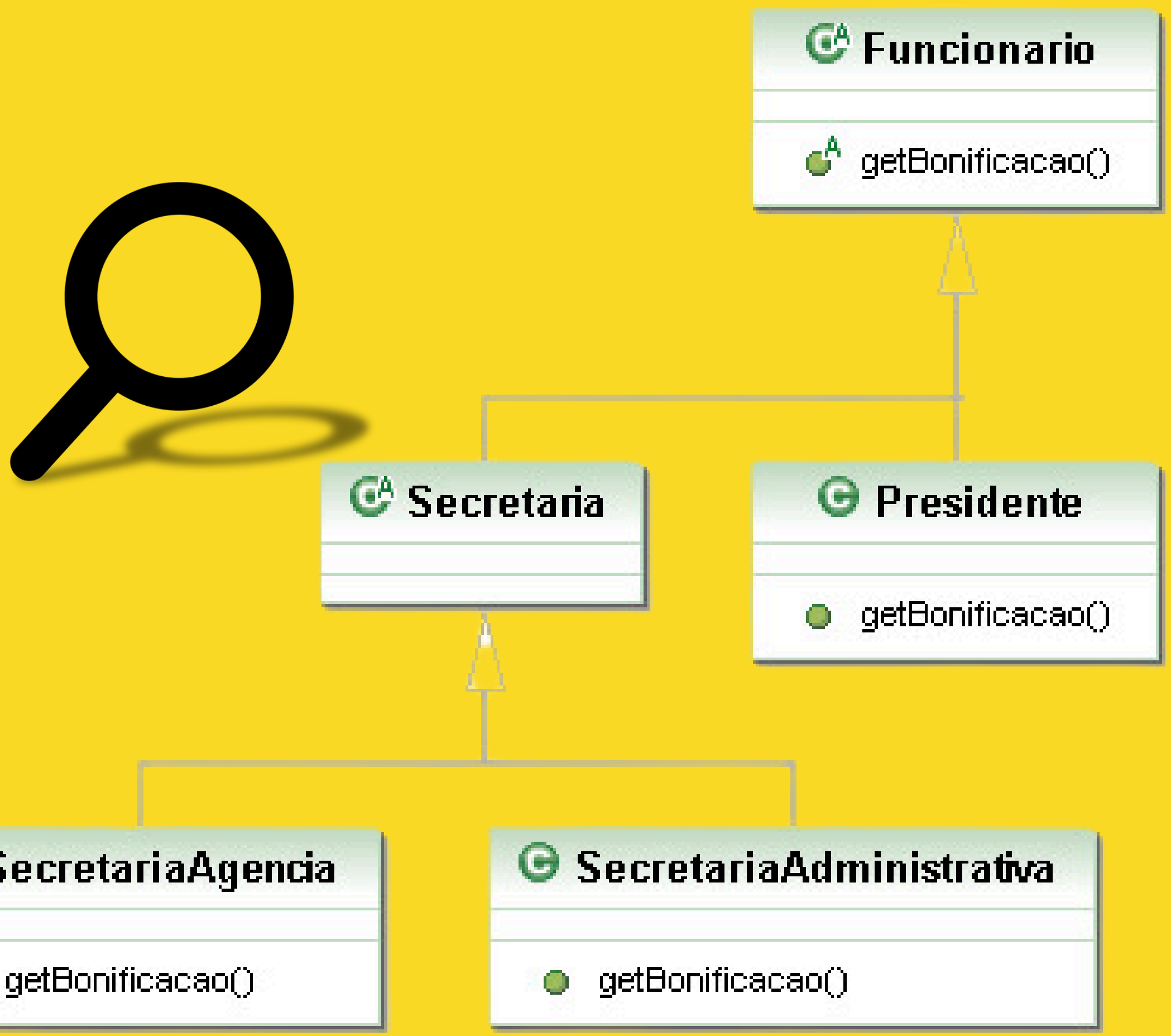
Pilares da Programação Orientada a Objetos

Abstração

Imagine que você está criando um sistema de RH, e precisa criar um método para calcular o salário dos funcionários. Pode parecer um desafio simples, mas e se eu te dissesse que teremos cálculos diferentes para diferentes cargos e setores?

Com a abstração, não precisamos nos preocupar em fazer todos esses métodos, pelo menos não na classe Funcionário. Funcionário seria uma abstração de todos os cargos existentes, informando seus atributos e métodos genéricos.



Quando temos uma classe Abstrata, ela abre mão da responsabilidade de estruturar o método "getBonificação()", passando essa tarefa para as outras classes que irão herdar seus métodos e atributos.

Herança

Como o próprio nome diz, trata-se de uma herança, algo que passa de uma pessoa para outra. No nosso caso, é de uma classe para outra! Por exemplo: métodos e atributos.

Através do conceito de herança, podemos reduzir inúmeras linhas de código, apenas reutilizando atributos ou métodos de classes 'mães'.

Isso é muito útil e nos traz inúmeros benefícios, como: organização de código, hierarquia, facilidade na manutenção e nosso próximo assunto, polimorfismo.

```java
class Animal {
    void dormir() {
        System.out.println("Dormindo...");
    }
}

class Cachorro extends Animal {
    void latir() {
        System.out.println("Au Au!");
    }
}

public class Main {
    public static void main(String[] args) {
        Cachorro dog = new Cachorro();
        dog.dormir();  // Método herdado da classe Animal
        dog.latir();   // Método da própria classe Cachorro
    }
}
```

Neste exemplo, vemos herança, onde a classe Cachorro herda atributos e métodos da classe Animal. A classe Animal pode ser vista como um tipo genérico, e Cachorro como uma especialização desse tipo.

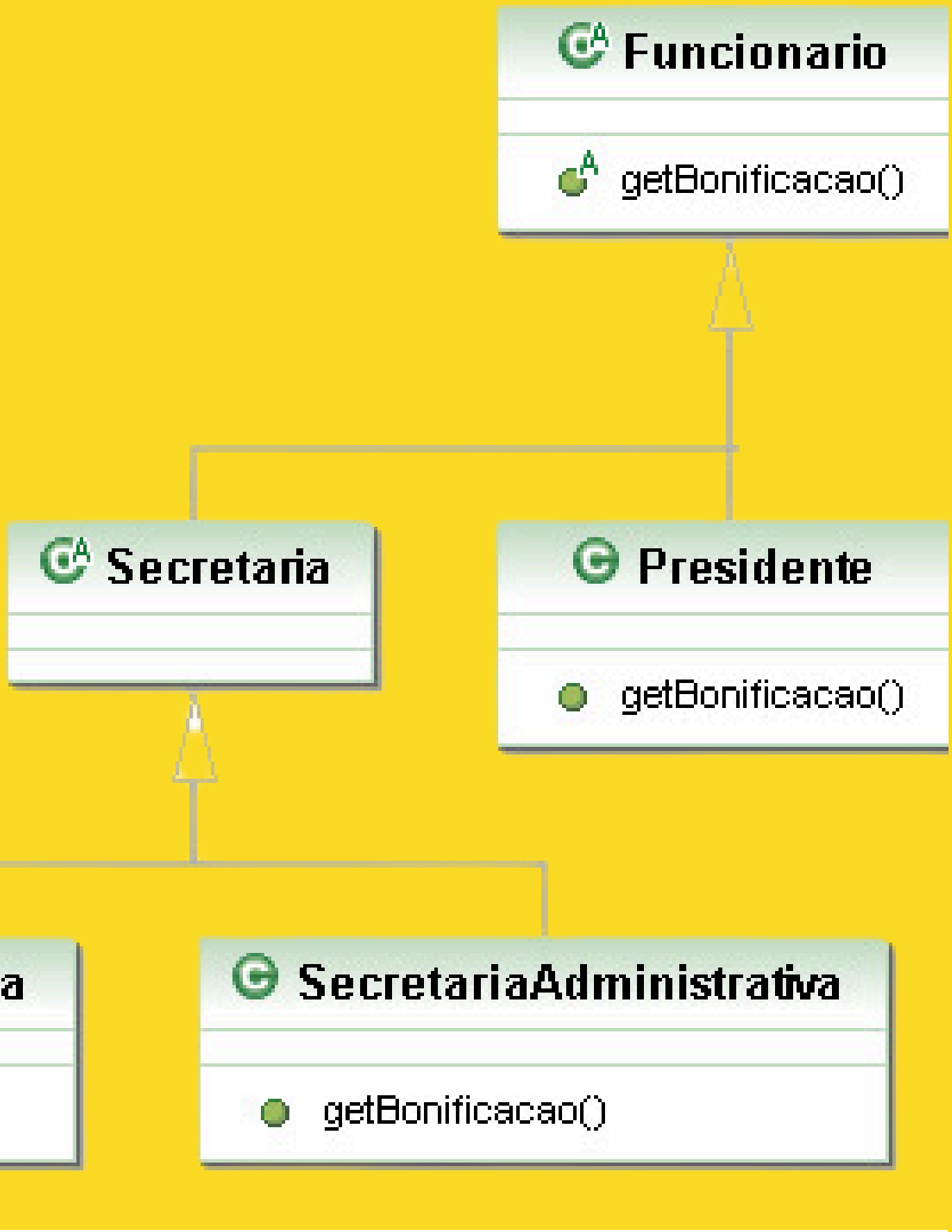
Polimorfismo

E se eu te disser que já vimos polimorfismo aqui hoje? Mas vamos tentar entender o que essa nova palavra significa:

Poli – Muito / Muitos

Morfo – Forma

Podemos entender polimorfismo então como muitas formas! E é exatamente isso! Pudemos ver isso logo no nosso primeiro exemplo. Bora lá conferir.



Aqui podemos ver o polimorfismo, de forma que todas as classes necessitam do método 'getBonificação()', porém, este método é diferente para cada cargo, assumindo assim diversas formas e maneiras de ser estruturado.

Muito prático, não?

Encapsulamento

O encapsulamento em Java está diretamente ligado aos modificadores de acesso: public, protected e private.

Utilizamos essa prática para proteger e manipular informações, assim evitando mexer diretamente nos atributos de uma classe, mas sim, manipulando-os por meio de métodos, também conhecidos como os getters e setters.

```java
public class Pessoa {
    private String nome;
    private int idade;

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public int getIdade() {
        return idade;
    }

    public void setIdade(int idade) {
        this.idade = idade;
    }
}
```

Através do modificador private, garantimos que os atributos só serão acessíveis dentro dessa classe. E com a ajuda de métodos getters e setters para cada atributo, podemos assegurar a integridade das informações manipuladas.