CLASSES E MÉTODOS ABSTRATOS

Vamos lembrar da nossa classe Funcionario:

```
class Funcionario {

  protected String nome;
  protected String cpf;
  protected double salario;

  public double getBonificacao() {
    return this.salario * 1.2;
  }
  // outros métodos aqui
}
```

E fizemos uma classe ControleDeBonificacoes:

```
class ControleDeBonificacoes {
  private double totalDeBonificacoes = 0;

    public void registra(Funcionario f) {
      System.out.println("Adicionando bonificação do funcionario: " + f);
      this.totalDeBonificacoes += f.getBonificacao();
    }
    public double getTotalDeBonificacoes() {
      return this.totalDeBonificacoes;
    }
}
```

Nosso método registra recebe qualquer referência do tipo Funcionario, isto é, podem ser objetos do tipo Funcionario e qualquer de seus subtipos: Gerente, Diretor e, eventualmente, alguma nova subclasse que venha ser escrita, sem prévio conhecimento do autor da ControleDeBonificacoes.

Estamos utilizando aqui a classe Funcionario para o polimorfismo. Se não fosse ela, teríamos um grande prejuízo: precisaríamos criar um método registra para receber cada um dos tipos de Funcionario, um para Gerente, um para Diretor, etc. Repare que perder esse poder é muito pior do que a pequena vantagem que a herança traz em herdar código.

Porém, em alguns sistemas, como é o nosso caso, 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.

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.

Referenciando Funcionario temos o polimorfismo de referência, já que podemos receber qualquer objeto que seja um Funcionario. 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. Algo mais concreto que um Funcionario.

CLASSE ABSTRATA

O que, exatamente, vem a ser a nossa classe Funcionario? Nossa empresa tem apenas Diretores, Gerentes, Secretárias, etc. Ela é uma classe que apenas idealiza um tipo, define apenas um rascunho.

Para o nosso sistema, é inadmissível que um objeto seja apenas do tipo Funcionario (pode existir um sistema em que faça sentido ter objetos do tipo Funcionario ou apenas Pessoa, mas, no nosso caso, não).

Usamos a palavra chave abstract para impedir que ela possa ser instanciada. Esse é o efeito direto de se usar o modificador abstract na declaração de uma classe:

```
abstract class Funcionario {
  protected double salario;


  public double getBonificacao() {
    return this.salario * 1.2;
  }
  // outros atributos e métodos comuns a todos Funcionarios
}
```

E, no meio de um código:

```
Funcionario f = new Funcionario(); // não compila!!! Dá erro!!
```

O código acima não compila. O problema é instanciar a classe - criar referência, você pode. Se ela não pode ser instanciada, para que serve? Serve para o polimorfismo e herança dos atributos e métodos, que são recursos muito poderosos, como já vimos.

Vamos então herdar dessa classe, reescrevendo o método getBonificacao:

```
class Gerente extends Funcionario {
  public double getBonificacao() {
    return this.salario * 1.4 + 1000;
  }
}
```

Mas qual é a real vantagem de uma classe abstrata? Poderíamos ter feito isto com uma herança comum. Por enquanto, a única diferença é que não podemos instanciar um objeto do tipo Funcionario, que já é de grande valia, dando mais consistência ao sistema.

Fique claro que a nossa decisão de transformar Funcionario em uma classe abstrata dependeu do nosso domínio. Pode ser que, em um sistema com classes similares, faça sentido que uma classe análoga a Funcionario seja concreta.

MÉTODO ABSTRATO

Levando em consideração que cada funcionário em nosso sistema tem uma regra totalmente diferente para ser bonificado, faz algum sentido ter esse método na classe Funcionario? Será que existe uma bonificação padrão para todo tipo de Funcionario? Parece que não, cada classe filha terá um método diferente de bonificação pois, de acordo com nosso sistema, não existe uma regra geral: queremos que cada pessoa que escreve a classe de um Funcionario diferente (subclasses de Funcionario) reescreva o método getBonificacao de acordo com as suas regras.

Existe um recurso em Java que, em uma classe abstrata, podemos escrever que determinado método será sempre escrito pelas classes filhas. Isto é, um método abstrato.

Ele indica que todas as classes filhas (concretas, isto é, que não forem abstratas) devem reescrever esse método ou não compilarão. É como se você herdasse a responsabilidade de ter aquele método.

```
abstract class Funcionario {
  abstract double getBonificacao();
  // outros atributos e métodos

}
```

Repare que não colocamos o corpo do método e usamos a palavra chave abstract para definir o mesmo. Por que não colocar corpo algum? Porque esse método nunca vai ser chamado, sempre que alguém chamar o método getBonificacao, vai cair em uma das suas filhas, que realmente escreveram o método.

Qualquer classe que estender a classe Funcionario será obrigada a reescrever este método, tornando-o "concreto". Se não reescreverem esse método, um erro de compilação ocorrerá.

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.