Instituto de Computação
Bacharelado em Sistemas de Informação
Disciplina: Tópicos em Projeto e Implementação de Sistemas I
Primeira lista de exercícios - 2014.1

UML

1) Quando é apropriado modelar utilizando-se de composição? Por exemplo, por que não utilizar a notação de composição da UML para mostrar que um cachorro é composto de altura, peso, cor e data de nascimento?

2) Um pão fatiado é constituído de fatias de pão. A associação entre pão e suas fatias corresponde a uma composição ou uma agregação?

3) Esboce um diagrama de agregação de objetos para um capítulo de livro com a seguinte estrutura: um capítulo compreende diversas seções, cada uma das quais compreende diversos parágrafos e figuras. Um parágrafo compreende diversas sentenças, cada uma das quais compreende diversas palavras. (Você pode ignorar a pontuação e não precisa ir mais adiante em busca da estrutura de uma figura.). Obs.: não esqueça de atribuir as multiplicidades.

4) Um município decide efetuar o recadastramento biométrico de seus eleitores. O sistema deve armazenar para cada eleitor o seu nome, identidade, cpf, endereço residencial e uma foto. Além disso, para cada eleitor é armazenada a impressão digital de cada um de seus dedos, todas obtidas eletronicamente. Cada eleitor é agrupado em seções que por sua vez pertencem a uma zona eleitoral. O sistema deve ser capaz de reconhecer um eleitor por sua impressão digital durante o processo eleitoral. A participação de cada eleitor em um turno de uma eleição, que ocorre em uma data específica, deve ser registrada no sistema que também computa o total de votos por seção e por zona de cada candidato da eleição. Escreva um diagrama de classes que descreve o sistema eleitoral.

5) Um usuário tem acesso a múltiplos arquivos, cada um acessível por um tipo de permissão de acesso. Descreva em UML o relacionamento entre as classes correspondentes.

6) Mostre um exemplo onde é necessário o uso de uma classe de associação e apresente um outro exemplo onde a classe de associação pode ser removida e mapeada em uma associação convencional.

Polimorfismo

1) Defina um sistema de tipos.

2) Diferencia uma linguagem fortemente tipada de uma linguagem fracamente tipada.

3) Dê exemplos de linguagens fracamente tipadas e justifique sua afirmação.

4) Defina o conceito de compatibilidade de tipos. Quais as duas formas mais comuns de compatibilidade e como funcionam?

5) Defina os diferentes tipos de polimorfismo.

6) Um Sistema de Informações Geográficas precisa gerenciar e manipular mapas de países, os quais são compostos por diferentes objetos geográficos. Considere como objetos geográficos possíveis os limites dos estados, municípios, as rodovias, as ferrovias e os rios. Escreva uma classe que descreva um mapa que pode conter os diferentes objetos geográficos mencionados. Implemente um método da classe Mapa, com assinatura void draw(Canvas c), que exiba cada um dos elementos geográficos em uma superfície de desenho de tipo Canvas. Não é necessário especificar a implementação do modo de exibição de cada tipo específico de objeto geográfico.

Java Generics

1) Diferencie um tipo genérico de um tipo parametrizado. Dê exemplos.

2) Considere uma interface Appendable que inclui um método append. Além disso considere duas classes MyString e MyList, as quais implementam Appendable. Deseja-se que o sistema permita que um objeto de MyString seja concatenado com um objeto MyString e um objeto de MyList possa ser concatenado com outro objeto de MyList, mas que não seja admitido a concatenação de tipos diferentes. Observando a declaração abaixo diga se a interface proposta satisfaz os requisitos especificados quanto a tipagem. Em caso contrário, ofereça uma solução que corrija o problema.

```
interface Appendable{
        Appendable append(Appendable a);
}
```

3) Escreva um método genérico que conte o número de elementos em uma coleção que tenha uma propriedade específica (por exemplo, inteiros ímpares, números primos, palíndromos, etc).

4) Escreva um método genérico que troque duas posições diferentes em um array.

5) Implemente uma árvore binária de busca genérica com iteradores para percorrimento em ordem, pré-ordem, pós-ordem e largura.

6) Implemente um algoritmo genérico que receba uma lista e retorne o elemento máximo.

7) Questão 3 (2.5 Pontos) Considere o código abaixo:

```
public static <E>
   void fazAlgo(List<E> listA, E a, List<? extends E> listB) {
   for (ListIterator<E> it = listA.listIterator(); it.hasNext(); ){
      if (a == null ? it.next() == null : a.equals(it.next())) {
         it.remove();
         for (E e : listB)
            it.add(e);
      }
   }
}
```

a) O código compila sem erros? Caso não compile informe o porquê. (0.5 ponto)
b) Se está correto, descreva o que ele faz e se está incorreto, corrija e informe o que deveria fazer. (1.0 ponto)
c) Explique porque é necessário definir o parâmetro formal de tipo E. (0.5 ponto)
d) Explique porque listB deve ser passado como List<? extends E>. (0.5 ponto)

8) A linguagem Lisp manipula estruturas simbólicas denominadas SExpressions. Uma SExpression é um átomo, isto é um dado considerado indivisível ou então é uma estrutura composta, formada pela sequencia "(", uma SExpression s1, ".", uma segunda SExpression s2 e ")".

Tais expressões simbólicas podem ser interpretadas, armazenadas e manipuladas na forma de uma lista genérica (ou lista de listas), onde cada nó da lista

contém duas referências, que podem apontar para um nó ou para um átomo. Exemplos de listas válidas em Lisp são:

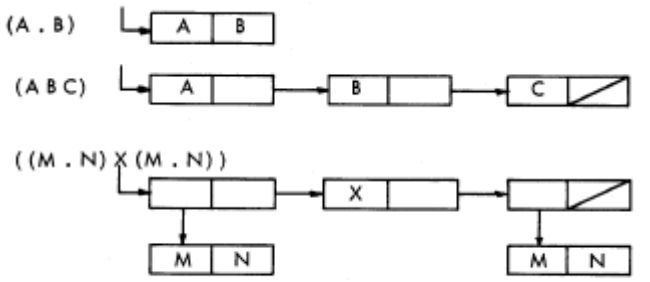


Figura obtida de [1]

Escreva uma classe genérica LispList<T> onde os átomos podem ser do tipo genérico T com as seguintes operações:

> cons - retorna uma nova lista a partir da concatenação de um par de elementos que podem ser dois átomos, um átomo e uma lista ou duas listas existentes.
>
> car - retorna o primeiro elemento da lista (cabeça).
>
> cdr - retorna o restante da lista, excluindo o primeiro elemento (cauda).

Obs.: as operações car e cdr têm comportamento indefinido quando aplicadas sobre átomos. Lembre que as operações podem receber qualquer tipo de lista que seja de um subtipo de T.

[1] John McCarthy, Paul W. Abrahams, Daniel J. Edwards, Timothy P.Hart and Michael I. Levin.Lisp 1.5 Programmer's Manual – The Computation Center and Research Laboratory of Eletronics – Massachusetts Institute of Technology. http://www.softwarepreservation.org/projects/LISP/book/LISP%201.5%20Programmers%20Manual.pdf em 03 de junho de 2013.

Coleções

1) Implemente uma classe pilha como uma extensão de uma Collection<E>. Veja como isto pode ser feito no tutorial da Oracle.

2) Implemente o algoritmo de ordenação HeapSort que atue sobre uma Coleção e que use um critério de ordenação que possa ser configurável para os elementos da coleção.

3) Uma empresa de brinquedos armazena registros de todos os seus empregados contendo nome, CPF, data de ingresso e cargo. Como política da empresa, é comum a doação de certos benefícios aos empregados e uso dos nomes de alguns dos empregados nos seus produtos.

   Considere as quatro interfaces principais da API Collections do Java, Set, List, Queue, and Map. Especifique quais das quatro interfaces é mais apropriada para armazenar os dados necessários para a execução de cada uma das atividades que deve ser realizada pelo sistema da empresa e explique a razão de sua escolha. As atividades são:

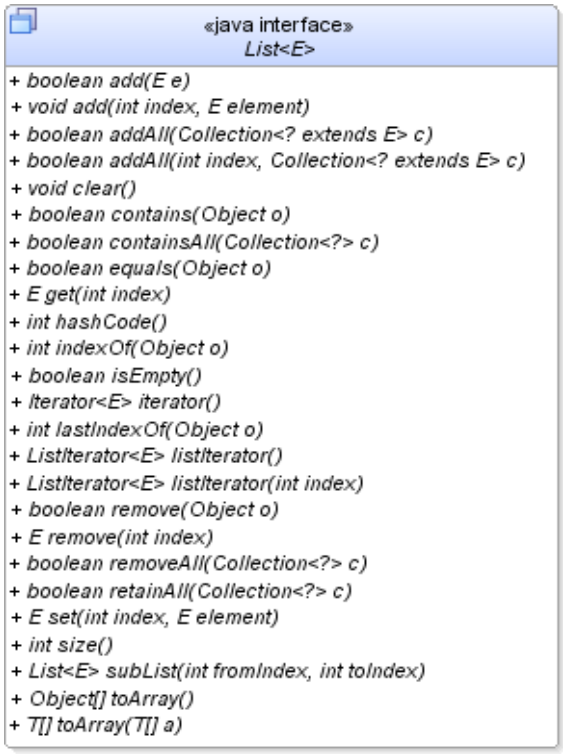
   a) Mensalmente um empregado deve ser escolhido aleatoriamente para receber um brinquedo grátis.

   b) Cada novo produto lançado pela fábrica deve ser nomeado a partir do primeiro nome de um empregado. Na escolha, não devem ser consideradas duplicatas de nomes.

   c) Os brinquedos especiais devem ser batizados com os nomes mais populares da empresa.

   d) A empresa adquiriu ingressos para partidas de futebol que serão distribuídos aos empregados. A empresa mantem uma ordem de espera para distribuição dos ingressos aos empregados, ordem que depende do momento da solicitação do ingresso por cada empregado.

4) Escreva um método genérico que ordene listas genéricas, que implementam a interface List do Java, de acordo com um critério de ordenação arbitrário. Obs.: Não utilize o método sort da classe utilitária Collections. Utilize apenas as operações da interface.


«java interface»
List<E>
+ boolean add(E e)
+ void add(int index, E element)
+ boolean addAll(Collection<? extends E> c)
+ boolean addAll(int index, Collection<? extends E> c)
+ void clear()
+ boolean contains(Object o)
+ boolean containsAll(Collection<?> c)
+ boolean equals(Object o)
+ E get(int index)
+ int hashCode()
+ int indexOf(Object o)
+ boolean isEmpty()
+ Iterator<E> iterator()
+ int lastIndexOf(Object o)
+ ListIterator<E> listIterator()
+ ListIterator<E> listIterator(int index)
+ boolean remove(Object o)
+ E remove(int index)
+ boolean removeAll(Collection<?> c)
+ boolean retainAll(Collection<?> c)
+ E set(int index, E element)
+ int size()
+ List<E> subList(int fromIndex, int toIndex)
+ Object[] toArray()
+ T[] toArray(T[] a)