Interfaces

"Uma imagem vale mil palavras. Uma interface vale mil imagens."
-- Ben Shneiderman

Não são telas ou interfaces graficas

É um **tipo especial** de classe contendo **métodos abstratos** e **atributos finais**

Notação UML

Interfaces por natureza **são abstratas**

Define um meio público e padrão de **especificar** o **comportamento** das classes

Criando Interfaces

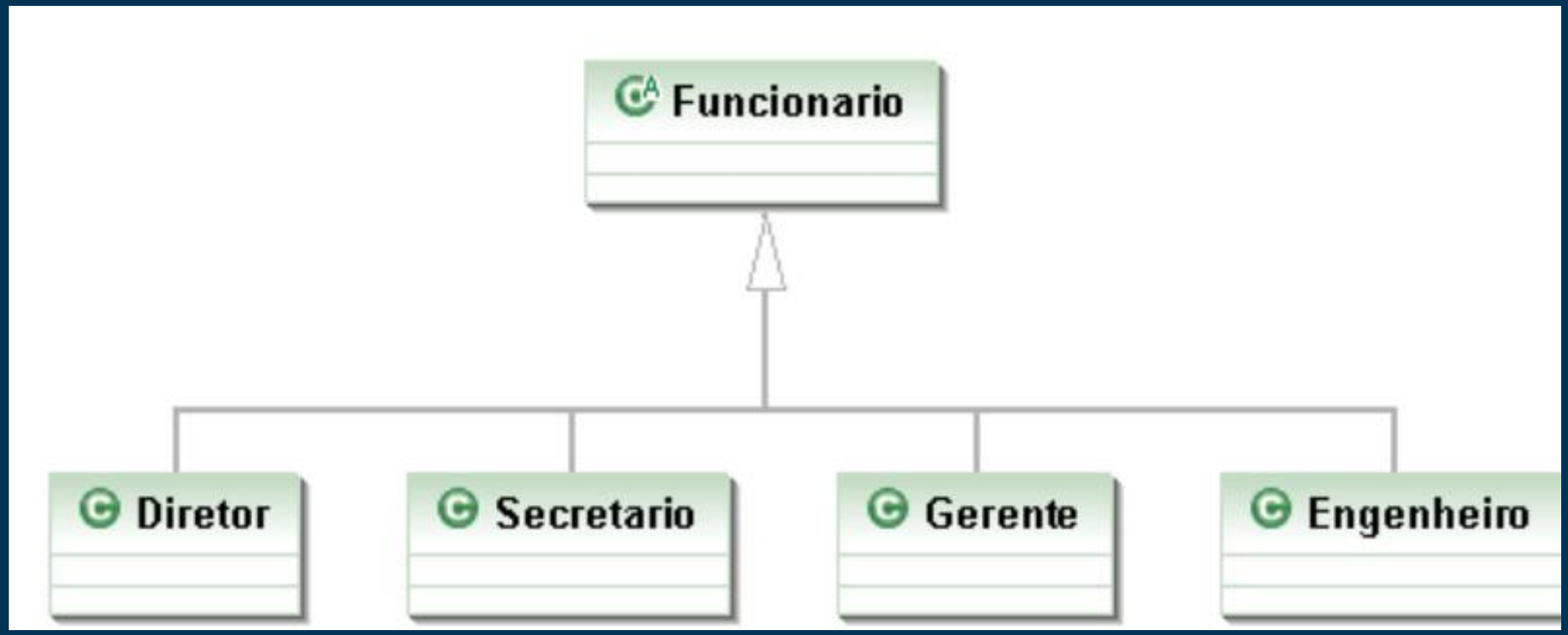
Imagine que um Sistema de Controle do Banco pode ser acessado, além de pelos Gerentes, pelos Diretores do Banco. Então, teríamos uma classe Diretor:


```java
public class Diretor extends Funcionario {

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

```java
public 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 também se o departamento dele
         tem acesso*/
        return false;
    }
}
```

Implementando Interfaces

Palavra reservada **implements** e usada
para implementar uma interface



Interfaces

Interfacesnão são partesda hierarquiade classe. Entretanto, interfaces podem ter relacionamentos de **herança entre elas** próprias

```
public class SistemaInterno {

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

Considere o SistemaInterno e seu controle: precisamos receber um Diretor ou Gerente como argumento, verificar se ele se autentica e colocá-lo dentro do sistema

Interfaces

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.

```
public class SistemaInterno {

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

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

}
```

Interfaces

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

```
public class FuncionarioAutenticavel extends Funcionario {

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

    // outros atributos e métodos

}
```

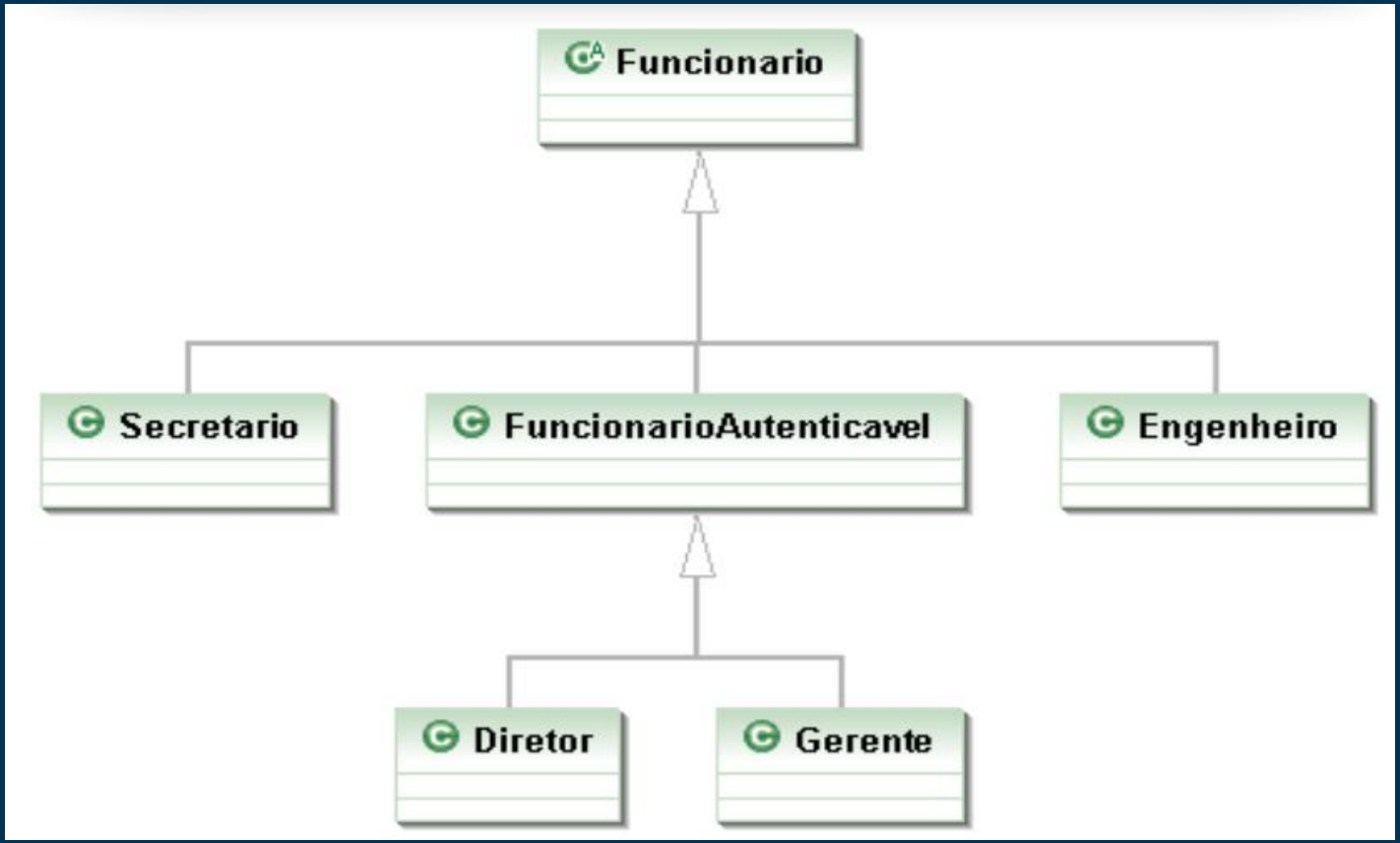
Interfaces

As classes Diretor e Gerente passariam a estender de FuncionarioAutenticavel, e o SistemaInterno receberia referências desse tipo

```
public class SistemaInterno {

    public void login(FuncionarioAutenticavel fa) {

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

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

    }
}
```

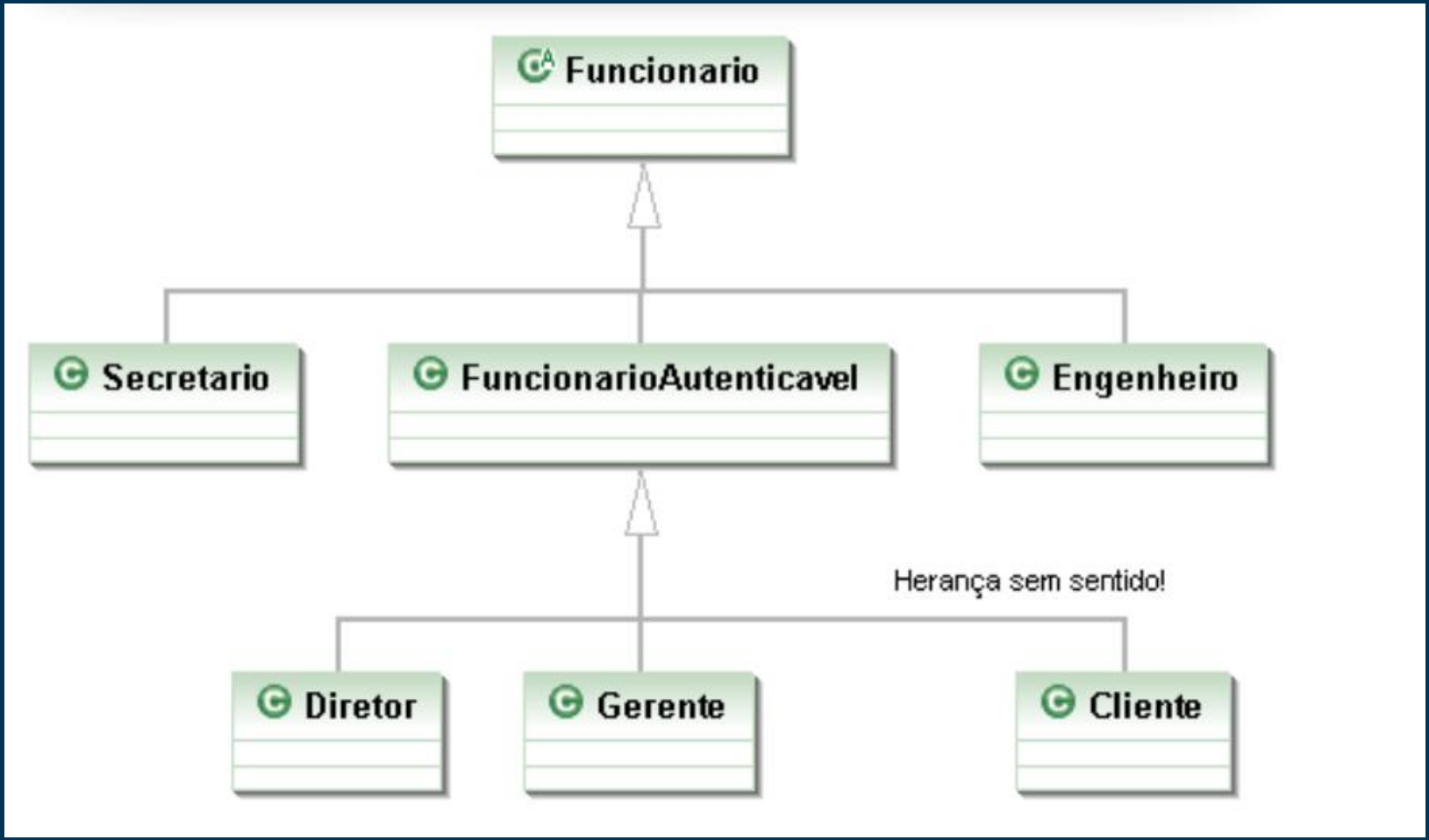
Interfaces

Repare que FuncionarioAutenticavel é uma forte candidata a classe abstrata. Mais ainda, o método autentica poderia ser um método abstrato.



Interface

O nosso Problema?



Interfaces

O que precisamos para resolver nosso problema? Arranjar uma forma de poder referenciar Diretor, Gerente e Cliente de uma mesma maneira, isto é, achar um fator comum.

Se existisse uma forma na qual essas classes garantissem a existência de um determinado método, através de um contrato, resolveríamos o problema.

Toda classe define 2 itens:

- o que uma classe faz (as assinaturas dos métodos)
- como uma classe faz essas tarefas (o corpo dos métodos e atributos privados)

Podemos criar um "contrato" que define tudo o que uma classe deve fazer se quiser ter um determinado status. Imagine:

Interfaces

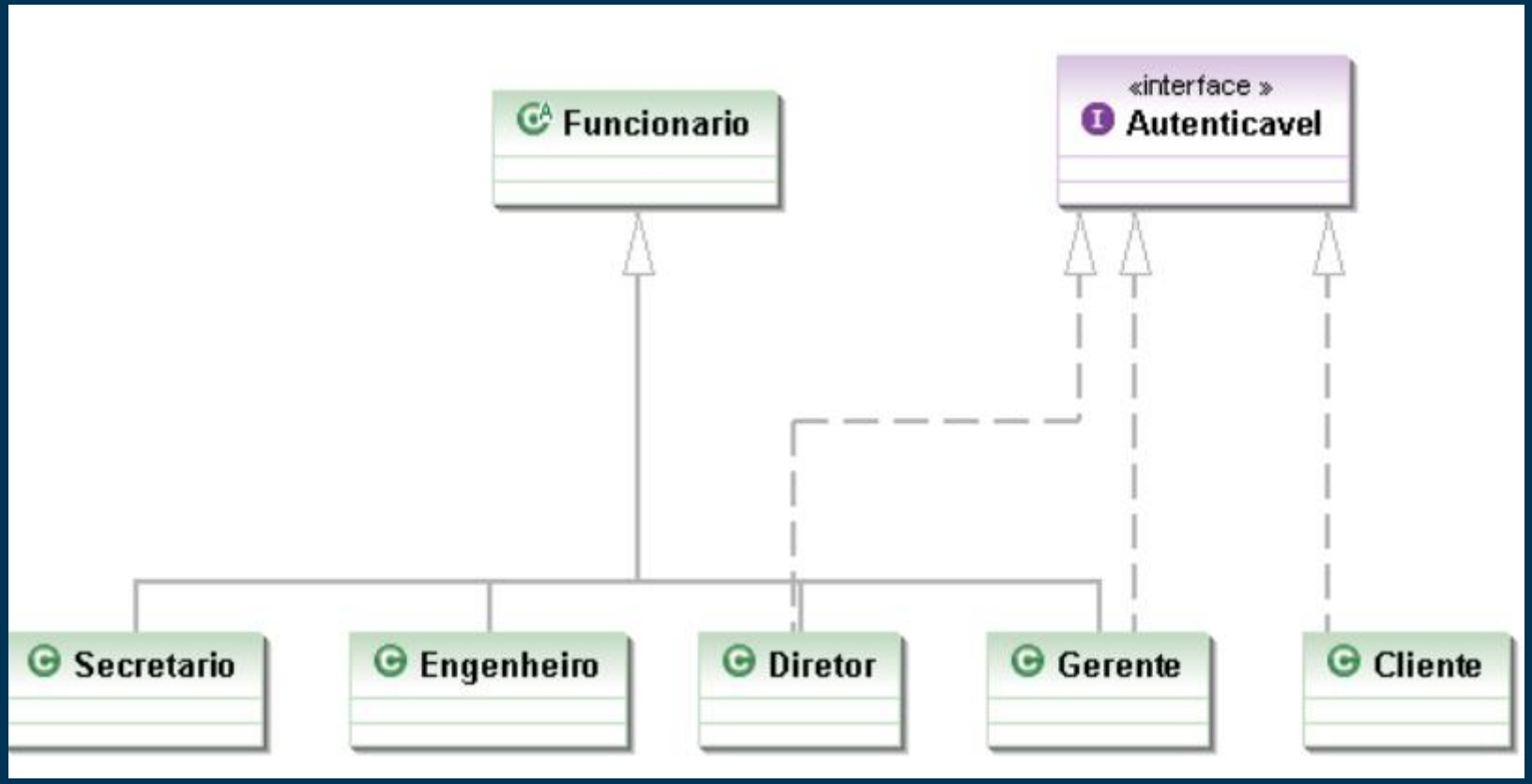
Quem quiser, pode "assinar" esse contrato, sendo assim obrigado a explicar como será feita essa autenticação. A vantagem é que, se um Gerente assinar esse contrato, podemos nos referenciar a um Gerente como um Autenticavel.

Podemos criar esse contrato em Java!

```
public interface Autenticavel {

    boolean autentica(int senha);

}
```

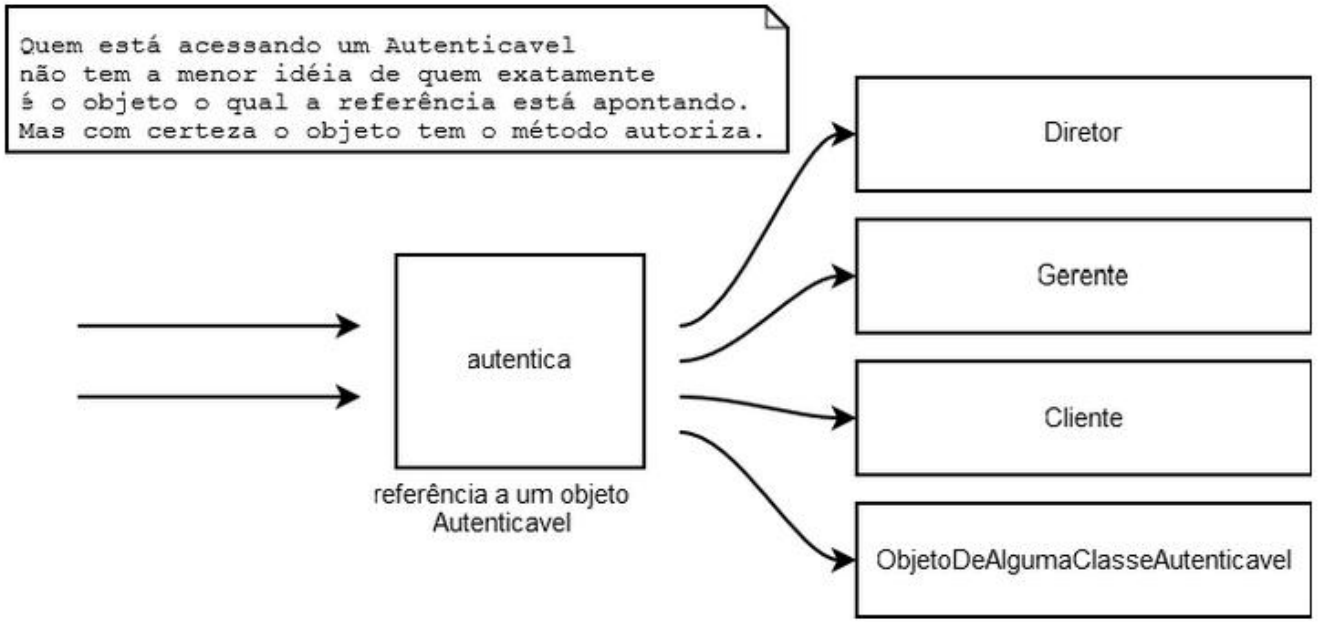
Uma interface pode definir uma série de métodos, mas nunca conter implementação deles. Ela só expõe o que o objeto deve fazer

Interfaces

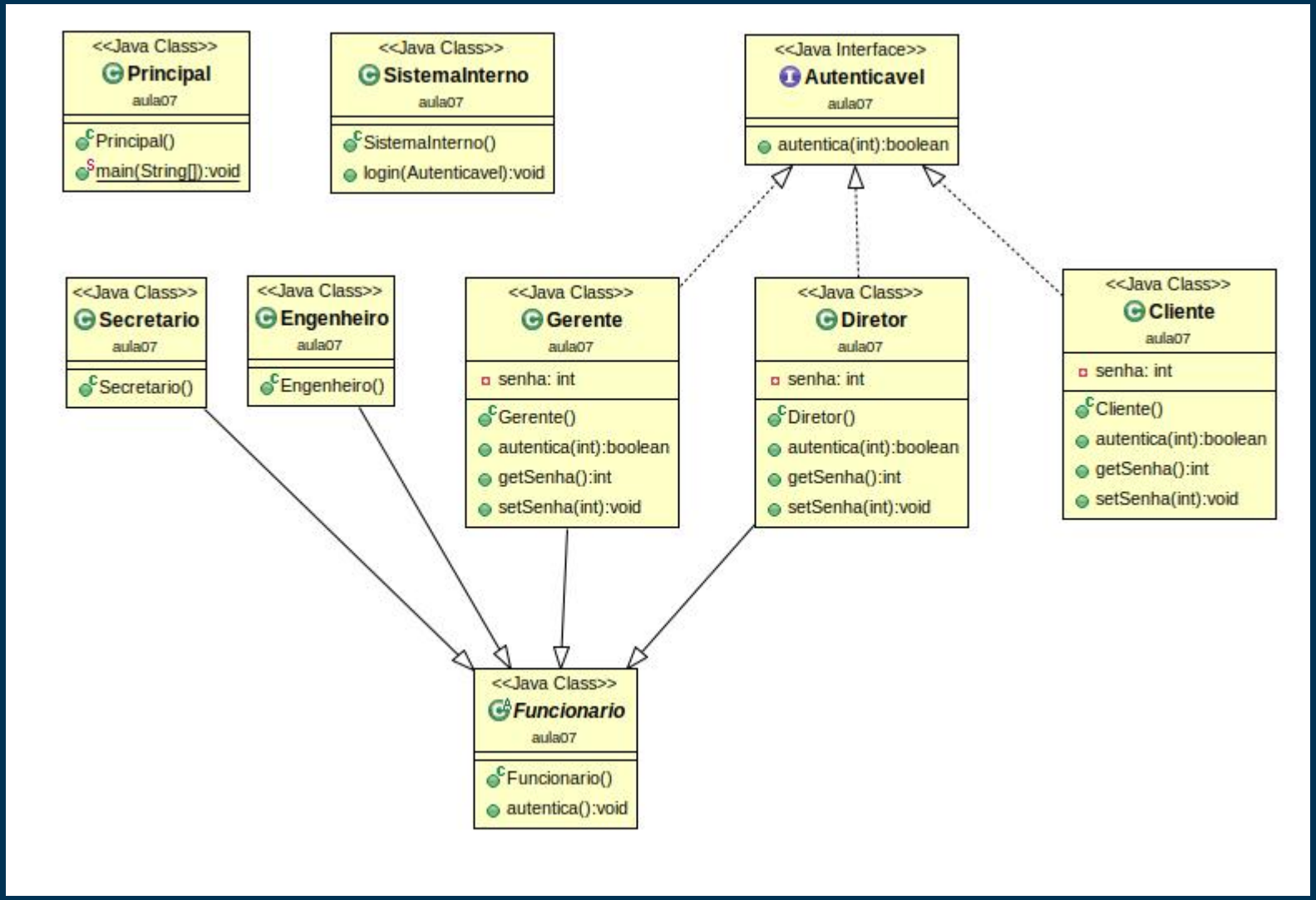

Funcionario
«interface »
Autenticavel
Secretario
Engenheiro
Diretor
Gerente
Cliente

Interface

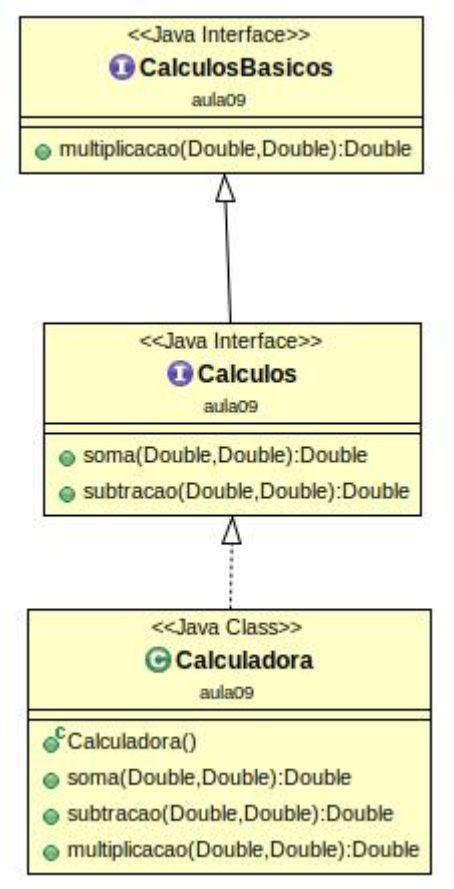
```
public class Diretor extends Funcionario implements Autenticavel {

    // métodos e atributos, além de obrigatoriamente ter o autentica

}
```



Interface


<<Java Class>>
Principal
aula07
Principal()
main(String[]):void
<<Java Class>>
SistemaInterno
aula07
SistemaInterno()
login(Autenticavel):void
<<Java Interface>>
Autenticavel
aula07
autentica(int):boolean
<<Java Class>>
Secretario
aula07
Secretario()
<<Java Class>>
Engenheiro
aula07
Engenheiro()
<<Java Class>>
Gerente
aula07
senha: int
Gerente()
autentica(int):boolean
getSenha():int
setSenha(int):void
<<Java Class>>
Diretor
aula07
senha: int
Diretor()
autentica(int):boolean
getSenha():int
setSenha(int):void
<<Java Class>>
Cliente
aula07
senha: int
Cliente()
autentica(int):boolean
getSenha():int
setSenha(int):void
<<Java Class>>
Funcionario
aula07
Funcionario()
autentica():void

Atividade01



Atividade02