O modelo de funcionários a seguir será utilizado para os exercícios de alguns dos posteriores capítulos.

O objetivo aqui é criar um sistema para gerenciar os funcionários do Banco. Os exercícios desse capítulo são extremamente importantes.

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).

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 ar- gumento. 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 funci- onário faz. Desenhe no papel tudo o que um Funcionario tem e tudo que ele faz.

1. Transforme o modelo acima em uma classe Java. 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.

Um esboço da classe:

class Funcionario {

double salario;

// seus outros atributos e métodos

void recebeAumento(double aumento) {

// o que fazer aqui dentro?

}

double calculaGanhoAnual() {

// o que fazer aqui dentro?

}

}

Tabela

Descrição gerada automaticamente com confiança médiaVocê pode (e deve) compilar seu arquivo java sem que você ainda tenha terminado sua classe Funcionario. Isso evitará que você receba dezenas de erros de compilação de uma vez só. Crie a classe Funcionario, coloque seus atributos e, antes de colocar qualquer método, compile o arquivo java. O ar- quivo Funcionario.class será gerado, mas não podemos “executá-lo” já que essa classe não tem um main. De qualquer forma, a vantagem é que assim verificamos que nossa classe Funcionario já está tomando forma e está escrita em sintaxe correta.

Esse é um processo incremental. Procure desenvolver assim seus exercícios, para não descobrir só no fim do caminho que algo estava muito errado.

Um esboço da classe que possui o main:

1 class TestaFuncionario {

2

3 public static void main(String[] args) {

4 Funcionario f1 = new Funcionario();

5

6 f1.nome = "Hugo";

y f1.salario = 100;

8 f1.recebeAumento(50);

9

1o System.out.println("salario atual:" + f1.salario);

11 System.out.println("ganho anual:" + f1.calculaGanhoAnual());

12 }

13 }

Incremente essa classe. Faça outros testes, imprima outros atributos e invoque os métodos que você criou a mais.

Lembre-se de seguir a convenção java, isso é importantíssimo. Isto é, preste atenção nas maiús- culas e minúsculas, seguindo o seguinte exemplo: nomeDeAtributo, nomeDeMetodo, nomeDeVariavel, NomeDeClasse, etc...

Todas as classes no mesmo arquivo

Você até pode colocar todas as classes no mesmo arquivo e apenas compilar esse arquivo. Ele vai gerar um .class para cada classe presente nele.

Porém, por uma questão de organização, é boa prática criar um arquivo .java para cada classe. Em capítulos posteriores, veremos também determinados casos nos quais você será **obrigado** a declarar cada classe em um arquivo separado.

Essa separação não é importante nesse momento do aprendizado, mas se quiser ir praticando sem ter que compilar classe por classe, você pode dizer para o javac compilar todos os arquivos java de uma vez:

javac \*.java

1. 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:

Funcionario f1 = new Funcionario();

// brincadeiras com f1....

f1.mostra();

Veremos mais a frente o método toString, que é uma solução muito mais elegante para mostrar a repre- sentação de um objeto como String, além de não jogar tudo pro System.out (só se você desejar).

O esqueleto do método ficaria assim:

class Funcionario {

// seus outros atributos e métodos void mostra() {

System.out.println("Nome: " + this.nome);

// imprimir aqui os outros atributos...

// também pode imprimir this.calculaGanhoAnual()

}

}

1. Construa dois funcionários com o new e compare-os com o ==. E se eles tiverem os mesmos atributos? Para isso você vai precisar criar outra referência:

Funcionario f1 = new Funcionario(); f1.nome = "Danilo";

f1.salario = 100;

Funcionario f2 = new Funcionario(); f2.nome = "Danilo";

f2.salario = 100;

if (f1 == f2) { System.out.println("iguais");

} else {

System.out.println("diferentes");

}

1. Crie duas referências para o **mesmo** funcionário, compare-os com o ==. Tire suas conclusões. Para criar duas referências pro mesmo funcionário:

Funcionario f1 = new Funcionario():

f1.nome = "Hugo"; f1.salario = 100;

Funcionario f2 = f1;

O que acontece com o if do exercício anterior?

1. (opcional) 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).

class Funcionario {

Data dataDeEntrada; // qual é o valor default aqui?

// seus outros atributos e métodos

}

class Data {

int dia; int mes; int ano;

}

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

Funcionario f1 = new Funcionario();

//...

Data data = new Data(); // ligação! f1.dataDeEntrada = data;

Faça o desenho do estado da memória quando criarmos um Funcionario.

y) (opcional) Modifique seu método mostra para que ele imprima o valor da dataDeEntrada daquele

Funcionario:

class Funcionario {

// seus outros atributos e métodos Data dataDeEntrada;

void mostra() {

System.out.println("Nome: " + this.nome);

// imprimir aqui os outros atributos...

System.out.println("Dia: " + this.dataDeEntrada.dia); System.out.println("Mês: " + this.dataDeEntrada.mes); System.out.println("Ano: " + this.dataDeEntrada.ano);

}

}

Teste-o. O que acontece se chamarmos o método mostra antes de atribuirmos uma data para este

Funcionario?

1. (opcional) O que acontece se você tentar acessar um atributo diretamente na classe? Como, por exemplo:

Funcionario.salario = 1234;

Esse código faz sentido? E este:

Funcionario.calculaGanhoAtual();

Faz sentido perguntar para o esquema do Funcionario seu valor anual?

1. (opcional-avançado) 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:

class Funcionario {

// atributos e metodos

void mostra() {

// imprime outros atributos...

System.out.println("Data de entrada: " + this.dataDeEntrada.formatada());

}

}