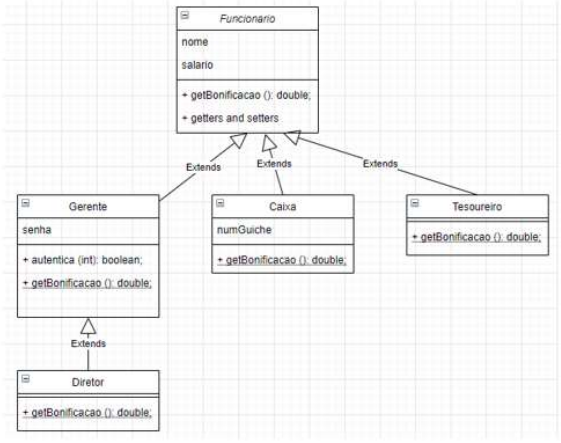
Aula 07 – parte 2
Exercício projeto bancov09. Exercício para ser feito em classe.

1) Abra o eclipse e crie um projeto chamado bancov09 a partir do projeto bancov08. Mantenha todas as classes.
2) Comente o método getBonificacao de Caixa para que ele passe a herdar o comportamento de Funcionario. Rode a classe TestaGerenciadorDeBonificacoes e vai perceber que agora a bonificação de um Caixa é de 10% sobre seu salário.
3) Volte com o método getBonificacao que retorna 15% sobre o salário de um Caixa.
4)
5) Dê uma boa olhada em nossa hierarquia de classes e perceba que Gerente, Diretor, Caixa e Tesoureiro são Funcionario e que todos possuem uma implementação diferente para o método getBonificação.



6) Pensando no mundo real, existe alguém que seja simplesmente Funcionário de um Banco? No mundo real, quem trabalha em um banco é Gerente, Caixa, Tesoureiro, Diretor, Atendente etc. Ninguém é simplesmente Funcionario. Isso não existe. Logo, não deveria ser possível em nosso Sistema dar new Funcionario(). Qual seria a solução para isso? Remover a classe Funcionario e descer todos os atributos e métodos para as subclasses? Assim, além de espalhar o código, perderíamos um recurso poderoso que é o polimorfismo. A possibilidade de referenciar todos como Funcionario. Isso impactaria diretamente em nossa classe ContabilizadorDeBonificacoes. Veja.

```
3  public class ContabilizadorDeBonificacoes {
4      private double totalEmBonificacoes=0;
5      //Agora posso receber todos os subtipos de Funcionario
6      public void contabilizaBonificacao(Funcionario f) {
7          //O getBonificacao invocado vai depender do objeto que foi passado como argumento
8          //Pode ser Gerente, Caixa, Tesoureiro ....
9          this.totalEmBonificacoes += f.getBonificacao();
10     }
11
12     public double getTotalEmBonificacoes() {
13         return totalEmBonificacoes;
14     }
15 }
```

7) Funcionário é apenas uma abstração, uma ideia do que todo Funcionario deve ter. Logo, é séria candidata a classe abstrata. Classes abstratas não podem ser instanciadas. Elas servem para concentrar regras e principalmente fornecer polimorfismo. Não podemos perder esse poder. Transforme a classe Funcionario em abstrata. É bem simples.

```
2  public abstract class Funcionario {
```

8) Agora, se você tentar instanciar um Funcionario, o eclipse vai acusar um erro. Veja. Problema resolvido!

```
10 public class TestaGerenciadorDeBonificacoes {
11     public static void main(String[] args) {
12         Funcionario F = new Funcionario();
13         Funcionario f = new Diretor();
14         Gerente g1 = new Gerente();
```

Observe as linhas 12 e 13. Não podemos instanciar um Funcionario, mas ainda podemos utilizar esta classe para referenciar seus subtipos. O melhor de tudo é que podemos utilizar Funcionario em nossos métodos para definir que determinado argumento pode ser qualquer subtipo de Funcionario, como já fazemos em contabilizaBonificacao.

Ao contrário de outras linguagens, no Java uma classe abstrata pode ter atributos, construtores (para fazer a compatibilidade e concentrar regras) e métodos implementados para serem herdados.

9) Inevitavelmente uma questão nos faz parar para pensar: Se nunca teremos uma instancia de Funcionario e o getBonificacao das subclasses é diferente do getBonificacao de Funcionario, não faz sentido ter esse método retornando 10% sobre o salário de um Funcionario. Afinal, nunca teremos uma instancia de Funcionario em memória. Devemos retirar o método getBonificacao de Funcionario?
10) Vamos dar uma boa olhada no método contabilizaBonificacao de GerenciadorDeBonificacoes e fazer algumas observações.

```
3  public class ContabilizadorDeBonificacoes {
4      private double totalEmBonificacoes=0;
5      //Agora posso receber todos os subtipos de Funcionario
6      public void contabilizaBonificacao(Funcionario f) {
7          //O getBonificacao invocado vai depender do objeto que foi passado como argumento
8          //Pode ser Gerente, Caixa, Tesoureiro ....
9          this.totalEmBonificacoes += f.getBonificacao();
10     }
11
12     public double getTotalEmBonificacoes() {
13         return totalEmBonificacoes;
14     }
15 }
```

Observações sobre o método contabilizaBonificacao:
Podemos receber todo subtipo de Funcionario como argumento.
Todo subtipo de Funcionario precisa ter um método getBonificacao. Vou além, como a referência que estamos utilizando é a de um Funcionario, precisamos fornecer a garantia de que todo Funcionario possui um getBonificacao.

11) Para garantir que todo Funcionario tem um getBonificacao, retire o comportamento desse método em Funcionário e declare o mesmo como abstrato. Quando dizemos que um método é abstrato, garantimos que nossas subclasses fornecerão uma implementação para esse método. Veja.

```
8      //Métodos abstratos possuem apenas assinatura
9      //Isso é uma garantia de que as subclasses concretas implementarão esse método
10     public abstract double getBonificacao();
11
```

O método abstrato acaba sendo uma garantia de que todas as filhas concretas implementarão esse método.

12) E se a subclasse também for abstrata? Nesse caso ela pode delegar essa obrigação para suas filhas que sejam classes concretas.
13) Faria sentido ter um método getBonificacao implementado em Funcionario? Sim. Desde que aquele comportamento seja herdado por uma ou mais subclasses. Por exemplo, se Caixa e Tesoureiro tivessem uma mesma bonificação, bastaria colocar essa regra em Funcionario e deixar essas duas subclasses herdarem essa regra. Assim a regra estaria concentrada em um único lugar e faria todo sentido! Não é o nosso caso por enquanto.
14) Deixe o método getBonificacao abstrato em Funcionario e perceba que tudo continua funcionando normalmente!