Conta em abstrato. PROBLEMAS?
- 4. Reescreva o método atualiza()!

Exercícios

Próxima aula...

Mais Exercícios

- 5. Crie uma classe chamada Pessoa. Uma pessoa possui um nome e uma idade.
  - crie 2 construtores: 1 que recebe o nome e a idade como parâmetros de entrada e um que não recebe parâmetros e inicializa os atributos com um valor padrão ("indefinido" para Strings e 0 para inteiros).
  - crie os métodos de acesso para os atributos (GET e SET).

Mais Exercícios

- 6. Crie uma classe Amigo, que herda Pessoa, e possui uma data de aniversário.
  - crie um construtor que não recebe parâmetros de entrada, e inicializa o atributo com um valor padrão ("indefinido", por exemplo).
  - crie os métodos de acesso para o atributo data de nascimento.

Mais Exercícios

- 7. Crie uma classe Conhecido, que herda Pessoa, e possui um email.
  - crie um construtor que não recebe parâmetros de entrada, e inicializa o e-mail com um valor padrão ("indefinido", por exemplo).
  - crie os métodos de acesso para este atributo.

Mais Exercícios

- 8. Crie agora, uma classe Agenda, que possui pessoas (em um array) e dois atributos que controlam: a quantidade de amigos e a quantidade de conhecidos.
  - crie um construtor que recebe por parâmetro a quantidade de pessoas que a agenda terá, e inicializa o array de Pessoa. Neste construtor, inicialize todas as posições do array criando ALEATORIAMENTE um Conhecido ou um Amigo
    utilize o comando *1 + (int) (Math.random() * 2)* para sortear valores entre 1 e 2. Se o valor encontrado for 1, crie um Amigo. Se o valor encontrado for 2, crie um Conhecido.
  - Crie os métodos GET para todos os atributos da classe Agenda.

Mais Exercícios

- 8....
  - crie um método chamado addInformacoes, que não recebe parâmetros de entrada. Para cada Pessoa na agenda, peça para o usuário digitar (via teclado) as informações cabíveis para cada tipo de Pessoa, e acesse os métodos SET para atribuir as informações.
  - crie um método chamado imprimeAniversários, que imprime os aniversários de todos os amigos que estão armazenados na agenda.
  - crie um método chamado imprimeEmail, que imprime os e-mails de todos os conhecidos que estão armazenados na agenda.

Mais Exercícios

- 9. Crie uma classe de teste para a Agenda.
  - peça para o usuário informar (via teclado) quantas pessoas ele deseja colocar na agenda, e crie uma Agenda com esta informação.
  - imprima na tela a quantidade de amigos e de conhecidos na agenda.
  - adicione informações à agenda.
  - imprima todos os aniversários dos amigos presentes na agenda.
  - imprima todos os e-mails dos conhecidos armazenados na agenda.

Dica: Comparação de Objetos

```
1  objeto.equals(obj); // Compara referências
2
3  objeto.getClass(); //retorna a classe do objeto
4  if (obj1.getClass() == obj2.getClass()) {...} //compara se
       as classes são as mesmas
5
6  if (objeto instanceof Classe) {...} //compara se objeto é da
       classe Classe
```

Na próxima aula...

Interfaces