Orientação a Obejtos Classica

Namom Alves Alencar

Criando e usando um objeto

- Já temos uma classe em Java que especifica o que todo objeto dessa classe deve ter. Mas como usá-la? Além dessa classe, ainda teremos o Programa.java e a partir dele é que vamos utilizar a classe Conta.

- Para criar (construir, instanciar) uma Conta, basta usar a palavra chave new. Devemos utilizar também os parênteses, que descobriremos o que fazem exatamente em um capítulo posterior:

```
class Programa {
        public static void main(String[] args) {
                new Conta();
        }
}
```

Criando e usando um objeto

- Bem, o código acima cria um objeto do tipo Conta, mas como acessar esse objeto que foi criado? Precisamos ter alguma forma de nos referenciarmos a esse objeto. Precisamos de uma variável:

```
class Programa {
        public static void main(String[] args) {
                Conta minhaConta;
                minhaConta = new Conta();
        }
}
```

Criando e usando um objeto

- Pode parecer estranho escrevermos duas vezes Conta: uma vez na declaração da variável e outra vez no uso do new. Mas há um motivo, que em breve entenderemos.
- Através da variável minhaConta, podemos acessar o objeto recém criado para alterar seu dono, seu saldo, etc:

```
class Programa {
        public static void main(String[] args) {
                Conta minhaConta;
                minhaConta = new Conta();
                minhaConta.dono = "Duke";
                minhaConta.saldo = 1000.0;
                System.out.println("Saldo atual: " + minhaConta.saldo);
        }
}
```

- É importante fixar que o ponto foi utilizado para acessar algo em minhaConta. A minhaConta pertence ao Duke, e tem saldo de mil reais.

MÉTODOS

- Dentro da classe, também declararemos o que cada conta faz e como isto é feito - os comportamentos que cada classe tem, isto é, o que ela faz. Por exemplo, de que maneira que uma Conta saca dinheiro? Especificaremos isso dentro da própria classe Conta, e não em um local desatrelado das informações da própria Conta. É por isso que essas "funções" são chamadas de métodos. Pois é a maneira de fazer uma operação com um objeto.
- Queremos criar um método que **saca** uma determinada **quantidade** e não devolve **nenhuma informação** para quem acionar esse método:

MÉTODOS

```
class Conta {
        double salario;
        // ... outros atributos ...

        void saca(double quantidade) {
                double novoSaldo = this.saldo - quantidade;
                this.saldo = novoSaldo;
        }
}
```

- A palavra chave void diz que, quando você pedir para a conta sacar uma quantia, nenhuma informação será enviada de volta a quem pediu.

- Quando alguém pedir para sacar, ele também vai dizer quanto quer sacar. Por isso precisamos declarar o método com algo dentro dos parênteses - o que vai aí dentro é chamado de argumento do método (ou parâmetro). Essa variável é uma variável comum, chamada também de temporária ou local, pois, ao final da execução desse método, ela deixa de existir.

MÉTODOS

- Dentro do método, estamos declarando uma nova variável. Essa variável, assim como o argumento, vai morrer no fim do método, pois este é seu escopo. No momento que vamos acessar nosso atributo, usamos a palavra chave this para mostrar que esse é um atributo, e não uma simples variável. (veremos depois que é opcional)

- Repare que, nesse caso, a conta pode estourar o limite fixado pelo banco. Mais para frente, evitaremos essa situação, e de uma maneira muito elegante.

- Da mesma forma, temos o método para depositar alguma quantia:

MÉTODOS

```
class Conta {
        // ... outros atributos e métodos ...
        void deposita(double quantidade) {
                this.saldo += quantidade;
        }
}
```

- Observe que não usamos uma variável auxiliar e, além disso, usamos a abreviação += para deixar o método bem simples. O += soma quantidade ao valor antigo do saldo e guarda no próprio saldo, o valor resultante.

- Para mandar uma mensagem ao objeto e pedir que ele execute um método, também usamos o ponto. O termo usado para isso é invocação de método.

- O código a seguir saca dinheiro e depois deposita outra quantia na nossa conta:

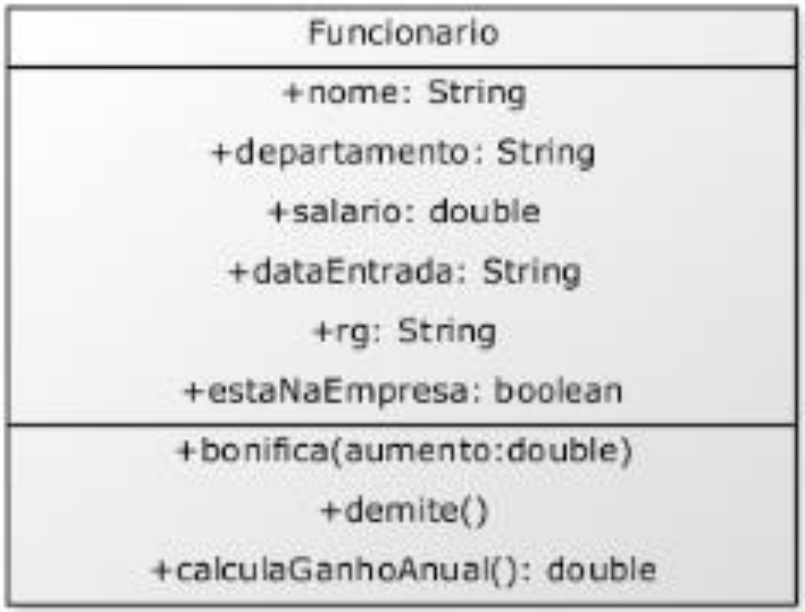
Exercícios

1. Modele um funcionário. Ele deve ter o nome do funcionário, o departamento onde trabalha, seu salário (double), a data de entrada no banco (String) e seu RG (String).
2. Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método recebeAumento que aumenta o salario do funcionário de acordo com o parâmetro passado como argumento. Crie também um método calculaGanhoAnual, que não recebe parâmetro algum, devolvendo o valor do salário multiplicado por 12.

A ideia aqui é apenas modelar, isto é, só identifique que informações são importantes e o que um funcionário faz. Desenhe no papel tudo o que um Funcionario tem e tudo que ele faz.

Exercícios

3. Teste-a, usando uma outra classe que tenha o main. Você deve criar a classe do funcionário com o nome Funcionario, mas pode nomear como quiser a classe de testes, contudo, ela deve possuir o método main. Incremente essa classe. Faça outros testes, imprima outros atributos e invoque os métodos que você criou a mais.



Exercícios

4. Crie um método mostra(), que não recebe nem devolve parâmetro algum e simplesmente imprime todos os atributos do nosso funcionário. Dessa maneira, você não precisa ficar copiando e colando um monte de System.out.println() para cada mudança e teste que fizer com cada um de seus funcionários, você simplesmente vai fazer:

Exercícios

4. Construa dois funcionários com o new e compare-os com o ==. E se eles tiverem os mesmos atributos? Dica: criar outra referência

5. Crie duas referências para o mesmo funcionário, compare-os com o ==. Tire suas conclusões. Dica: Referenciar o segundo funcionario com o primeiro.

Exercícios

6. Em vez de utilizar uma String para representar a data, crie uma outra classe, chamada Data. Ela possui 3 campos int, para dia, mês e ano. Faça com que seu funcionário passe a usá-la. (é parecido com o último exemplo, em que a Conta passou a ter referência para um Cliente).

7. Modifique sua classe TestaFuncionario para que você crie uma Data e atribua ela ao Funcionario.

8. Modifique seu método mostra para que ele imprima o valor da dataDeEntrada daquele Funcionario:

Exercícios

9. Crie um método na classe Data que devolva o valor formatado da data, isto é, devolva uma String com "dia/mes/ano". Isso para que o método mostra da classe Funcionario possa ficar assim:

```
this.dataDeEntrada.formatada();
```

Bons Estudos

Namom Alves Alencar