Exercícios

Orientação a Objetos

Básico

1) Construir a classe *Horario* com os seguintes membros (atributos + métodos):
   Atributos: *hora, min*
   Métodos:
   - *getHora()*:
     não recebe valor
     retorna o atributo *hora*
   - *getMin()*:
     não recebe valor
     retorna o atributo *min*
   - *setHora()*:
     recebe o novo valor da hora
     retorna *true* se a hora for válida, false c. c.
   - *setMin()*:
     recebe o novo valor do minuto
     retorna *true* se o min for válido, false c. c.
   - *calcularIntervalo()*:
     recebe um horário h (objeto da classe *Horario*) como parâmetro
     calcula o intervalo de tempo deste *Horario* até o *Horario* h
     retorna o valor do intervalo em minutos (int)

   Construir a classe principal para testar a classe *Horario.* Esta classe:
   - pede ao usuário dois horários, de entrada e de saída
   - cria dois objetos da classe *Horario*
   - calcula o intervalo de tempo entre eles
   - calcula quanto custou:
     se intervalo menor do que 3 h -> R$ 4,50
     se intervalo entre 3 h e 12 h -> R$ 0,75 a cada 15 min excedente
     se intervalo maior do que 12 h -> R$ 33,00

2) Adaptar a classe *Horario* do exercício anterior para inicializar os valores de hora e minuto através de uma construtora. Qual(is) é(são) a(s) alternativa(s) para efetuar a validação dentro da construtora e retornar o status (sucesso ou erro)? Esta construtora poderia ser sobrecarregada de alguma forma?

3) Criar uma classe *Caixa*, cujo objetivo é representar uma caixa em um sistema de transporte de cargas.

   Atributos: largura, altura e profundidade
   Métodos:
   setLargura(), setAltura() e setProfundidade() – permitem ajustar os valores dos atributos (“setters”)

calcularAreaExt(), calcularVolume() – calculam e retornam os valores de área externa da caixa e volume, respectivamente.
Criar uma classe ExTestadorDeCaixa que:

- pergunte ao usuário as dimensões de 2 caixas
- crie 2 objetos da classe Caixa, preencha os atributos (utilizando os "setters") e chame os métodos para mostrar a área externa e o volume de cada uma

area = 2(largura*altura + largura*profundidade + altura*profundidade)*
*volume = largura*altura*profundidade*

4) Criar a classe *Pessoa* com as seguintes características:
   - atributos: idade e dia, mês e ano de nascimento, nome da pessoa
   - métodos:
     - *calculaIdade()*, que recebe como parâmetro a data atual em dia, mês e ano e calcula e armazena no atributo *idade* a idade atual da pessoa, sem retornar valor
     - *getIdade(),* que retorna o valor da idade
     - *getNome(),* que retorna o nome da pessoa
     - *setNome(),* que recebe o nome da pessoa como parâmetro e inicializa o atributo da classe
     - *setDataDeNascimento()*, que recebe dia, mês e ano de nascimento como parâmetros e preenche nos atributos correspondentes do objeto.
   - Fazer uma classe principal que crie dois objetos da classe *Pessoa*, um representando Albert Einstein (nascido em 14/3/1879) e o outro representando Isaac Newton (nascido em 4/1/1643). Em seguida, mostre quais seriam as idades de Einstein e Newton caso estivessem vivos.
   - Exemplo de classe de entrada:

```
public class ExPessoa
{
      public static void main (String [] args)
      {
            Pessoa p1, p2;

            p1 = new Pessoa();
            p2 = new Pessoa();

            p1.setNome("Isaac Newton");
            p1.setDataDeNascimento(4, 1, 1643);

            p2.setNome("Albert Einstein");
            p2.setDataDeNascimento(14, 3, 1879);

            p1.calculaIdade(30, 9, 2010);
            p2.calculaIdade(30, 9, 2010);

            int idadeP1 = p1.getIdade();
            int idadeP2 = p2.getIdade();

            System.out.println("Idade de p1: " + idadeP1);
            System.out.println("Idade de p2: " + idadeP2);
```

```
        }
    }
```

5) Alterar o programa do exercício anterior para substituir o método *setDataDeNascimento* e o método *setNome* por uma construtora

6) Implementar a classe *PolReg*, que define um polígono regular
   - Atributos: número de lados, tamanho do lado
   - Métodos: cálculo do perímetro, cálculo do ângulo interno e cálculo de área. Este último deve retornar o valor zero, dado que não é possível calcular a área de um polígono regular genérico
   - Construtora que recebe o número de lados e o tamanho dos lados como parâmetros e inicializa os valores dos atributos

Implementar uma classe principal para testar a classe *PolReg*.
- pedir ao usuário os dados de um polígono regular
- criar o objeto, lembrando de passar os valores para a construtora
- chamar os métodos e mostrar o que eles retornam

7) Implemente uma classe chamada *Carro* com as seguintes propriedades:

- Um veículo tem um certo consumo de combustível (medidos em km/litro), uma certa capacidade máxima de combustível e uma certa quantidade de combustível no tanque.
- O consumo e a capacidade máxima são passados como parâmetro para o construtor e o nível de combustível inicial é 0.
- Forneça um método *andar( )* que simule o ato de dirigir o veículo por uma certa distância, reduzindo o nível de combustível no tanque de gasolina.
- Forneça um método *getCombustivel( )*, que retorna o nível atual de combustível.
- Forneça um método *abastecer( )*, para abastecer o tanque.
- Escreva um pequeno programa que teste sua classe. Exemplo de uso:

```
Carro gol = new Carro(12, 45); // 12 quilômetros por litro
                               //de combustível,
                    //capacidade do tanque é 45 litros
gol.abastecer(20); // abastece com 20 litros de combustível.
Carro uno = new Carro(14,40);
uno.abastecer(25);
uno.andar(150); // anda 150 quilômetros.
int sobra = uno.getCombustivel() // Exibe o combustível que resta
no tanque.
System.out.println("Litros restantes no Uno: " + sobra);
gol.andar(80);
sobra = gol.getCombustivel();
System.out.println("Litros restantes no Gol: " + sobra);
```

8) Alterar o exercício anterior para que a classe *Carro* leve em consideração tanto o consumo urbano quanto o consumo em estrada (o método *andar* deve receber um parâmetro que indica se está andando na cidade ou na estrada). Em

seguida, implemente um programa que, utilizando esta classe, efetue as seguintes operações:

- receba do usuário uma quantidade de km e a quantidade de litros que o carro consumiu para andar aquela quantidade de km
- calcule e mostre quantos km ele andou na estrada e quantos ele andou na cidade.

9) Implementar a classe *Colaborador* com as seguintes características:

- atributos: nome, tempo de serviço na empresa em anos, tipo do vínculo (empregado, sócio ou estagiário), valor da hora de trabalho e número de horas que trabalha.
- métodos:
  - construtora que recebe o nome e o tipo do vínculo do colaborador e zera os demais atributos
  - métodos "setter" para cada um dos outros atributos
  - *calculaRendimentos()*, que calcula e retorna os rendimentos daquele colaborador. As regras para cálculo de rendimento são as seguintes:
    i. Estagiários recebem o valor da hora vezes o número padrão de horas trabalhadas (80 por mês)
    ii. Empregados recebem o número de horas trabalhadas vezes o valor da hora, sendo que este valor aumenta em 10% para cada ano de serviço na empresa. Caso o número de horas trabalhadas exceda 144, o empregado recebe 50% a mais por hora extra.
    iii. Sócios recebem o valor da hora vezes o número de horas trabalhadas.
  - *calculaCusto(),* que calcula e retorna quanto aquele colaborador custa mensalmente para a empresa. As regras para cálculo do custo são as seguintes:
    i. Estagiários e sócios não apresentam nenhum custo adicional além dos rendimentos.
    ii. Empregados custam o valor dos seus rendimentos mais 80% relativos a encargos (impostos, INSS, etc).
  - *getNome()*, que retorna o nome do colaborador.

Implementar um programa que:

- receba do usuário os dados de 3 colaboradores.
- calcule os rendimentos e custos de cada um e informe qual deles tem o maior rendimento e qual custa mais para a empresa.

10) Implementar a classe *Relogio* em Java*,* que pretende ser utilizada como modelo para a criação de um relógio com resolução de nanossegundos:

- atributos: ano, mês, dia, hora, minutos, segundos e nanossegundos da data/hora ao qual este objeto corresponde.
- métodos:
  - construtora para iniciar o objeto com os parâmetros fornecidos.
  - construtora para iniciar o objeto com a data/hora atuais (dica: utilizar *Calendar.getInstance()*). Como a classe *Calendar* possui resolução de milissegundos, o atributo de nanossegundos receberá um múltiplo de 10^6 (1 ms = 10^6 ns)

- método *getAgora()*, que retorna o valor da hora atual, com precisão de nanossegundos em relação ao valor original. Dica: utilizar *System.nanoSeconds()* para verificar o tempo que se passou desde a criação do relógio, em nanossegundos.

Implementar um programa para testar esta classe, obtendo os valores de tempo repetidamente e verificando se são valores coerentes.

11) Fazer uma classe *Vetor* em C++ que pretende representar a entidade geométrica vetor em um espaço bidimensional. Esta classe deve possuir as seguintes características:
- Atributos: *dx* e *dy*
- Operador + sobrecarregado, que recebe outro objeto da classe *Vetor* como parâmetro e retorna um terceiro vetor cujas dimensões são a soma das dimensões dos vetores anteriores.
- Operador * sobrecarregado, que recebe um inteiro como parâmetro e o multiplica pelas dimensões do vetor.
- Construtora que recebe as dimensões como parâmetro e inicializa os atributos do objeto.

Escrever um programa que crie dois vetores e exercite os seus métodos.

12) Repetir o exercício 11) em Java, substituindo os operadores sobrecarregados por métodos que executem as mesmas operações.

13) Fazer um programa orientado a objetos para simular a atração gravitacional entre dois corpos celestes:
- o programa instancia um frame, o qual possui uma área de desenho na qual é desenhada uma porção bidimensional do espaço com dimensões em escala planetária (por exemplo, 100e6 km de largura por 100e6 km de altura)
- dois corpos são posicionados aleatoriamente (ou por entrada do usuário) no espaço bidimensional. Os corpos possuem massas em escala planetária (por exemplo, 6e27 kg), também definida pelo usuário
- os corpos possuem velocidades nos eixos x e y, também em escala planetária (por exemplo, 30 km/s) e também definida pelo usuário
- ao iniciar-se o funcionamento do programa, os corpos devem ter sua interação gravitacional calculada (terceira lei de Newton), o que permite calcular a aceleração, variação da velocidade e variação da posição para um intervalo de tempo (também definível pelo usuário). Os corpos devem então ser redesenhados e o processo deve ser repetido, simulando a trajetória dos corpos sujeita à atração gravitacional.

Relacionamentos entre classes

14) Modelar em UML os relacionamentos existentes entre as classes básicas de um sistema de biblioteca: mídia, livro, DVD, revista, empréstimo, usuário, biblioteca, assunto, ... ?

15) Elaborar um programa em Java com as seguintes características:

- 1 classe *Cliente*, contendo:
  atributos: nome e CPF do cliente
  métodos: construtora para iniciar os atributos, *gerarRelatorio()* para retornar uma *String* com relatório do cliente e *aplicarRecursos()* que recebe um inteiro como parâmetro.

- 1 classe *Conta*, contendo:
  atributos: número e saldo
  métodos: construtora para iniciar o número, *sacar(), depositar(), getSaldo()* e *getNumero()*

- *Cliente* e *Conta* possuem relação de associação, com as seguintes características:
  - 1 *Cliente* pode ser titular de até 3 *Conta*
  - 1 *Conta* é de até 2 *Cliente*
  - associação é unidirecional de *Cliente* para *Conta*
  - atributo do link na classe *Cliente* se chama contas

16) Fazer um programa com as seguintes características:

- Uma classe chamada *Universidade* que terá como atributo um nome e terá um método para obter o seu nome (*getNome*).
- Relacionar a classe *Pessoa* para com a classe *Universidade*. Cada pessoa poderá ser associada a uma Universidade.
- A classe *Pessoa*, por sua vez, terá um método que imprimirá no console o seu nome, a sua idade caso estivesse vivo e em que universidade trabalhava.
- Fazer uma classe de teste para:
  - Criar dois objetos da classe *Pessoa*, um representando Albert Einstein (nascido em 14/3/1879) e o outro representando Isaac Newton (nascido em 4/1/1643).
  - Criar dois objetos de *Universidade*, associando um para Einstein e outro para Newton.
    - Einstein trabalhou como professor de física em Princeton (Nova Jersey - Estados Unidos da América).
    - Newton trabalhou como professor de matemática em Cambridge (Inglaterra).
  - Exercitar os métodos das classes e mostrar os resultados.

17) Fazer um programa para:
- Criar uma classe *Departamento* que permita relacionar um objeto (Departamento) à classe Universidade por composição (Universidade "contém" Departamento)
- Adaptar a classe *Pessoa* para que ela possua uma referência ao departamento que trabalha, ou seja, ela deve possuir uma associação com a classe *Departamento*, permitindo que cada objeto Pessoa tenha a referência de um objeto *Departamento*.
- A partir da classe *Pessoa* deve ser possível imprimir o seu nome, a sua idade (caso estivesse viva), o *Departamento* em que trabalhou e a *Universidade* à

qual este departamento pertence. Dica: navegar entre *Departamento* e a *Universidade* que o contém por meio do link da composição.

18) Fazer a classe *Assunto*:
- atributos: texto do assunto
- métodos: *setTexto*() para definir o texto do assunto e *getTexto*() para retornar o assunto

Fazer a classe *Mensagem*:
- atributos: texto da mensagem
- métodos: *addMensagem*() para adicionar ao texto da mensagem que já existe e *getMensagem*() para retornar o texto

Fazer uma classe *EMail* com as seguintes características:
- atributos: objetos das classes *Assunto* e *Mensagem* (por composição)
- métodos:
  - *setConteudoAssunto*() para definir o conteúdo do assunto
  - *addTextoMensagem*() para adicionar texto à mensagem.
  - *mostraDados*() que mostra no console os dados do e-mail no seguinte formato:

Assunto: xxxxx
Mensagem:
xxxxxxxxxxxxxxxxxxxx

Fazer uma classe principal que cria um *EMail*, preenche assunto e texto e depois chama *mostraDados*() para mostrar os dados do e-mail

19) Adaptar a classe *Pessoa*, implementada em exercício anterior, para substituir os atributos de dia, mês e ano de nascimento por um objeto da classe *Calendar* (ver http://java.sun.com/javase/6/docs/api para referência). Ou seja, existe uma relação de composição entre *Pessoa* e *Calendar.*
- Dicas:
  - *Calendar.getInstance()* deve ser chamado para obter um objeto novo da classe *Calendar* (este método é estático)
  - *set()* permite ajustar os valores de dia, mês e ano para este objeto
  - *get()* permite obter os valores para efetuar os cálculos de idade

20) Adaptar o enunciado de exercício anterior para:
- alterar a relação entre *Universidade* e *Departamento* para uma agregação, ou seja, uma *Universidade* pode inicialmente não ter *Departamento* porém eles podem ser criados posteriormente.
- Fazer com que uma *Universidade* possa ter 50 *Departamentos*.
- Fazer com que um *Departamento* referencie a *Universidade* a qual está filiada.
- Criar mais *Departamentos* filiando-os às *Universidades*.

21) Fazer um programa para:

- Criar a classe *Item*, que pretende representar um item sendo comprado em um site de e-commerce.
    - Atributos: nome do item, valor unitário e quantidade
    - Métodos:
        - construtora que recebe como parâmetro e ajusta os valores dos atributos
        - *getNome()*
        - *getValorTotal()*
- Criar a classe *CarrinhoDeCompras,* que permite criar uma lista de compras em um site de e-commerce.
    - Atributos: array de *Item.* Ou seja, existe uma relação de **agregação** entre *CarrinhoDeCompras* e *Item.*
    - Métodos:
        - *adicionarItem(Item i)*
        - *removerItem(Item i)*
        - *getValorTotal()*
- Criar a classe *Principal,* com o objetivo de criar itens, uma lista de compras e exercitar os métodos.

22) Fazer uma classe *Aluno* que possua as seguintes características:

- dois atributos do tipo inteiro: primeira nota parcial (de 0 a 100) e Segunda nota parcial (de 0 a 100)
- um atributo *String* representando o nome do aluno
- possua métodos para ler e escrever os atributos (ou uma construtora)

Fazer uma classe *Turma* que possua as seguintes características:

- atributos: coleção de *Aluno* (vetor ou objeto da classe *ArrayList*), número de vagas e nome da turma (disciplina e código).
- métodos: *matricularAluno(),* que recebe um *Aluno* como parâmetro e o matricula na *Turma* se houver vaga; *cancelarAluno(),* que recebe um *Aluno* como parâmetro e cancela sua matrícula; *gerarRelatorio(),* que gera uma String contendo: média da turma, quantos alunos foram aprovados, quantos foram para a final e quantos foram reprovados e os códigos de todos os alunos cujas notas ficaram abaixo da média da turma

Fazer uma classe *Controle* que:

- pergunte ao usuário o nome da turma, número de vagas e instancie um novo objeto da classe *Turma.*
- em laço, pergunte ao usuário se quer matricular um novo aluno, cancelar um aluno já matriculado, mostrar relatório ou sair.
    - caso queira matricular, pergunte ao usuário o nome e as duas notas parciais de um aluno. Caso o nome entrado seja “fim” isso significa que o usuário não quer inserir mais nenhum aluno, do contrário deve ser instanciado um objeto da classe *Aluno*, armazenados os dados digitados e matriculado na turma instanciada.

- caso queira cancelar, pergunte ao usuário o nome do aluno e cancela a sua matrícula na turma instanciada.
- caso queira mostrar relatório, chamar o método correspondente na turma instanciada e mostrar o relatório gerado.
- caso queira sair, encerrar o programa.

23) Fazer um sistema de calculadora simples, composto das seguintes classes:

CalcControle: controle da calculadora ("processador"), com os seguintes métodos:

- *public void executar()* – faz a calculadora funcionar por meio dos seguintes passos:
  - Recebe primeiro operando do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*
  - Recebe segundo operando do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*
  - Recebe operação do usuário através de *CalcInterface* e armazena no objeto de *CalcDados*. Se a operação for igual a 's', finaliza o programa (*System.exit(0)*).
  - Executa a operação (método *operar*) e mostra o resultado através de *CalcInterface.*
  - Armazena resultado como primeiro operando para a próxima operação e volta para o segundo passo
- *private double operar(double opn1, double opn2, char op)* - executa a operação desejada e retorna o resultado.

CalcDados: implementa a parte da calculadora que armazena os dados (operandos e operação) para o seu funcionamento ("memória"). Possui as seguintes características:

Atributos: dois números do tipo *double* para armazenar os operandos e um dado do tipo *char* para armazenar a operação.

Métodos:

- *public void setOperando(int i, double valor)* – armazena o i-ésimo operando com o valor expresso em *valor*
- *public double getOperando(int i)* – retorna o valor do i-ésimo operando
- *public void setOperacao(char op)* – armazena o caracter *op* como sendo a operação atual
- *public char getOperacao()* – retorna o valor da operação atual

CalcInterface: implementa a parte da calculadora que coleta e exibe informações ao usuário (display e teclado da calculadora). Possui os métodos:

- *public double recebeOperando(int i)* – recebe o operando i da operação (i vale 1 ou 2) e retorna.
- *public char recebeOperacao()* – recebe um char representando uma operação válida (+, -, * ou /) e retorna.
- *public void mostraResultado(double res)* – mostra o resultado recebido como parâmetro.

Criar a classe *Principal*, cujo único objetivo é instanciar os objetos de controle, dados e interface e criar os vínculos (associações) entre eles. Todas as classes

citadas devem possuir, além dos atributos citados, outros atributos que representem as referências para os outros objetos (criando as associações entre eles). Desenhar o diagrama de classes em UML para este sistema.

24) Adaptar o sistema de calculadora do exercício anterior para:
    - definir a classe *Calculadora,* que pretende representar a calculadora como um todo. Ou seja, a classe *Calculadora* possui uma relação de composição (agregação forte) com as demais classes já definidas (*CalcControle, CalcInterface* e *CalcDados*).
    - a classe *Calculadora* define o método *funcionar(),* o qual executa as seguintes operações:
        i. cria os componentes (objetos) e os vínculos de associação entre eles, conforme o exercício anterior
        ii. mostra uma mensagem de boas vindas na interface com o usuário.
        iii. reseta operandos e operações na memória.
        iv. chama o método *executar()* do objeto da classe *CalcControle.*

25) Desenvolva um sistema no qual existam a classe *Familia* e a classe *Pessoa.* Em seguida, modele e implemente a relação entre *Familia* e *Pessoa* na forma de uma agregação.
    Sugestões de atributos: principal, cônjuge e filhos para *Familia,* pais para *Pessoa*, sobrenome para *Familia,* nome para *Pessoa.*
    Sugestões de métodos: *listarArvoreFamiliar()* para *Familia, getNomeCompleto()* para *Pessoa,* métodos para adicionar filhos à *Familia,* métodos para adicionar filhos à *Pessoa.*

26) Crie uma classe que representa um ponto no plano cartesiano. Em seguida, crie uma classe que representa um triângulo, reusando a classe anterior por composição. Finalmente, escreva um programa que receba do usuário as coordenadas dos vértices do triângulo e imprima seu perímetro.
    Fonte:http://74.125.47.132/search?q=cache:uMJJLyJh1CwJ:www.disi.unitn.it/~vitorsouza/sites/default/files/JavaSwingBDExerciciosRevisao.pdf+exerc%C3%ADcios+java&cd=20&hl=pt-BR&ct=clnk&client=opera

27) Altere o exercício anterior para que a relação entre figura geométrica e ponto cartesiano seja de agregação. Ou seja, o formato da figura pode ser alterado, mediante adição ou remoção de pontos, bem como os pontos podem ser compartilhados por figuras geométricas adjacentes (interseção de vértices ou arestas).

Herança e polimorfismo

28) Implemente a classe *CarroEsporte*, derivada da classe *Carro* do exercício 7), com as seguintes características:

- atributos: tamanho do spoiler, número de adesivos do *tuning*, potência extra do turbo
- métodos:
    - redefinição de *andar(),* que gasta 50% a mais de combustível do que na classe base
    - *acelerar(),* que simula o ronco do motor (e gasta 1 litro por acelerada)

Fonte:http://www.bernhard.pro.br/disciplinas/java/ensino/java-L01.pdf

29) Implementar a classe *PolReg*, que define um polígono regular
- Atributos: número de lados, tamanho do lado
- Métodos: cálculo do perímetro, cálculo do ângulo interno e cálculo de área. Este último deve retornar o valor zero, dado que não é possível calcular a área de um polígono regular genérico
- Construtora que inicializa os valores dos atributos

Fazer a classe *TrianguloEq,* derivada de *PolReg,* que:
- redefine o método de cálculo de área para calcular e retornar a área do triângulo equilátero

Fazer a classe *Quadrado*, também derivada de *PolReg,* que:
- redefine o método de cálculo de área para calcular e retornar a área do quadrado

Fazer um programa que crie objetos das classes *PolReg, TrianguloEq* e *Quadrado* e exercite os seus métodos.

30) Aproveitar as classes do exercício anterior e adaptar a classe de entrada para:

- declarar uma referência *p* para PolReg
- perguntar ao usuário que tipo de objeto ele quer criar (polígono regular, triângulo equilátero ou quadrado)
- perguntar número de lados (se for polígono regular) e tamanho dos lados
- instanciar o objeto e armazenar no atributo *p*.
- no final do programa, usar a referência *p* para mostrar o valor do perímetro, ângulo interno e área calculados

31) Escrever a classe *Complexo*, que pretende representar um modelo para a criação de números complexos. A classe *Complexo* deve possuir:
- 2 atributos double: *real* e *imag*
- 1 construtora que recebe a parte real e a parte imaginária do número e inicializa os atributos
- 1 método *modulo*() que retorna o módulo do número complexo.
- 1 método *angulo*() que retorna o ângulo do número complexo em graus.

Escrever a classe *Real*, derivada da classe *Complexo,* que pretende representar um modelo para a criação de números reais. A classe *Real* deve:
- possuir uma construtora que recebe o valor do número como parâmetro e chama a construtora da base.
- adicionar o método *sinal*(), que retorna 1 se o número for positivo e -1 se for negativo.

Escrever um programa em Java para testar estas classes (instanciar objetos, chamar métodos, etc.).

32) Escrever a classe *Curso,* que pretende representar um modelo para a criação de um curso em uma instituição de ensino. A classe *Curso* deve possuir:
   - atributos: carga horária mínima, nome.
   - métodos: construtora para setar os atributos; *cumpriuCriterios(),* que recebe a carga horária já cumprida pelo aluno e retorna *true* se cumpriu, *false* caso contrário; *getNome().*

   Escrever a classe *CursoMestrado,* derivada de *Curso,* com as seguintes características:
   - atributos: nota da defesa de dissertação.
   - métodos: construtora para setar os atributos; *cumpriuCriterios(),* que recebe a carga horária já cumprida e a nota da dissertação e retorna *true* se cumpriu, *false* caso contrário; *getNome().*

   Escrever classe de entrada para testar as classes e métodos. O método *cumpriuCriterios()* é polimórfico?

33) Fazer uma classe *Pessoa*:
- Atributos privados:
  - Nome completo
  - Gênero (masculino ou feminino)
  - Dia, mês e ano de nascimento
- Métodos públicos
  - *Pessoa(nome, gênero, dia, mês, ano)*
    Por default, gênero='m', dia=1, mês=1, ano=1970
  - *obter_descricao()* retorna
    "<nome>, [homem|mulher], nasceu em dd/mm/aa"
  - Fazer um setter para dia, mês e ano que faça consistência dos dias em função dos meses
  - Fazer um setter para gênero que só aceita 'm' ou 'f', caso contrário não efetua o set.

Fazer duas especializações da classe *Pessoa*:
- *Aluno*
  - Atributos: curso e universidade
  - Ano de ingresso
  - Sobrescrever o método *obter_descricao()*:
    "<nome>, [homem|mulher], nasceu em dd/mm/aa, cursa <curso> desde <ano de ingresso>"
- *Professor*
  - Universidade
  - Ano de contratação
  - Sobrescrever o método *obter_descricao()*:

"<nome>, [homem|mulher], nasceu em dd/mm/aa, professor da <universidade> desde <ano de ingresso>".

No programa principal:

- Instanciar uma mulher com data de nascimento.
- Instanciar um homem com valores padrão.
- Imprimir as descrições na tela.

34) Escrever a classe *Empregado*, que pretende representar um empregado em um sistema de RH.

- Atributos: nome (String), CPF (int) e cargo (String)
- Construtora que recebe nome, CPF e cargo como parâmetros e preenche os atributos
- Método *mostraDados*(), que mostra os dados do empregado no formato: Nome: xxx, CPF: xxx, Cargo: xxx
- Método *calculaSalario*(), que recebe o número de horas trabalhadas como parâmetro e retorna 415,00.

Escrever a classe *Faxineiro*, derivada de *Empregado:*

- construtora que recebe nome e CPF e repassar os valores pra base
- método *mostraDados*(), que mostra os dados no formato:
  Nome do faxineiro: xxx, CPF: xxx
- método *calculaSalario*(), que recebe o número de horas trabalhadas e retorna: até 176 horas – 4,50/hora, hora extra – 6,00/hora

Escrever a classe *Gerente*, derivada de *Empregado*

- Atributos: bonus - int.
- construtora que recebe nome, CPF e bonus como parâmetros
- método *mostraDados*(), que mostra os dados no formato:
- *Nome do gerente: xxx, CPF: xxx, bonus: xxx*
- método *calculaSalario*(), que recebe o número de horas trabalhadas e retorna: até 176 horas – 30,00/hora + bonus, hora extra – não paga

Escrever uma classe de entrada para:

- Perguntar ao usuário se quer inserir os dados de um *Faxineiro*, *Empregado* ou *Gerente.*
- Instanciar o objeto correspondente
- Inserir o número de horas trabalhadas e mostrar os dados da pessoa, bem como o salário para aquele mês.

35) Adaptar o exercício anterior para:

- Perguntar ao usuário quantos empregados ele deseja cadastrar
- Declarar um vetor de referências para *Empregado*
- Perguntar se quer inserir os dados de um faxineiro ou de um gerente e instanciar o objeto correspondente

- Varrer o vetor de empregados e calcular a soma dos salários de todos os empregados da empresa

36) Implementar um sistema de controle de uma conta bancária:

- Classe *Conta*: contém os seguintes membros:
  - Atributos: *saldo* (double) e *número* (int)
  - Métodos:
    - *depositar()*, que recebe um valor como parâmetro e adiciona ao saldo da conta
    - *getSaldo()*
    - *sacar()*, que recebe um valor como parâmetro, subtrai do saldo da conta se houver saldo suficiente e retorna *true.* Caso o saldo não seja suficiente, retorna *false.*
    - *aplicarJurosDiarios()* (abstrato), que recebe um número de dias como parâmetro e atualiza o saldo por meio da aplicação de juros diários na quantidade de dias informada.
- Classe *ContaCorrente*, derivada de *Conta*:
  - Implementa *aplicarJurosDiarios()* de 0,1% sobre o que exceder R$ 1.000,00.
- Classe *ContaPoupança*, derivada de *Conta*:
  - Implementa *aplicarjurosDiarios()* de 0,2%.
- Aplicação:
  - Menu com opções para: criar conta, ler saldo, depositar, sacar e atualizar saldo em função de dias de aplicação

37) Escrever a classe *Pessoa* com as seguintes características:

- atributos: *nome* e *idade*
- métodos: construtora para inicializar os parâmetros e *mostraDados()* que exibe os dados da pessoa no console na forma:

Nome da pessoa: xxx
Idade da pessoa: yyy

Escrever a classe *Aluno*, derivada de *Pessoa,* com as seguintes características:

- atributos: nome do curso que está cursando
- métodos: construtora para inicializar os atributos e redefinição do método *mostraDados()* para exibir as seguintes mensagens:

Nome do aluno: xxx
Idade do aluno: yyy
Curso do aluno: zzz

Elaborar um programa em Java que:

- declare uma referência para objeto da classe *Pessoa*
- pergunte ao usuário, via console, se ele deseja instanciar um aluno ou uma pessoa
- crie o objeto correspondente, referencie com a referência já criada e chame o método *mostraDados()* para exibir os dados da pessoa ou do aluno

38) Alterar a classe *Pessoa* para incluir um atributo estático que permita informar quantos objetos de *Pessoa* (ou derivadas desta) existem no sistema.

39) Alterar o exercício sobre polígonos regulares, triângulos e quadrados para que a classe *PolReg* não apresente uma definição para o método de cálculo de área; ou seja, a classe *PolReg* deve ser transformada em uma classe abstrata

40) Crie uma classe para representar uma conta-corrente, com métodos para depositar uma quantia, sacar uma quantia e obter o saldo. Para cada saque será debitada também uma taxa de operação equivalente à 0,5% do valor sacado. Crie, em seguida, uma classe derivadas desta classe anterior para representar uma conta-corrente de um cliente especial. Clientes especiais pagam taxas de operação de apenas 0,1% do valor sacado. Faça testes com as duas classes e verifique seus resultados.
Fonte:http://74.125.47.132/search?q=cache:uMJJLyJh1CwJ:www.disi.unitn.it/~vitorsouza/sites/default/files/JavaSwingBDExerciciosRevisao.pdf+exerc%C3%ADcios+java&cd=20&hl=pt-BR&ct=clnk&client=opera

41) Fazer um programa orientado a objeto com o seguinte enunciado:
    - Implementar a interface ou classe abstrata *Classificavel,* que define o método boolean *maiorQue(Classificavel outro)*
    - Adaptar as classes *PolReg, TriEq* e *Quad* para implementar a interface/classe abstrata *Classificavel*
    - Adicionar o atributo "cor" à classe *PolReg*
    - Implementar uma classe que:
        - Pergunte ao usuário número de lados e tamanho do lado de 5 polígonos.
        - Adicione os polígonos a um array.
        - Chame o método estático void *classifica(Classificavel [] conj)* da classe principal para classificar o conjunto por área crescente.
        - Mostrar os polígonos classificados (cor, número de lados e área).

42) Implementar uma hierarquia de classes para a definição e o cálculo de expressões numéricas. Algumas classes que podem (ou devem) constar desta hierarquia: operando, operador, soma, mult, div, expressão.

43) Uma locadora de veículos aluga diversos tipos de veículos: caminhões, carros de passeio categoria A, B e C, utilitários (furgões e camionetes) e motos. O preço da diária dos veículos varia em função do imposto de locação e do valor de locação (ambos diários). A fórmula de cálculo do aluguel é diferente para caminhões, carros de passeio, utilitários e motos. Caminhões levam em conta o número de eixos e motos, a cilindrada (caso não seja informado, assume-se 125cc). Organize as classes hierarquicamente em um diagrama de classes com seus atributos e o método para calcular o valor da locação diária. Em seguida, implemente este exemplo em uma linguagem que suporte OO e efetue testes.

44) Um usuário deseja ter vários tipos de assinaturas para colocar nos seus e-mails de acordo com o destinatário (há também uma assinatura default para outros não especificados). Como você utilizaria uma classe abstrata, herança e polimorfismo para resolver este problema? A assinatura é composta por três atributos: mensagem, nome, ramal. Deve haver um método *imprimirAssinatura*. Implemente e faça o diagrama e classes em UML.

Gabaritos

45) Implementar o gabarito para a classe *Pilha.* Ele deve conter:
- construtora para definir o tamanho máximo da pilha
- métodos *push*() e *pop*() para empilhar e desempilhar elementos, respectivamente. O tipo de cada elemento é determinado pelo parâmetro formal de tipo fornecido na criação do objeto.
- Escrever um programa para:
    i. criar uma pilha de inteiros e uma pilha de caracteres
    ii. efetuar operações de inserção e remoção, mostrando o resultado.

46) Implementar uma classe de gabarito / genérico para efetuar a ordenação por meio do método *Bubble-sort.* Esta classe deve:
- definir um atributo genérico que corresponde ao vetor de elementos, de tipo genérico, a serem ordenados
- definir um método *classifica()* que efetua a ordenação. Para comparar elementos adjacentes e decidir sobre a sua ordem relativa, este método deve chamar o método *maiorQue()* da classe dos elementos sendo ordenados.

O programa deve rodar com o seguinte código (caso seja feito em Java):

```
public class ExBubble
{
	public static void main (String [] args)
	{
		AlgoOrdenavel [] aov = new AlgoOrdenavel[3];

		aov[0] = new AlgoOrdenavel(5);
		aov[1] = new AlgoOrdenavel(3);
		aov[2] = new AlgoOrdenavel(-2);

		Bubble<AlgoOrdenavel> b = new Bubble<AlgoOrdenavel>(aov);

		b.classifica();

		for(int i = 0; i < aov.length; i++)
		{
			System.out.println("Valor = " + aov[i].getValor());
		}
	}
}

class AlgoOrdenavel
{
	private int valor;
	public AlgoOrdenavel(int v) { valor = v; }
	public boolean maiorQue(AlgoOrdenavel ao)
	{
		if (valor > ao.valor) return true;
```

```
            return false;
        }
        public int getValor() { return valor; }
}
```

UML

47) Elaborar um diagrama de classes UML que represente o seguinte conteúdo:
    - 1 classe *Cliente*, contendo:
        - atributos: nome e CPF do cliente
        - métodos: construtora para iniciar os atributos, *gerarRelatorio()* para retornar uma String com relatório do cliente e *aplicarRecursos()* que recebe um inteiro como parâmetro.
    - 1 classe *Conta*, contendo:
        - atributos: número e saldo
        - métodos: construtora para iniciar o número, *sacar(), depositar(), getSaldo()* e *getNumero()*
    - *Cliente* e *Conta* possuem relação de associação, com as seguintes características:
        - 1 *Cliente* pode ser titular de até 3 *Conta*
        - 1 *Conta* é de até 2 *Cliente*
        - associação é unidirecional de *Cliente* para *Conta*
        - atributo do link na classe *Cliente* se chama *contas*

48) Considerando que se deseja modelar em UML um sistema de software utilizado para controlar o estoque e os pontos de venda (“caixas”) em um supermercado, efetuar as seguintes tarefas:
    - definição das classes envolvidas, para pelo menos 2 casos de uso, e classificação por estereótipo
    - levantamento dos relacionamentos entre as classes
    - elaboração do(s) diagrama(s) de classes correspondente(s).

49) Idem ao exercício anterior, considerando um sistema de gerenciamento de consultas em uma clínica médica

50) Elaborar o diagrama de classes para o programa de processamento de expressões numéricas (exercício 42).

Design patterns

51) Implemente um programa que pretende obter as duas notas parciais de um aluno e calcular a sua média e sua situação (aprovado, em exame ou reprovado). O programa tem as seguintes características:

- o programa pergunta ao usuário, na inicialização, qual a fonte dos dados que vai utilizar: arquivo de texto ou entrada padrão (console).

- dependendo da resposta, o programa instancia um DAO correspondente que permite acessar os dados. Tanto o DAO da entrada padrão quanto o DAO do arquivo de texto são derivados da classe *DAOAbstrato*, a qual implementa os métodos *lerNota1()* e *lerNota2()* (ambos retornam o valor da nota correspondente).
- o programa obtém as notas a partir da chamada dos métodos e mostra o resultado.

Dica: para o DAO de arquivo, considerar que o método *lerNota1()* lê o conteúdo completo do arquivo (dois valores de 0 a 100, separados por espaço) e armazena internamente para poder retornar a segunda nota quando o método *lerNota2()* for chamado.

52) Adaptar uma classe projetada segundo o padrão Singleton para permitir que dois objetos sejam criados ao invés de um. O método *getObjeto(i)* deve receber o índice *i* (0 ou 1) do objeto a ser retornado.