UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE CIÊNCIA E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Lista de Exercícios de Linguagens de Programação – Prog. Orientada a Objetos

1) Implemente uma agenda telefônica usando um vetor ou lista de objetos. Cada contato da agenda deve ser representado como uma classe, a qual conterá informações como nome, telefone, endereço e relação (nome de empresa ou nome de pessoa física que se refere este contato, uma espécie de lembrete). A agenda deve ser implementada como uma outra classe, a qual conterá o vetor ou lista de objetos (contatos). Caso seja um vetor, uma agenda pode armazenar até 1000 contatos (Como sugestão, pode ser criado um campo último, o qual guarda a última posição inserida no vetor e evita que este seja percorrido por completo em cada operação). Deve ser criada uma terceira classe, chamada Principal, a qual utilizará as classes criadas anteriormente. Além disso faça:

(a) Implemente um método de busca na agenda, o qual recebe um nome, ou parte deste, e retorna o contato. Caso haja mais de um contato, a busca deve retornar apenas o primeiro.
(b) Implemente métodos de inserção, alteração e remoção de contatos da agenda. Caso o nome de um contato sendo inserido já exista na agenda, a inserção se torna uma alteração.
(c) Implemente um método para listar os contatos de uma agenda. Para auxiliar tal listagem, crie um método toString(), na classe contato, o qual retorna uma string (qualquer formato) com o conteúdo de um contato.
(d) Implemente métodos para salvar/recuperar os contatos de arquivos, de forma que estes possam durar a após as execuções do programa.
(e) Insira o seguintes contatos na agenda a ser criada na classe Principal:

Nome	Telefone	Endereço	Relação
Fulano	99999999	Rua A	UFF
Ciclano	88888888	Rua B	Cederj
Beltrano	88889999	Rua C	Infância

(f) Ainda na classe Principal, chame o método de inserção novamente para o contato Fulano, 77777777, Rua D; remova o contato Ciclano e liste o conteúdo da agenda.

2) Na Matemática, um intervalo é um conjunto de números reais com a propriedade que qualquer número pertencente entre 2 números num conjunto também pertence a este conjunto. Por exemplo, o conjunto de todos os números que satisfaçam [0, 1] é um intervalo que contém os valores 0, 1 e todos que estão entre estes 2 números. Intervalos são elementos fundamentais na matemática intervalar, uma técnica de computação numérica que garante resultados, mesmo na presença de incertezas e/ou aproximações.

Crie um tipo de dados (classe) para modelar um intervalo de números na reta dos inteiros. Por exemplo, [-3, 7) representa o intervalo que compreende os valores de -3 a 7, incluindo o -3 e excluindo o 7. Defina as seguintes operações (métodos) sobre os intervalos:

(a) *a.contém(v)* retorna verdadeiro se o valor *v* pertence ao intervalo *a*; caso contrário, retorna falso
(b) *a.intercepta(b)* retorna verdadeiro se há interseção entre os intervalos *a* e *b*; caso contrário, a operação retorna falso
(c) *a.media()* retorna a média dos valores pertencentes ao intervalo
(d) na aritmética de intervalos temos a combinação dos seus limites; implemente o método *a.produto(b)* que retorna um novo intervalo *c* com os seguintes limites: [min (infa*infb, infa*supb, supa*infb, supa*supb), max (infa*infb, infa*supb, supa*infb, supa*supb)], onde os prefixos *inf* indica o limite inferior do intervalo e *sup* o superior.
(e) *a.uniao(b)* deve representar todos os valores que fazem parte dos intervalos *a* e *b*; se podemos ter intervalos sem interseção, como podemos representar esta união? Adeque a solução apresentada para trabalhar com estes novos tipos de intervalos.

3) Suponha o código abaixo que manipula um objeto que representa um carrinho de compras.

```
1. public class Principal {
2.     public static void main(String[] args) {
3.             EstoqueProdutos estoque = new EstoqueProdutos();
4.             estoque.adicionaProduto(new ProdutoEstoque ("monitor", 500, 100));
5.             estoque.adicionaProduto(new ProdutoEstoque ("telefone", 150, 300));
6.             estoque.adicionaProduto(new ProdutoEstoque ("teclado", 70, 50));
7.             estoque.adicionaProduto(new ProdutoEstoque ("mouse", 50, 50));
8.
9.             CarrinhoCompra carrinho = new CarrinhoCompra(estoque);
10.            carrinho.adicionaItem("monitor", 2);
11.            carrinho.adicionaItem("telefone", 5);
12.            carrinho.adicionaItem("teclado", 2);
13.
14.            carrinho.finalizaCompra();
15.
16.            System.out.println("A soma dos produtos   : " + carrinho.calculaTotal());
17.    }
18.}
```

Para o programa acima, realize as seguintes inserções (utilize os conceitos de OO vistos sempre que possível):

(a) Defina a classe *ProdutoEstoque*, cujo objetos são criados das linhas 4 a 7. Um produto, representado por esta classe, possui um nome, um valor e a quantidade deste produto em estoque.
(b) Crie a classe *EstoqueProdutos*, cujo objeto é criado na linha 3. Esta classe deve permitir que produtos possam ser armazenados, como indica o método *adicionaProduto*, chamado das linhas 4 a 7.
(c) Crie a classe CarrinhoCompra, a qual permitirá a adição de produtos à partir de um estoque. Esta classe precisará implementar, pelo menos, 3 métodos: adicionaItem(), (linhas 10 a 12), que adiciona um item ao carrinho; finalizaCompra(), (linha 13), que debita efetivamente a quantidade do produto no estoque da quantidade comprada, e; calculaTotal(), (linha 15), que dá o cômputo total dos produtos (soma dos valores * a quantidade destes).

4) Suponha que necessitamos implementar um sistema que precisa manipular datas. Faça um programa em Java a partir dos seguintes passos para este fim:

a) Crie uma classe chamada MinhaData, a qual deverá conter 3 campos inteiros que representam o dia, mês e ano desta data.
b) Defina um construtor que receba 3 valores e inicialize os 3 campos de um objeto.
c) Defina um 2o. construtor que inicialize um objeto a partir de uma string contendo uma data (p. ex., “1/4/2013”).
d) Crie um método toString() para esta classe que retorna uma string representando o objeto data.
e) Crie métodos para adicionar, ou diminuir, dias, meses e anos de uma data.
f) Crie um método chamado compara, que compara a data representada pelo objeto alvo da chamada com uma data passada como argumento para o método; o valor retornado deve ser 0 se essas datas são iguais, -1 se a primeira data é anterior à última, ou +1 se a primeira é posterior à última.
g) Crie uma segunda classe, chamada DataComemorativa, a qual representará as diferentes datas comemorativas. Uma data comemorativa normalmente contém um nome, se é feriado ou não, se este feriado é mundial e o dia associado.
h) Crie uma terceira classe chamada DatasComemorativas, a qual deverá conter uma coleção que armazenará todas as datas comemorativas existentes.
i) Implemente nesta terceira classe o método adiciona(), que insere uma data comemorativa na lista.
j) Implemente nesta mesma classe o método remove(nome), que remove da lista a data comemorativa que possui o parâmetro nome fornecido.
k) Implemente um método chamado horasNaoTrabalhadas(), o qual deve retornar a quantidade de horas não trabalhadas. Para tal, deve-se contar a quantidade de datas comemorativas que são feriados e multiplicá-la por 8 (oito) que é a carga horária diária usual de trabalho.
l) Teste as classes criadas da seguinte forma: i) No método main(), crie 1 data que represente a data atual e outra que represente o Natal deste ano; ii) Chame o método de comparação das datas e imprima seu valor; iii) Adicione o objeto Natal àc coleção DatasComemorativas e chame o método horasNaoTrabalhadas().

5) Suponha que você possua uma biblioteca particular e queira criar um sistema para controlar os empréstimos destes livros a seus amigos.

a) Crie uma classe chamada Emprestimo, a qual deve conter as datas de retirada, devolução e devolução efetiva, caso seu amigo entregue com atraso. Além disso, seu amigo deve deixar um e-mail para contato (na verdade, cobrança). Além desses dados, informações básicas do livro, como título, autor, editora e ano devem ser armazenados. Obs.: As datas devem ser criadas como objetos da classe java.time.LocalDate.

Após iniciar esse sistema, sua mãe, ou algum parente próximo, reclama que empresta itens, como vasilhames, canetas, e estas nunca são devolvidas. Na intenção de resolver o problema desse seu parente, devemos modificar a classe anterior para não registrar apenas livros.

b) Crie uma classe específica para manipular livros.
c) Crie outra classe, chamada Utensílios, para armazenar itens emprestados por seu parente, como materiais de cozinha, canetas, etc. Esta classe deve conter um campo chamado descrição e outro que informa o material que constitui o item (alumínio, metal, madeira, plástico, etc) para que o item seja mais precisamente descrito.
d) Altere a classe Emprestimo, substituindo os campos referentes a livros por um atributo geral que represente tanto um livro quanto os utensílios.

Para verificarmos se as classes estão funcionando corretamente, crie um método main() que faça as seguintes operações:

e) Crie um objeto para o livro “O Monge e o Executivo”, do autor James Hunter, da editora Sextante. Este livro foi publicado em 2004.
f) Crie um objeto que represente uma frigideira de teflon.
g) Crie uma lista de empréstimos, a qual lista os produtos que foram emprestados.
h) Adicione o livro e a frigideira à lista de empréstimos. O livro foi emprestado no dia 3/1/2014, enquanto que a frigideira foi emprestada no dia 23/12/2013. A data de entrega do livro é 3/2/2014, enquanto que a frigideira é dia 27/12/2013. Ambos ainda não foram entregues.
i) Ainda no método main(), percorra a lista de empréstimos e imprima o nome dos produtos que estão atrasados. A data corrente pode ser obtida chamando o método estático java.time.LocalDate.now().

Utilize arquivos (texto ou binários) para salvar os dados momentâneos de empréstimo e permitir que estes sejam recuperados em futuras execuções do programa.

6) Considere que fomos contratados para implementar um sistema simples de agendamento de pacientes. O código abaixo inicia a implementação desse sistema:

```
enum PLANO {
	AMIL, UNIMED, AMS, ASSIM, GOLDEN, OUTRO, PARTICULAR
}

class Contato {
	String nome;
	PLANO plano;
}

class Clientes {
	Map<Contato, java.time.LocalDate> clientes;

	public Clientes() {
		this.clientes = new HashMap<Contato, java.time.LocalDate>();
	}

	public Clientes(Map<Contato, java.time.LocalDate> clientes) {
		this.clientes = clientes;
	}
}
```

O tipo enum lista os planos permitidos no sistema. Para exemplificar o seu uso, referirmo-nos ao plano da ASSIM escrevendo “PLANO.ASSIM”. A classe Contato representa o paciente no sistema. A classe Clientes representa os pacientes do sistema. Observe que a coleção declara nesta classe é um mapa (java.util.Map), uma vez que queremos armazenar o contato associado com a hora em que este foi agendado. Logo abaixo a declaração desta coleção temos 2 construtores, os quais inicializam a coleção com um mapa vazio (primeiro construtor), ou com um mapa passado por parâmetro (segundo construtor).

a) Insira campos na classe Contato que permitam armazenar o telefone e e-mail do paciente.
b) Declare um construtor que inicialize os campos da classe.
c) Suponha que os clientes possam ser pessoas jurídicas (empresas). Crie uma classe ContatoEmpresa, a qual terá todos os campos de Contato mais o nome da empresa e seu CNPJ. Declare o construtor desta classe para inicialização de seus campos. Os objetos desta classe também poderão ser adicionados ao mesmo mapa dos clientes comuns.
d) Usando o mapa definido na classe Clientes, defina o método *agendaCliente,* o qual recebe um contato (Contato), uma data (java.time.LocalDate) e adiciona essa associação no mapa. É importante observar que só é possível um agendamento por vez. Ou seja, se o cliente já existir no mapa, ele não é adicionado novamente. Caso a data existente no mapa seja anterior a data atual (agendamento passado), a data é atualizada com a nova, passada como parâmetro. Caso o agendamento ainda seja atual (data existente posterior a data atual), a data passada por parâmetro é descartada.
e) Crie no método main 2 objetos para ilustrar a manipulação das classes definidas. Os valores dos objetos são:

Tipo	Nome	Telefone	Email	Plano	Empresa	CNPJ

Física	Fulano	22222222	fulano@dominio.com	Particular		
Jurídica	Beltrano	33333333	beltrano@dominio.com	Golden Cross	UFF	11111

Adicione estes 2 objetos na coleção declarada.

f) Redefina na classe Clientes o método Object.toString, o qual retorna uma string do objeto da classe Clientes. A string retornada deve exibir os pacientes no formato abaixo:

```
Beltrano 1/10
Fulano 30/9
```

Ou seja, deve exibir o nome e a data (dia/mês) de agendamento de todos os pacientes no sistema.

Dica: Caso tenha dificuldades na iteração de objetos num mapa, observe essa discussão no site StackOverflow: http://stackoverflow.com/questions/46898/iterate-over-each-entry-in-a-map

g) No método main, chame o método toString() para imprimir o conteúdo da coleção.

h) Implemente operações para salvar os dados do sistema, de forma a permitir que estes sejam recuperados em execuções futuras.

7)

```
class Pilha {
	private static int TAM_MAX = 1000;
	private int valores[];
	private int topo;
}
```

O código acima inicia a declaração de uma classe que implementa uma estrutura de dados do tipo pilha. A implementação dessa pilha é baseada em vetor (*valores*). A variável *topo* indica o topo corrente da pilha (-1 para quando a pilha está vazia). A variável TAM_MAX contém o maior tamanho da pilha. Baseado neste código, faça:

(a) Defina um construtor para a classe.
(b) Altere a declaração dos campos de forma que estes não possam ser modificados fora da classe Pilha.
(c) Declare o método *empty*, o qual testa se uma pilha está vazia.
(d) Declare o método *push*, o qual insere um valor no topo da pilha.
(e) Declare uma variação do push, o qual permite o empilhamento de vários valores simultaneamente.
(f) Declare uma outra variação do push que permite que uma pilha inteira seja empilhada em outra pilha.
(g) Declare o método *pop*, o qual remove um valor do topo da pilha e o retorna.
(h) Declare uma variação do método *pop*, a qual recebe um número e desempilha tantos valores quanto o número passado. Caso o valor seja maior que o número de números empilhados, todos os valores são removidos. Esta função não retorna nenhum valor.
(i) Declare o método *top*, o qual apenas retorna o valor do topo da pilha, sem modificá-la.
(j) Crie uma classe Principal com um método main(). Crie um objeto do tipo Pilha e insira os valores 10, 20 e 30. Remova 2 elementos da pilha e exiba o seu topo resultante.
(k) Altere a classe para manipular objetos quaisquer, não apenas inteiros.

8) O código abaixo apresenta um trecho da classe Pedido, a qual é utilizada num sistema que gerencia pedidos de clientes.

```
class Pedido {
	int numero;
	Cliente cliente;
}
```

Baseado neste código, faça:

(a) Defina uma classe Cliente. Neste sistema só é necessário armazenar nomes, telefones e endereços dos clientes. Crie um construtor para facilitar a criação destes.
(b) Altere a classe Pedido para conter também um campo data (objeto da classe java.time.LocalDate), a qual representa a data de realização do pedido, e um campo preço. Crie também um construtor para facilitar a criação de objetos desta classe.
(c) Neste sistema, o cliente também pode solicitar que seu pedido seja entregue de forma expressa. Altere este sistema, sabendo que este tipo terá um novo campo data, a qual conterá a data de entrega do pedido. Pedidos expressos terão seu preço original acrescido de 20%.
(d) Ainda com respeito a datas, pedidos entregues no prazo são pedidos entregues no mesmo dia em que foram solicitados. Insira um método que verifique esta situação, ou seja, retorne verdadeiro se solicitação e entrega dos pedidos foram feitos no mesmo dia. Caso contrário, o método deve retornar falso.
(e) Num método main(), crie 2 clientes: Fulano, morador da rua A, com telefone 9999 e Ciclano, morador da rua B, com telefone 8888. O primeiro cliente realiza um pedido simples, com valor de 500 dinheiros, na data e hora correntes (construtor padrão – *default* – da classe java.time.LocalDate), enquanto que o segundo realiza um pedido expresso com mesmo valor. A empresa consegue entregar o pedido na mesma data. Em seguida, imprima, para cada cliente, seu nome, o preço de seu pedido e se foi entregue no prazo, quando se aplicar. Defina um método chamado toString() na classe Pedido para retornar a String a ser impressa a partir deste método main().
(f) Implemente operações para salvar e recuperar em arquivos as informações de clientes e pedidos.

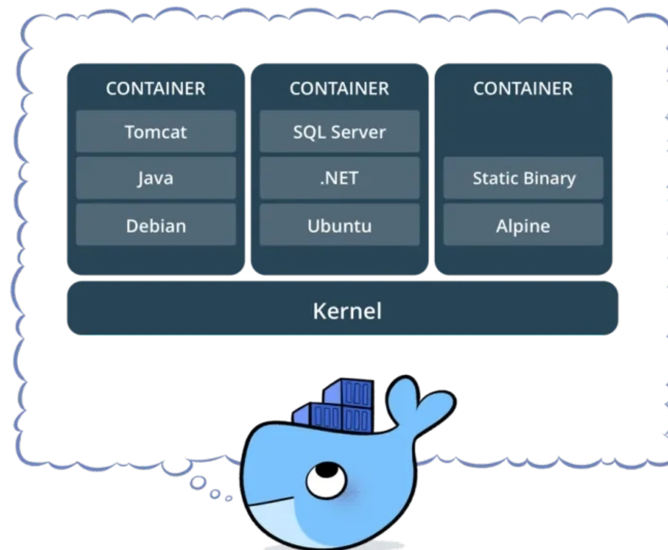
9) O código abaixo apresenta um método main() que utiliza diferentes classes que serão definidas por você, as quais manipulam publicações (livros, revistas, etc).

```
public class Principal {
	public static void main(String[] args) {
		Editora oreilly = new Editora("O'Reilly", "http://oreilly.com/");
		String [] autores = {"Eric Freeman", "Elisabeth Freeman"};
		Publicacao headfirst = new Livro("Padrões de Projeto",
"Programação", "2a", oreilly, autores);
		Editora tres = new Editora("Editora Três",
"http://editora3.terra.com.br/");
		Publicacao oracle = new Revista("Isto É", "Notícias", "2279",
tres, "semanal");
		System.out.println("O'Reilly: " + oreilly.getNumPubs());
	}
}
```

Analisando o código acima, identificamos diferentes classes que precisam ser definidas. Baseado neste código, faça:

(a) Defina uma classe chamada Editora, com seu respectivo construtor, a qual tem como campos seu nome e o seu website.
(b) Defina uma classe chamada Livro e seu respectivo construtor. Os campos definem um objeto do tipo livro e são informados na sua construção são o nome do livro, o assunto, a edição, a editora e os autores.
(c) De forma similar, defina uma classe chamada Revista. Os campos dos objetos desta são o nome, o assunto, a edição, a editora e a periodicidade (diária, semanal, quinzenal, mensal, etc).
(d) Observe que, no código acima, os objetos *headfirst* e *oracle* são declarados do tipo Publicacao, mas instanciados com as classes Livro e Revista, respectivamente. Defina uma classe chamada Publicacao. Qual a relação entre as classes Publicacao, Livro e Revista? Se necessário, altere-as de forma a refletir esta relação.
(e) No último comando deste código é chamado o método *getNumPubs()*. Este método retorna o número de publicações atribuídas a uma dada editora. Para calcular este valor podemos declarar um contador na classe Editora e alterar algum construtor criado para atualizar este contador sempre que uma publicação citar esta editora.
(f) Ao final do método *main()* crie um objeto que represente esta publicação: http://shop.oreilly.com/product/0636920029274.do Em seguida, imprima a quantidade de publicações de cada editora criada.
(g) Crie métodos para salvar e recuperar estas publicações de arquivos, de forma que estes dados possam existir entre as execuções da aplicação.

10) Contêineres em TI funcionam como máquinas virtuais leves que permitem, por exemplo, criarmos diferentes ambientes para testes de aplicações. Na figura abaixo, a máquina que hospeda os contêineres (máquina host) contém um kernel, sobre o qual estão instalados 3 contêineres.



Imagine que queiramos simular este ambiente. Considere o código abaixo:

```
1: public class AP3_2020_2_Q1 {
2:      public static void main(String[] args) {
3:              Software sublime = new Software("Sublime", 100);
4:              Software firefox = new Software("Firefox", 1500);
5:              Software chrome = new Software("Chrome", 2500);
6:              Software eclipse = new Software("Eclipse", 500);
7:              Software ubuntu = new Software("Ubuntu", 2000);
8:
9:              Container c1 = new Container();
10:             c1.addSoftware(chrome);
11:             c1.addSoftware(sublime);
12:
13:             Container c2 = new Container();
14:             c2.addSoftware(ubuntu);
15:             c2.addSoftware(firefox);
16:             c2.addSoftware(eclipse);
17:
18:             Compose containers = new Compose();
19:             containers.addContainer(c1);
20:             containers.addContainer(c2);
21:
22:             containers.run();
23:             c1.stop();
24:
25:             System.out.println("Tamanho total: " + containers.getTamanhoTempoReal());
26:     }
27:}
```

Das linhas 3-7 criamos softwares que podem ser instalados em contêineres. Nesta criação informamos os seus nomes e respectivos tamanhos. Das linhas 9-11 temos um exemplo de criação de contêineres e adição de softwares. De forma a facilitar a manipulação de diversos contêineres, na linha 18 temos um novo tipo (classe) sendo usado para facilitar esta manipulação. Na linha 22 vemos um exemplo de manipulação, onde todos os contêineres são executados a partir deste comando (neste nosso cenário,

um contêiner, quando criado, possui estado inicial inativo). Apesar da facilidade de manipulação conjunta de contêineres, podemos manipulá-los de forma individual também, como está ocorrendo na linha 23, onde interrompemos a execução de um contêiner. Na linha 25 temos a chamada ao método getTamanhoTempoReal, o qual retornará a soma dos tamanhos de todos os softwares de contêineres que estejam ativos.

Implemente os recursos necessários de forma que o código acima funcione. Use OO sempre que possível. **Para testar sua implementação, reescreva a main() criando os contêineres da figura. Execute-os usando o tipo Compose e imprima o tamanho total dos contêineres**. Leve em conta os softwares listados na figura acima, os quais possuem os seguintes tamanhos:

Software	Tomcat	Java	Debian	SQL Server	.NET	Ubuntu	Static Binary	Alpine
Tamanho	300	1000	2000	400	1000	2500	200	800