INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA

Classe Abstrata

Expandindo o Sistema

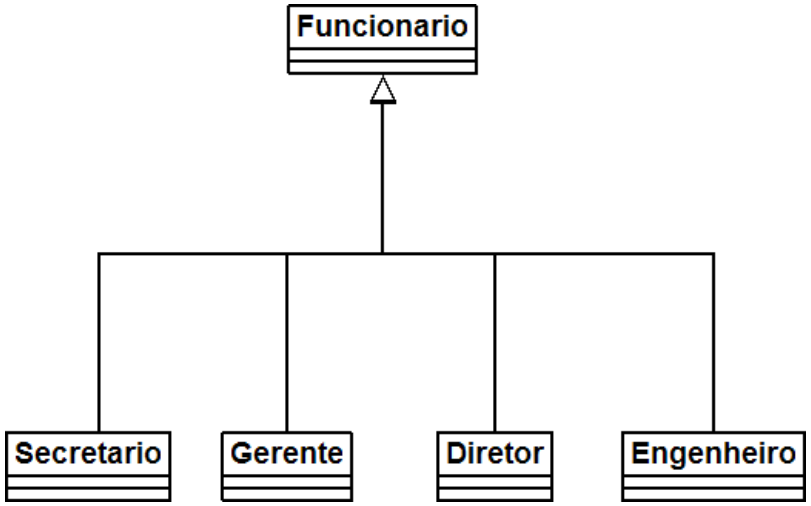
- Imagine que um **Sistema** de **Controle do Banco** pode ser **acessado**, além dos **Gerentes**, pelos **Diretores** do Banco

```
class Diretor extends Funcionario {

    public boolean autentica(int senha) {
        // verifica aqui se a senha confere com a recebida como parametro
    }

}
```

E a classe `Gerente`:

```
class Gerente extends Funcionario {

    public boolean autentica(int senha) {
        // verifica aqui se a senha confere com a recebida como parametro
        // no caso do gerente verifica tambem se o departamento dele
        // tem acesso
    }

}
```

Expandindo o Sistema



- **Considere** o **SistemaInterno** e seu controle: precisamos **receber** um Diretor ou Gerente como **argumento**, **verificar**, **autenticar** e **colocá-lo** dentro do sistema.

```
class SistemaInterno {

    void login(Funcionario funcionario) {
        // invocar o método autentica? não da! Nem todo Funcionario tem
    }
}
```

Expandindo o Sistema


```
class SistemaInterno {

    void login(Funcionario funcionario) {
        funcionario.autentica(...); // não compila
    }
}
```

- Uma **possibilidade** é **criar dois métodos login** no SistemaInterno: um para receber Diretor e outro para receber Gerente. Já vimos que essa não é uma boa **escolha**. Por que?

```
class SistemaInterno {

    // design problemático
    void login(Diretor funcionario) {
        funcionario.autentica(...);
    }

    // design problemático
    void login(Gerente funcionario) {
        funcionario.autentica(...);
    }
```

Possível Solução

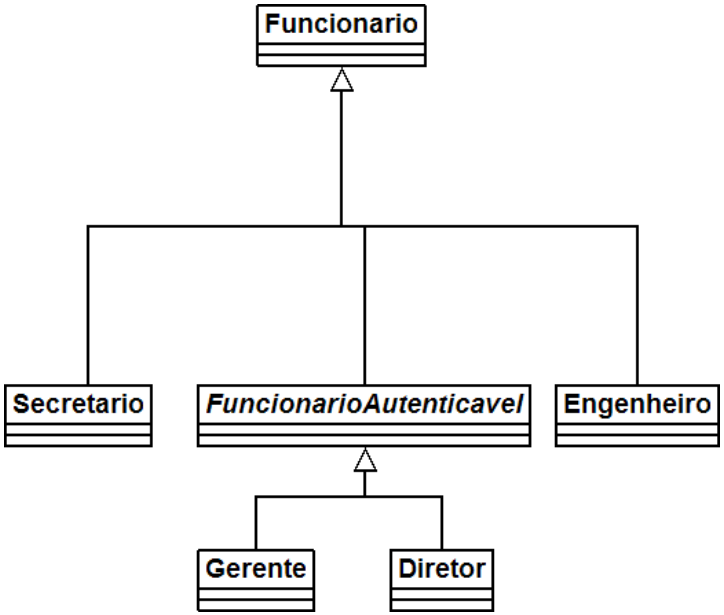
- Uma solução mais **interessante** seria criar uma classe no meio da **árvore** de **herança**, FuncionarioAutenticavel:

```
class FuncionarioAutenticavel extends Funcionario {

    public boolean autentica(int senha) {
        // faz autenticacao padrao
    }

    // outros atributos e metodos

}
```

- As classes **Diretor** e **Gerente** passariam a **estender** de **FuncionarioAutenticavel**, e o SistemaInterno receberia referências desse tipo, como a seguir:

```
class SistemaInterno {

    void login(FuncionarioAutenticavel fa) {

        int senha = //pega senha de um lugar, ou de um scanner de polegar

        // aqui eu posso chamar o autentica!
        // Pois todo FuncionarioAutenticavel tem
        boolean ok = fa.autentica(senha);

    }
```

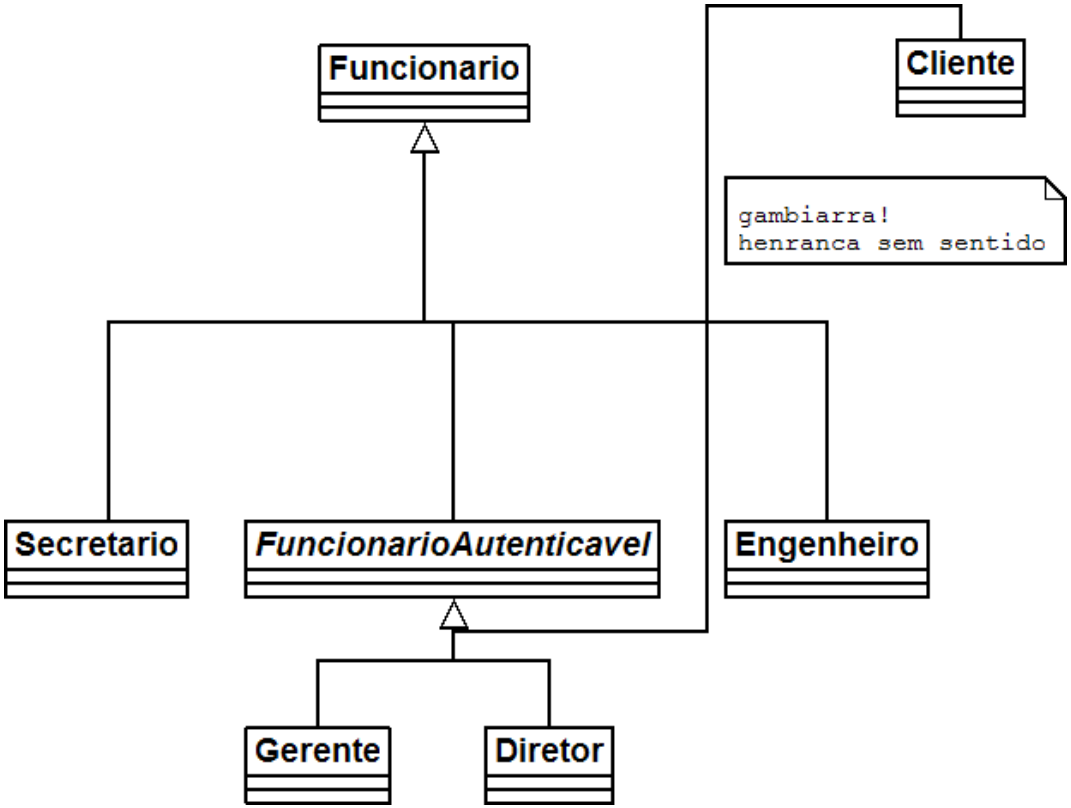
Possível Solução



- **FuncionarioAutenticavel** é uma **forte candidata** a **classe abstrata**. Mais ainda, o **método autentica** poderia ser um **método abstrato**.
- O uso de **herança resolve** esse **caso**, mas se precisamos que todos os **clientes** também tenham acesso ao **SistemaInterno**

Classe Abstrata

- A **solução aplicada** por novos **programadores** é fazer uma **herança** sem **sentido** para resolver o **problema**, por exemplo, fazer **Cliente** extends **FuncionarioAutenticavel**.



Classe Abstrata

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- Mas qual é a **real vantagem** de uma classe **abstrata**? Poderíamos ter feito isto com uma **herança comum**.

- 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**.

Métodos Abstratos

- Se o método **getBonificacao** não fosse **reescrito**, ele seria **herdado** da classe mãe, **fazendo** com que **devolvesse** o **salário** mais **20%**.

- 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**?

Métodos Abstratos

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- **Poderíamos**, então, **jogar fora** esse método da **classe** Funcionario?

- O **problema** é **que**, se ele não **existisse**, não **poderíamos** chamar o **método** apenas com uma **referência** a um **Funcionario**, pois **ninguém garante** que essa **referência** aponta para um **objeto** que **possui** esse **método**.

Método Abstrato

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

- 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 metodos

}
```

Método Abstrato

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA

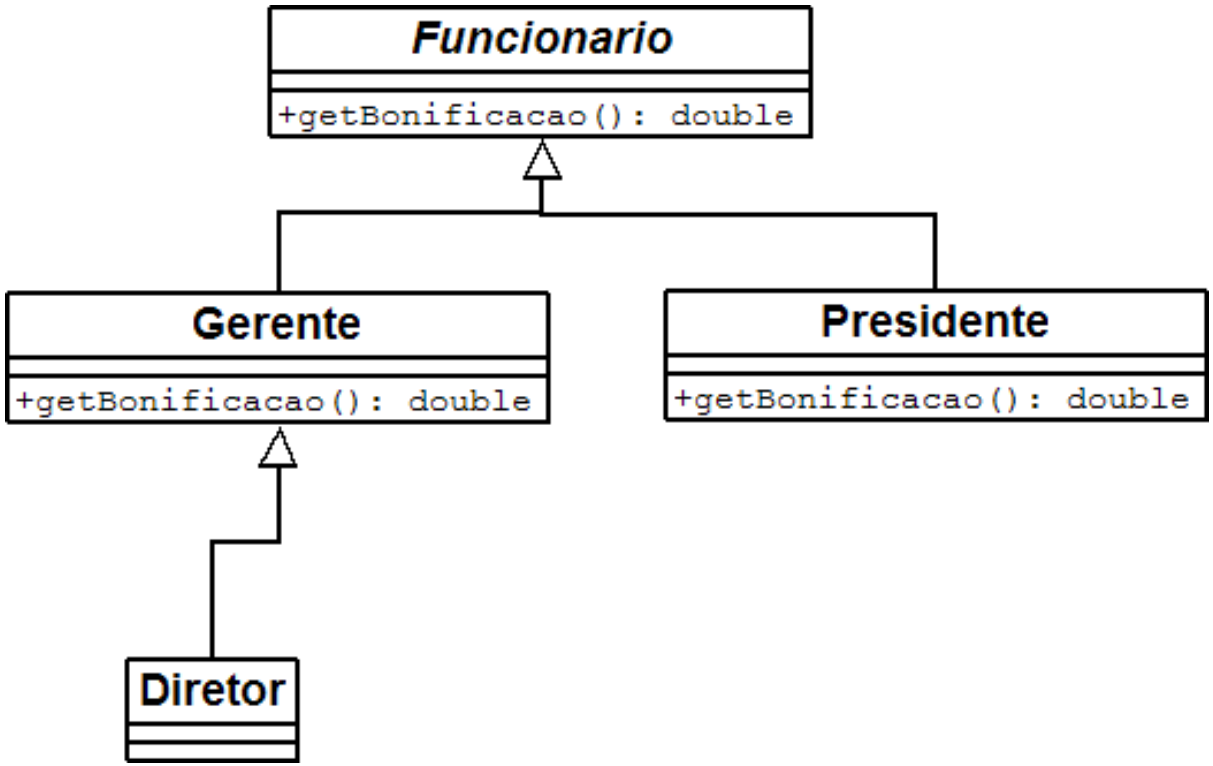
- Repare que não **colocamos** o **corpo** do **método** e usamos a palavra **chave** **abstract** para definir o mesmo.

- 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".

Invocando o método reescrito

- E se, no nosso exemplo de empresa, tivéssemos o seguinte diagrama de classes com os seguintes métodos:



- Essas classes vão compilar? Vão rodar?