INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA BAHIA Campus Irecê	**Instituto Federal de Educação, Ciência e Tecnologia da Bahia – *Campus* Irecê** **Disciplina: Linguagem Técnica I** **Prof° Jonatas Bastos**

Nome: _____

LISTA DE EXERCÍCIO 2 - Exercícios de Orientação a Objetos

O objetivo aqui é criar um sistema para gerenciar os funcionários do Banco.

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), seu RG (String) e um valor booleano que indique se o funcionário ainda está ativo na empresa ou se já foi mandado embora. Você deve criar alguns métodos de acordo com sua necessidade. Além deles, crie um método bonifica que aumenta o salario do funcionário de acordo com o parâmetro passado como argumento. Crie, também, um método demite, que não recebe parâmetro algum, só modifica o valor booleano indicando que o funcionário não trabalha mais aqui. A idéia 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.

2) 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 chamada Funcionario, e a classe de teste você pode nomear como quiser. A de teste deve possuir o método main. Um esboço da classe:

```
class Funcionario {

    double salario;
    // seus outros atributos e métodos
    void bonifica(double aumento) {
       // o que fazer aqui dentro?
}      void demite() {
       // o que fazer aqui dentro?
} }
```

Você 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. Funcionario.class será gerado, não podemos "executá-la" pois não há um main, mas assim verificamos que nossa classe Funcionario já está tomando forma.

Um esboço da classe que possui o main:

```
class TestaFuncionario {

    public static void main(String[] args) {
        Funcionario f1 = new Funcionario();

        f1.nome = "Fiodor";
        f1.salario = 100;
        f1.bonifica(50);

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

    }
}
```

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, nomeDeAtributo, nomeDeMetodo, nomeDeVariavel, NomeDeClasse, etc...

3) 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 representação 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...
    }
}
```

. 4) 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 = "Fiodor";
  f1.salario = 100;
  Funcionario f2 = new Funcionario();
  f2.nome = "Fiodor";
  f2.salario = 100;
  if (f1 == f2) {
     System.out.println("iguais");
  } else {
     System.out.println("diferentes");
}
```

5) 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 = "Fiodor";
  f1.salario = 100;
  Funcionario f2 = f1;
```

O que acontece com o if do exercício anterior?

. 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).

```
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.

7) 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?

8) 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.demite();
```

Faz sentido pedir para o esquema do Funcionario demitir?

9) Um método pode chamar ele mesmo. Chamamos isso de **recursão.** Você pode resolver a série de fibonacci usando um método que chama ele mesmo. O objetivo é você criar uma classe, que possa ser usada da seguinte maneira:

```
Fibonacci fibo = new Fibonacci();

int i = fibo.calculaFibonacci(6);
System.out.println(i);
```

Aqui imprimirá 8, já que este é o sexto número da série. Este método calculaFibonacci não pode ter nenhum laço, só pode chamar ele mesmo como método. Pense nele como uma função, que usa a própria função para calcular o resultado.

10 Por que o modo acima é extremamente mais lento para calcular a série do que o modo iterativo (que se usa um laço)?

11) Escreva o método recursivo novamente, usando apenas uma linha. Para isso, pesquise sobre o **operador condicional ternário.** (ternary operator)

Exercícios para Fixação do Conhecimento.

O objetivo dos exercícios a seguir é fixar o conceito de classes e objetos, métodos e atributos. Dada a estrutura de uma classe, basta traduzí-la para a linguagem Java e fazer uso de um objeto da mesma em um programa simples.

12) Programa 1 Classe: Pessoa Atributos: nome, idade. Método: void fazAniversario(). Crie uma pessoa, coloque seu nome e idade iniciais, faça alguns aniversários (aumentando a idade) e imprima seu nome e sua idade.

13) Programa 2 Classe: Porta Atributos: aberta, cor, dimensaoX, dimensaoY, dimensaoZ Métodos: void abre(), void fecha(), void pinta(String s), boolean estaAberta() Crie uma porta, abra e feche a mesma, pinte-a de diversas cores, altere suas dimensões e use o método estaAberta para verificar se ela está aberta.

14) Programa 3 Classe: Casa Atributos: cor, porta1, porta2, porta3 Método: void pinta(String s), int quantasPortasEstaoAbertas() Crie uma casa e pinte-a. Crie três portas e coloque-as na casa; abra e feche as mesmas como desejar. Utilize o método quantasPortasEstaoAbertas para imprimir o número de portas abertas.