# LPOO – Lista de Exercícios

## Classes e Objetos

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 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.

   *Dica: use o software Violet UML Editor (violet.sourceforge.net) para fazer a modelagem*

2. Transforme o modelo acima em uma classe Java. Teste-a, usando uma outra classe que tenha *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 pelo menos o método *main*.

   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. O arquivo *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.

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();
   // ...
   f1.mostra();
   ```

4. Construa dois funcionários com *new* e compare-os utilizando o operador ==. Faça com que esses funcionários tenham os mesmos atributos. Eles foram considerados iguais? Explique o motivo.

5. Crie duas referências para o mesmo funcionário e compare-os utilizando o operador ==. Eles foram considerados iguais? Explique o motivo.

6. Ao invés 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. Em alguma classe que possua *Main*, crie um *Funcionario*. Faça o desenho do estado da memória quando um *Funcionario* é criado.

7. Escreva um modelo para representar uma *Lâmpada* que está à venda em um supermercado. Que dados devem ser representados por este modelo? Feita a modelagem da classe *Lâmpada* imagine uma lâmpada que possa ter três estados: apagada, acesa e meia-luz. Usando o modelo "Lâmpada" como base, escreva o modelo "LampadaTresEstados".

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

(a) `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.

(b) `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.

(c) `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.

9. Qual o propósito da palavra-chave *new*? Explique o que acontece quando você a utiliza.

10. O que é um construtor padrão? Como os atributos de um objeto são inicializados se uma classe tiver somente um construtor padrão?

11. Explique por que uma classe pode e deve fornecer um método *set* e um método *get* para atributos de uma classe.

12. Crie uma classe *GradeBook* com as seguintes características:

    - Atributos privados *courseName* (*String*) e *instructorName* (*String*);
    - Construtor que recebe apenas um *courseName* como parâmetro. O atributo *instructorName* deve receber "Not defined yet";
    - Construtor que recebe um *courseName* e um *instructorName* como parâmetro;
    - Métodos *setters* e *getters* para os atributos privados, com comportamento padrão;
    - Método *displayMessage*, que exibe uma mensagem de boas-vindas, apresentando o nome do curso e o nome do professor (se houver).

    Crie uma classe *GradeBookTest* que testa todos os atributos e métodos da classe *GradeBook*.

13. Crie uma classe *Account* com as seguintes características:

    - Atributo privado *balance* (*double*);
    - Construtor que recebe um *initialBalance* como parâmetro e o atribui para o atributo *balance*. O parâmetro *initialBalance* só deve ser atributo para *balance* se for maior que 0;
    - Método *credit*, que adiciona uma determinada quantia à conta. A quantia deve ser passada através de um argumento *amount* (*double*);
    - Método *getter* para a variável *balance*;

- Método *debit*, que retira dinheiro de uma *Account*. Assegure que a quantidade de débito não exceda o saldo de *Account*. Se exceder, o saldo deve ser deixado inalterado e o método deve imprimir uma mensagem que indica "Debit amount exceeded account balance".

Crie uma classe *AccountTest* que testa todos os atributos e métodos da classe *Account*.

14. Crie uma classe chamada *Invoice* para que uma loja de suprimentos de informática possa utilizá-la para representar uma fatura de um item vendido na loja. Uma *Invoice* deve incluir quatro informações como atributos: número (*String*), descrição (*String*), quantidade comprada de um item (*int*) e o preço por item (*double*). Sua classe deve ter um construtor que inicializa os quatro atributos. Todos os atributos são privados e você deve fornecer *setters* e *getters*. Forneça também um método *getInvoiceAmount*, que calcula o total da fatura (isto é, multiplica a quantidade pelo preço por item) e retorna um *double*. Se a quantidade comprada não for positiva, ela deve ser configurada como 0. Se o preço por item não for positivo, ele deve ser configurado como 0.0. Escreva um aplicativo teste chamado *InvoiceTest* que demonstra as capacidades da classe *Invoice*.

15. Crie uma classe chamada *Employee* que inclua três atributos: primeiro nome (*String*), sobrenome (*String*) e salário mensal (*double*). Forneça um construtor que inicializa os três atributos. Forneça um método *set* e um *get* para cada atributo. Se o salário mensal não for positivo, não configure seu valor. Escreva uma classe de teste chamada *EmployeeTest* que demonstra as capacidades da classe *Employee*. Crie dois objetos *Employee* e exiba o salário anual de cada objeto. Então dê a cada *Employee* um aumento de 10% e exiba novamente o salário anual de cada *Employee*.

16. Crie uma classe chamada *Date* que inclua três atributos: dia (*int*), mês (*int*) e ano (*int*). Forneça um construtor que inicializa os três atributos e suponha que os valores fornecidos estejam corretos. Forneça um método *set* e um *get* para cada variável de instância. Forneça um método *displayDate* que exibe o dia, o mês e o ano separados por barras normais (/). Escreva uma classe de teste chamada *DateTest* que demonstra as capacidades da classe *Date*.

17. Modifique a classe Date criada anteriormente, para que ela realize uma verificação de erros nos valores inicializadores dos atributos *month*, *day* e *year*. Forneça um método *nextDay* para incrementar o dia por um. O objeto *Date* deve permanecer em um estado consistente. Escreva um programa que testa o método *nextDay* em um loop que imprime a data durante cada iteração para ilustrar que esse método funciona corretamente. Teste os seguintes casos:

    - Incrementar para o próximo mês;
    - Incrementar para o próximo ano.

18. Ao realizar exercícios físicos, você pode utilizar um monitor de frequência cardíaca para ver se sua frequência permanece dentro de um intervalo seguro, sugerido pelos seus treinadores e médicos. Segundo a a *American Heart Association*, a fórmula para calcular a *frequência cardíaca máxima* por minuto é 220 menos a idade. Sua *frequência cardíaca alvo* é um intervalo entre 50%-85% da frequência cardíaca máxima. Crie uma classe chamada *HeartRates*. Os atributos da classe devem incluir o nome, sobrenome e data de nascimento da pessoa (utilize a classe *Date* criada anteriormente). Sua classe deve ter um construtor que recebe esses dados como parâmetros. Para cada atributo, forneça métodos *set* e *get*. A classe também deve incluir um método que calcula e retorna a idade da pessoa (em anos), um método que calcula e retorna a frequência cardíaca máxima da pessoa e um método que calcula e retorna a frequência cardíaca-alvo da pessoa. Escreva uma classe de teste *HeartRatesTest* que solicita as informações da pessoa, instancia um objeto da classe *HeartRates* e logo após imprime as seguintes informações a partir desse objeto: nome, sobrenome, data de nascimento, idade, intervalo de frequência cardíaca máxima e frequência cardíaca-alvo.

19. Escreva uma classe *HealthProfile* que representará o perfil da saúde de uma pessoa cadastrada em um sistema. Sua classe deve conter os atributos nome *String*, sobrenome *String*, sexo *char*, data de nascimento (do tipo *Date* criado anteriormente), altura (*float*, em metros) e peso (*float*, em quilogramas). Sua classe deve ter um construtor que recebe esses dados. Para cada atributo,

forneça métodos *set* e *get*. A classe também deve incluir métodos que calculem e retornem a idade do usuário em anos, intervalo de frequência cardíaca máxima, frequência cardíaca-alvo e índice de massa corporal (IMC). Escreva uma classe *HealthProfileTest* que solicite informações da pessoa, instancie um objeto da classe *HealthProfile* para essa pessoa e imprime as informações a partir desse objeto.

20. Crie uma classe *Rectangle* com atributos *length* e *width*, cada um dos quais assume o padrão de *1*. Forneça métodos que calculem o perímetro e a área do retângulo. A classe tem métodos *set* e *get* para o comprimento (*length*) e a largura (*width*). Os métodos *set* devem verificar se *length* e *width* são, cada um, números de ponto flutuante maiores que $0,0$ e menores que $20,0$. Escreva uma classe *RectangleTest* para testar a classe *Rectangle*.

21. Considere a classe *Time2* implementada abaixo:

```
public class Time2 {
    private int hour;
    private int minute;
    private int second;
    public Time2() {
        this(0, 0, 0);
    }
    public Time(int h) {
        this(h, 0, 0);
    }
    public Time2(int h, int m) {
        this(h, m, 0);
    }
    public Time2(int h, int m, int s) {
        setTime(h, m, s);
    }
    public Time2(Time2 time) {
        this(time.getHour(), time.getMinute(), time.getSecond());
    }
    public void setTime(int h, int m, int s) {
        setHour(h);
        setMinute(m);
        setSecond(s);
    }
    public void setHour(int h) {
        hour = ( ( h >= 0 && h < 24) ? h : 0);
    }
    public void setMinute(int m) {
        minute = ( (m >= 0 && m < 60) ? m : 0);
    }
    public void setSecond(int s) {
        second = ( (s >= 0 && s < 60) ? m : 0);
    }
    public int getHour() {
        return hour;
    }
    public int getMinute() {
        return minute;
    }
    public int getSecond() {
```

```
        return second;
    }
    public String toUniversalString() {
         return String.format("%02d:%02d:%02d",
         getHour(), getMinute(), getSecond());
    }
    public STring toString() {
        return String.format("%d:%02d:%02d %s",
        ( (getHour() == 0 || getHour() == 12) ? 12 : getHour() % 12 ),
        getMinute(), getSecond(), ( getHour < 12 ? "AM" : "PM"));
    }
}
```

Modifique a classe *Time2* para que a hora seja representada internamente como o número de segundos a partir da meia-noite em vez dos três valores inteiros *hour*, *minute* e *second*. Os clientes poderiam utilizar os mesmos métodos *public* e obter os mesmos resultados. Crie uma classe *Time2Test* e mostre que não há alteração visível para os clientes da classe.

22. Aprimore a classe Time apresentada em enunciados anteriores e faça com que ela inclua um método *tick*, que incrementa a hora armazenada em um objeto *Time2* e um segundo. forneça um método *incrementMinute* para incrementar o minuto um por um e o método *incrementHour* para incrementar a hora por uma. O objeto *Time2* sempre deve permanecer em um estado consistente. Escreva um programa que testa o método *tick*, o método *incrementMinute* e o método *incrementHour* para assegurar que eles funcionam corretamente. Certifique-se de testar os seguintes casos:

    - Incrementar para o próximo minuto;
    - Incrementar para a próxima hora;
    - Incrementar para o próximo dia (isto é, 11:59:59 PM para 12:00:00 AM).

23. Crie uma classe *SavingsAccount*. Utilize uma variável *static annualInterestRate* para armazenar a taxa de juros anual para todos os correntistas. Cada objeto da classe contém um atributo privado *savingsBalance* para indicar a quantidade que o cliente atualmente tem em depósito. Forneça método *calculateMonthlyInterest* para calcular os juros mensais multiplicando o *savingsBalance* por *annualInterestRate* dividido por 12 - esses juros devem ser adicionados ao *savingsBalance* Forneça um método *static modifyInterestRate* que configure *annualInterestRate* como um novo valor. Escreva um programa para testar a classe *SavingsAccount*. Instancie dois objetos *savingsAccount*, *saver1* e *saver2*, com saldos de R$ 2.000,00 e R$ 3.000,00, respectivamente. Configure *annualInterestRate* como 4% e então calcule o juro mensal de cada um dos 12 meses e imprima os novos saldos para os dois poupadores. Em seguida, configure *annualInterestRate* para 5%, calcule a taxa do próximo mês e imprima os novos saldos para os dois poupadores.

24. Crie uma classe chamada *Complex* para realizar aritmética com números complexos. Os números complexos têm a forma

    *parteReal* + *parteImaginária*i*

    onde *(I)* é $\sqrt{1}$.

    Escreva um programa para testar sua classe. Utilize variáveis de ponto flutuantes para representar os dados *private* da classe. forneça um construtor que permita que um objeto dessa classe seja inicializado quando ele for declarado. Forneça um construtor sem argumento com valores padrão caso nenhum inicializados seja fornecido. Formeça métodos *public* que realizam as seguintes operações:

    - Somar dois números *Complex*: as partes reais são somadas de um lado e as partes imaginárias são somadas de outro;

- Subtrair dois números *Complex*: a parte real do operando direito é subtraída da parte real do operando esquerdo e a parte imaginária do operando direito é subtraída da parte imaginária do operando esquerdo;
- Imprimir números *Complex* na forma $(a, b)$, onde $a$ é a parte real e $b$ é a parte imaginária.

25. Crie uma classe *DateAndTime* que combina as versões mais recentes das classes *Time* e *Date* criadas anteriormente. Modifique o método *incrementHour* para chamar o método *nextDay* se a hora for incrementada para o próximo dia. Modifique os métodos *toString* e *toUniversalString* para gerar saída da data além da hora. Escreva um programa para testar a nova classe *DateAndTime*. Especificamente, teste o incremento de tempo para o próximo dia.

26. Crie uma classe *Teclado* que contenha operações básicas de leitura do teclado. Decida quais serão as operações criadas, os modificadores de acesso dos métodos, se eles serão *static* e qual das classes Java você utilizará para leitura (*Scanner* ou *JOptionPane*). Crie uma classe *KeyboardTest* que teste as funcionalidades da classe criada.