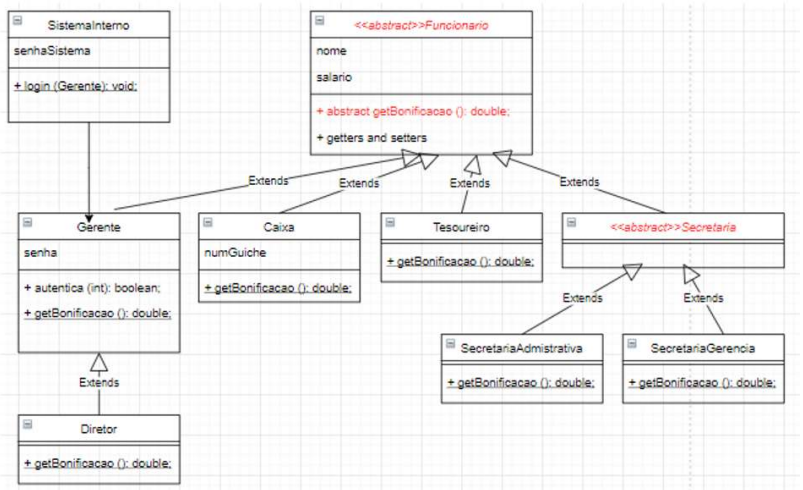
**Aula 09**
**Exercício projeto bancov010. Exercício para ser feito em classe.**

1) Importe o projeto bancov010 fornecido pelo professor (import → exists projects into...). Esse projeto não tem a lógica das classes Conta. Abra o projeto e feche todos os outros.
2) Observe bem a hierarquia abaixo.

Gerente e Diretor podem se logar no SistemaInterno




3) Crie a classe SistemaInterno com o atributo senhaDoSistema e o método login que deve aceitar Gerente e Diretor.

```
3  public class SistemaInterno {
4      private int senhaDoSiatema = 123;
5
6      public void login(Gerente g) {
7          if(g.autentica(this.senhaDoSiatema)==true)
8              System.out.println("Acesso liberado!");
9          else
10             System.out.println("Acesso negado.");
11     }
12 }
```

4) Crie a classe testaSistemaInterno com 2 gerentes e um diretor e verifique se tudo está funcionando como deveria.

```
7  public class TestaSistemaInterno {
8      public static void main(String[] args) {
9          SistemaInterno si = new SistemaInterno();
10         Gerente g1,g2;
11         g1 = new Gerente();
12         g2 = new Gerente();
13         g1.setNome("Rafael");
14         g1.setSenha(123);
15         g2.setNome("Renata");
16         g2.setSenha(124);
17         Diretor d1 = new Diretor();
18         d1.setNome("Reinaldo");
19         d1.setSenha(123);
20
21         si.login(g1);
22         si.login(g2);
23         si.login(d1);
24     }
25 }
```

5) Novo problema. Tesoureiro precisa se logar no sistema. Ocorre que Tesoureiro não é um Gerente e se eu afrouxar a assinatura do método p/ Funcionário, permitirei que Caixa e outros tipos de Funcionario consigam logar no sistema. Outro problema é que Tesoureiro não possui senha e nem o método autentica. Por enquanto, adicione o atributo senha e seus métodos get e set, além do método autentica.

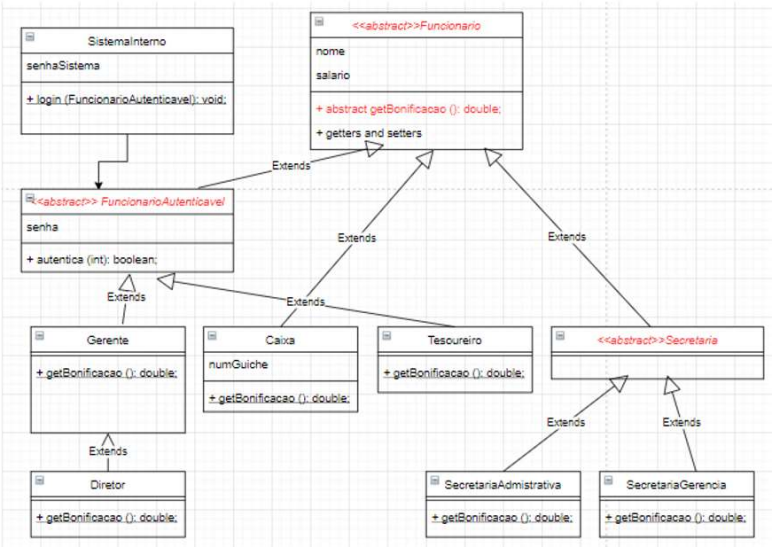
```
3  public class Tesoureiro extends Funcionario {
4      private int senha;
5
6      public int getSenha() {
7          return this.senha;
8      }
9      public void setSenha(int senha) {
10         this.senha = senha;
11     }
12     public boolean autentica(int senha) {
13         if (this.senha == senha)
14             return true;
15         return false;
16     }
17     @Override
18     public double getBonificacao() {
19         //super é mais descritivo uma vez que o atributo está na superclasse
20         return super.salario * 0.20;
21     }
22 }
```

6) A partir de TestaSistemaInterno, crie TestaSistemaInterno2 que possui um Tesoureiro. Obviamente ainda não podemos passar um Tesoureiro pelo método login (linha 28).

```
8  public class TestaSistemaInterno2 {
9      public static void main(String[] args) {
10         SistemaInterno si = new SistemaInterno();
11         Gerente g1,g2;
12         g1 = new Gerente();
13         g2 = new Gerente();
14         g1.setNome("Rafael");
15         g1.setSenha(123);
16         g2.setNome("Renata");
17         g2.setSenha(124);
18         Diretor d1 = new Diretor();
19         d1.setNome("Reinaldo");
20         d1.setSenha(123);
21         Tesoureiro t = new Tesoureiro();
22         t.setNome("Nilzete");
23         t.setSenha(123);
24
25         si.login(g1);
26         si.login(g2);
27         si.login(d1);
28         si.login(t);//Não compila...
29     }
30 }
```

7) Nesse momento é importante você perceber que apesar de precisarmos de uma "plaquinha" que sirva para Gerente (e Diretor) e Tesoureiro, sobrecarregar o método login não é uma boa solução. Estaríamos duplicando uma regra.
8) Uma solução mais inteligente pode ser utilizada. Vamos utilizar os conceitos que aprendemos recentemente. Vamos obter polimorfismo por meio de uma classe abstrata. Veja.

FuncionarioAutenticavel pode se logar no SistemaInterno




9) Crie a classe abstrata FuncionarioAutenticavel que herda de Funcionario. Retire toda a lógica de autenticação da classe Gerente e coloque em FuncionarioAutenticavel. Apague essa mesma lógica da classe Tesoureiro e faça as duas classes herdarem de FuncionarioAutenticavel. Veja como devem ficar as 3 classes.

FuncionarioAutenticavel

```
3  public abstract class FuncionarioAutenticavel extends Funcionario {
4      protected int senha;//Não se esqueça de mudar o nível de visibilidade!!
5
6      public int getSenha() {
7          return this.senha;
8      }
9      public void setSenha(int senha) {
10         this.senha = senha;
11     }
12     public boolean autentica(int senha) {
13         if (this.senha == senha)
14             return true;
15         return false;
16     }
17 }
```

Gerente

```
3  public class Gerente extends FuncionarioAutenticavel {
4
5      @Override
6      public double getBonificacao() {
7          return this.salario * 0.30;
8      }
9  }
```

Tesoureiro

```
3  public class Tesoureiro extends FuncionarioAutenticavel {
4
5      @Override
6      public double getBonificacao() {
7          //super é mais descritivo uma vez que o atributo está na superclasse
8          return super.salario * 0.20;
9      }
10 }
```

10) Por fim, para que TestaSistemaInterno2 funcione corretamente, basta modificar a "plaquinha" do método login de SistemaInterno para FuncionarioAutenticavel. Veja!

```
3  public class SistemaInterno {
4      private int senhaDoSiatema = 123;
5
6      public void login(FuncionarioAutenticavel fa) {
7          if(fa.autentica(this.senhaDoSiatema)==true)
8              System.out.println("Acesso liberado!");
9          else
10             System.out.println("Acesso negado.");
11     }
12 }
```

11) Rode a classe TestaSistemaInterno2 novamente e verá que tudo está resolvido. É o poder do polimorfismo sendo utilizado por meio de uma classe abstrata! Agora se algum outro tipo de Funcionario precisar se autenticar no sistema, basta herdar de FuncionarioAutenticavel. Percebam que nossa lógica de contabilizar bonificações permaneceu intacta mesmo após todas essas mudanças. GerenciadorDeBonificacoes e TestaGerenciadorDeBonificacoes continuam funcionando normalmente. Fizemos várias alterações e o impacto no sistema, como prometido anteriormente, foi ZERO!