PROGRAMAÇÃO EM JAVA

MÓDULO 4

CONCEITOS IMPORTANTES

Acoplamento e composição vs herança

Agora que você entendeu o que é e como funciona a herança, precisamos tratar de dois conceitos importantes:

1. **Acoplamento:** Você deve ter percebido que a classe Gerente depende da classe Funcionário para ser executado corretamente. Isso também vai acontecer com a classe Secretário, caso fosse implementada, determinando o grau de acoplamento entre as classes.

   O acoplamento entre classes indica o grau de dependência entre os atributos e métodos dessas classes. Isto quer dizer que se duas ou mais classes possuem um grau de acoplamento alto, qualquer modificação na classe mãe pode atrapalhar o funcionamento das classes filhas.

2. **Composição vs Herança:** São duas abordagens da programação orientada a objetos para a reutilização de código (atributos e métodos comuns entre classes). Para entender a diferença entre composição e herança precisamos apresentar um exemplo:

   a. O conceito de **Herança** funciona como um relacionamento de categorização, isto é, um gerente é um funcionário. Em outras palavras, um gerente pertence a categoria de funcionário, por isso ele (o gerente) herda as características e comportamentos do funcionário;

   b. O conceito de **Composição** faz relação às partes de alguma coisa ou objeto, por exemplo:

      i. Uma pessoa possui pernas, braços, cabeça, etc...

      ii. Uma árvore possui tronco, raiz, folhas, galhos, etc...

Polimorfismo

Outro tópico importante quando se trata de programação orientada a objetos e que está ligado ao conceito de Herança é o chamado Polimorfismo. Há dois tipos de polimorfismo:

- **Polimorfismo dinâmico**: Acontece quando duas classes herdam as características e métodos de outra classe, porém, cria um método igual ao que existe na classe mãe. Também conhecido como SobreEscrita ou Override.
- **Polimorfismo estático**: Acontece quando uma mesma classe possui dois métodos com o mesmo nome, mas com número de parâmetros diferentes. Também conhecido como Sobrecarga de métodos ou OverLoad;

Sobrescrever um método – Override

Imagine que o seu chefe solicitou a você e sua equipe de programadores que criasse mais um método na classe Funcionário. O problema surgiu por que a diretoria da empresa resolveu dividir seus lucros dando uma bonificação de final de ano para todos os funcionários. Porém, um funcionário que não fosse Gerente receberia 10% de bonificação e os Gerentes deveriam receber 15%. Outro aspecto importante é que todas as bonificações têm como base o salário do funcionário, então, para o caso dos gerentes a bonificação ficaria 15% do salário e para os demais funcionários: 10% do salário. Como você resolveria este problema?

Para resolver este problema, você precisa seguir tais etapas:

- Primeiramente, abra a classe "Funcionario" e vamos criar o método BonificacaoAnual(). Ele deve ficar como mostrado abaixo:

```
public double BonificacaoAnual()
{
        double Salario_Com_Bonificacao = this.Salario_Func  + this.Salario_Func * 0.1;
        return Salario_Com_Bonificacao;
}
```

IMPORTANTE

Note que o método não altera o valor do salário, por que a bonificação não é um aumento de salário, mas sim deve acontecer apenas uma vez por ano. Por isso, o valor do salário não foi alterado e o método BonificacaoAnual() deve retornar o valor do Salário acrescido da Bonificação.

- Agora que o método BonificacaoAnual() foi criado, vamos utilizá-lo para funcionário com cargos de Gerente e para os demais. Para isso, altere a classe Missao6_principal para que fique da seguinte forma:

```
1  package Missao6;
2
3  public class Missao6_principal {
4
5      public static void main(String[] args)
6      {
7          Funcionario F1 = new Funcionario();
8          F1.setNome_Func("Luiz");
9          F1.setCPF_Func("555");
10         F1.setSalario_Func(500.00);
11
12         Gerente G = new Gerente();
13         G.setNome_Func("Alberto");
14         G.setCPF_Func("445588");
15         G.setSalario_Func(1000.00);
16         G.setSenha_Ger(888);
17         G.setArea_Ger("CONTAB");
18
19         double Bonif_F = F1.BonificacaoAnual();
20         double Bonif_G = G.BonificacaoAnual();
21
22         //apresentando os dados no console
23         System.out.println("DADOS DO GERENTE:");
24         System.out.println("Nome = " + G.getNome_Func());
25         System.out.println("Salario = " + G.getSalario_Func());
26         System.out.println("Salário com Bonificação = " + Bonif_G);
27
28         //apresentando os dados no console
29         System.out.println("DADOS DO FUNCIONARIO NAO GERENTE:");
30         System.out.println("Nome = " + F1.getNome_Func());
31         System.out.println("Salario = " + F1.getSalario_Func());
32         System.out.println("Salário com Bonificação = " + Bonif_F);
33     }
34 }
```

Override – parte 1

- Se você salvar e executar o programa, o resultado no console será o seguinte:

```
DADOS DO GERENTE:
Nome = Alberto
Salario = 1000.0
Salário com Bonificação = 1100.0
DADOS DO FUNCIONARIO NAO GERENTE:
Nome = Luiz
Salario = 500.0
Salário com Bonificação = 550.0
```

- Note que o valor de ambos os salários foram acrescidos de 10%. Mas isso não é o que foi solicitado, pois o salário do funcionário com cargo de gerente deveria ter uma bonificação de 15% do salário.
- Para corrigir isso, uma das opções seria criar um novo método na classe gerente, chamado, por exemplo **BonificacaoGerente**. O problema é que agora teríamos dois métodos para bonificação: um na classe funcionário e outro na classe gerente. Imagine a confusão se cada cargo tivesse uma bonificação diferente. Teríamos os métodos **BonificacaoGerente**, **BonificaçãoSecretaria**, **BonificacaoEngenheiro** e assim por diante.
- Para resolver este problema, usamos a Sobreescrita de Método ou Override (em inglês).

Sobrecarga (Overload)

A sobrecarga de método acontece quando dois métodos são criados na mesma classe com o mesmo nome, porém com assinaturas diferentes. Neste sentido, podemos criar um novo método BonificacaoAnual() dentro da classe Funcionario da seguinte forma:

```
package Missao6;
public class Funcionario
{
...
        public double BonificacaoAnual()
        {
            double Salario_Com_Bonificacao = this.Salario_Func + this.Salario_Func * 0.1;
            return Salario_Com_Bonificacao;
        }
        public double BonificacaoAnual(double salario )
        {
            double Salario_Com_Bonificacao;
            if (salario < 1000)
            Salario_Com_Bonificacao = this.Salario_Func + this.Salario_Func* 0.1 + 100;
            else
            Salario_Com_Bonificacao = this.Salario_Func + this.Salario_Func* 0.1;
            return Salario_Com_Bonificacao;
        }
...
}
```

Note que os métodos têm o mesmo nome, mas assinaturas diferentes, pois um deles possui parâmetros e o outro não. Analisando os métodos, o primeiro (sem parâmetro) soma 10% ao valor do salário e retorna este valor. Já o segundo verifica qual o valor do salário do funcionário (que recebe como parâmetro) e depois decide se vai acrescentar 100 reais ao valor da bonificação ou não.

API DateTime

Muitas vezes é necessário trabalhar com datas, dias da semana, horários, etc... para informar, por exemplo quando um cliente foi cadastrado, quando um saque foi realizado. Para tanto, a partir do Java 8, uma nova API foi criada para facilitar este tratamento. Vamos aqui verificar algumas de suas funcionalidades.

O Excerto de código abaixo é a classe API_DateTime criada para apresentar algumas funcionalidades desta API.

```java
package Missao6;

import java.time.Duration;
import java.time.LocalTime;
import java.time.YearMonth;


public class API_DateTime
{
    public void horario()
    {
        LocalTime tempo;
        //Mostrando a hora atual
        LocalTime HoraAgora = LocalTime.now();
        System.out.println("Hora agora: " + HoraAgora);
        //Subtraindo tempo
        tempo = LocalTime.now().minusHours(5);
        System.out.println("Horário atual menos 5 horas: " + tempo);
        tempo = LocalTime.now().plusHours(4);
        System.out.println("Horário atual mais 4 horas: " + tempo);

        //intervalo de tempo
        Duration tempo2 = Duration.between(HoraAgora, tempo);
        System.out.println("Intervalo de tempo entre dois pontos no tempo: " + tempo2);

        //numeros de dias num mês
        int daysInMonth = YearMonth.of(1990, 2).lengthOfMonth();
        System.out.println("Numero de dias no mes de fevereiro de 1990: " + daysInMonth );

    }

```

API DateTime

Caso você abra a classe Missao6_principal e instancie a classe API_DateTime, chamando o método horário(), o resultado será o seguinte:

```
Hora agora: 11:42:15.167410900
Horário atual menos 5 horas: 06:42:15.167410900
Horário atual mais 4 horas: 15:42:15.167410900
Intervalo de tempo entre dois pontos no tempo: PT4H
Número de dias no mês de fevereiro
```