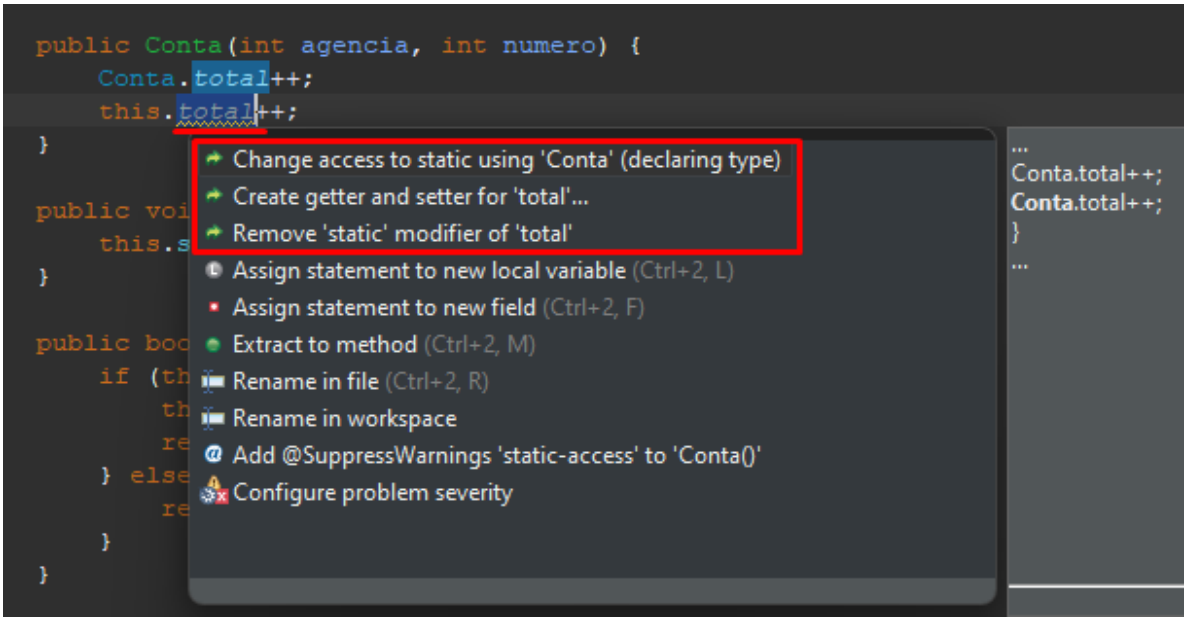
Herança

Anotações de coisas avulsas notadas:

Quando se coloca um atributo ou método como static a gente não pode chama-lo com this.atributo ou com o (objeto).atributo (lembrando que objeto é uma classe já instanciada) e sim com (classe).atributo. Ex:



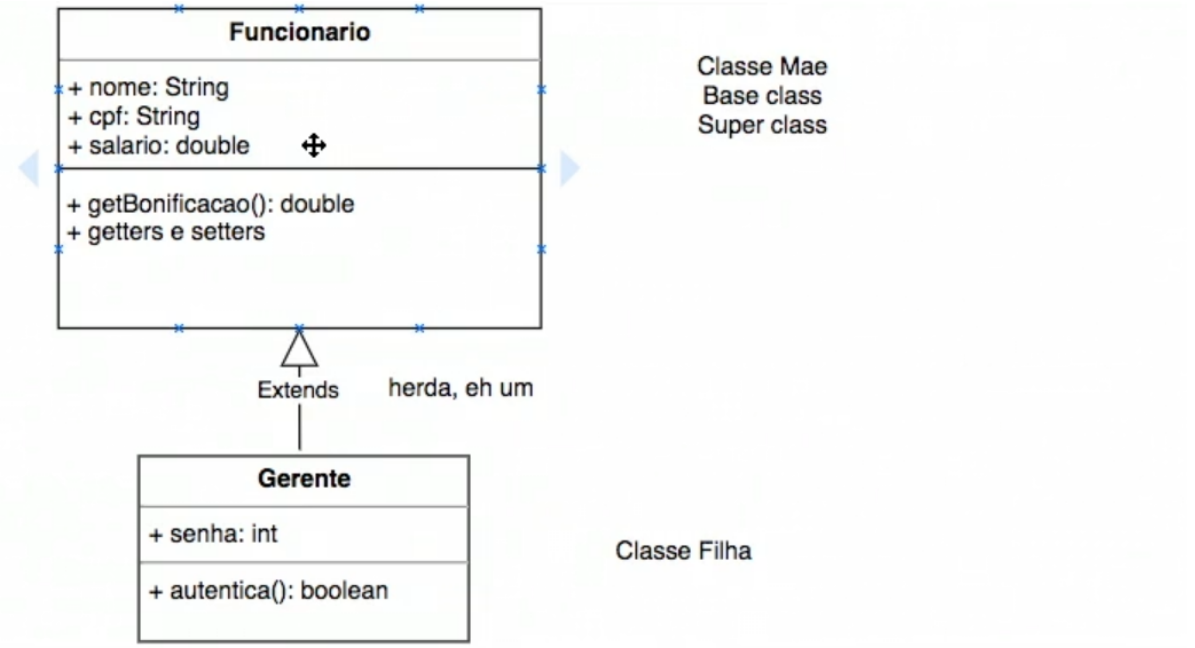
Herança também é um relacionamento entre classes, composição por exemplo é um relacionamento entre classes que é quando damos a um atributo o tipo de outra classe nossa, fugindo do convencional que é String, int, boolean, etc. Ex:

- ```
  private int numero;
  private Cliente titular;
  ```

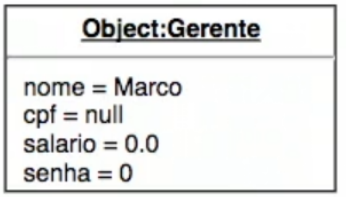
Se não criamos nenhum construtor o compilador automaticamente insere um construtor padrão que é o vazio.

Ao criamos uma classe Funcionário ela deve conter atributos que tem na classe gerente e diretor, para não precisarmos em cada uma repetir métodos e atributos a gente na gerente e diretor estende a funcionário que aí conseguimos usar os métodos e atributos dela nessas classes.

Assim segue o nosso projeto até o momento:
  ```




```
Gerente g1 = new Gerente();
g1.setNome("Marco");
```



Temos como usar o protected no atributo salario na classe Funcionário para escrever que o método getBonificacao() do gerente seja sem a comissão e apenas o salário.

```
3 usages  1 inheritor
public class Funcionario {

    2 usages
    private String nome;
    2 usages
    private String cpf;
    4 usages
    protected double salario;
```

O protected diz que tal atributo ou método ou até mesmo classe pode somente ser acessado por classes que estendem daquela classe ou seja **classes filhas**. **(Mas não é uma boa prática, pois é melhor usarmos os métodos get and setters).**

Ao puxarmos algum atributo ou método da classe mãe em classes filhas é uma boa prática usarmos o super ao invés de this, pois this é usado geralmente para nos referirmos ao que se tem na classe em questão e o super é para falar que estamos pegando aquilo da classe mãe.

A **assinatura** de um **método** é exatamente o seu tipo de acesso, o tipo de retorno que ele vai retornar, o nome e os parâmetros, exemplo abaixo:

```
public double getBonificacao()
```

Essa assinatura por boas práticas não alteramos ela quando reescrevemos o método, somente em algumas regrinhas que podemos alterar.

```
1 usage
public double getBonificacao() {
    return super.salario;
    // return this.salario;
}
```

Super com métodos:

Podemos também chamar um método da **classe mãe** (**super class**) com o **super** e usar ele dentro deste mesmo método só que reescrito na **classe filha**. Ex:

```
public double getBonificacao() {
    return (super.getBonificacao()) + super.salario;
}
```