

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Algoritmos e Estruturas de Dados II

Rodrigo Richard

**Lista de Exercícios**

TAD – Tipo Abstrato de Dados: Lista – Fila – Pilha

Luana Campos Takeishi

Belo Horizonte 1º semestre de 2021

Exercícios escolhidos (10):

**1 –** Crie na CLista o método void InsereAntesDe(Object ElementoAInserir, Object Elemento) que insere o ElementoAInserir na posição anterior ao Elemento passado por parâmetro.

/\*\*

\* Método para inserir antes de - Exercicio 1

\* Insere o Item passado por parâmetro na posição anterior ao outro Item.

\* **@param** ElementoAInserir é o item a ser inserido na lista

\* **@param** Elemento indica a posição, anterior a este item

\* **@return** sem retorno

\*\*/

**public** **void** InsereAntesDe(Object ElementoAInserir, Object Elemento) {

//Verifica se o elemento de referencia existe e guarda sua posição

**boolean** achou = **false**;

**int** pos = 0;

**for** (CCelula aux = primeira.prox; aux != **null** && !achou; aux = aux.prox) {

achou = Elemento.equals(aux.item);

pos++;

}

//Se existir, insere o outro na posição anterior

**if**(achou) {

CCelula aux = primeira;

**for**(**int** i = 0; i < pos-1; i++, aux = aux.prox);

CCelula tmp = **new** CCelula(ElementoAInserir);

tmp.prox = aux.prox;

aux.prox = tmp;

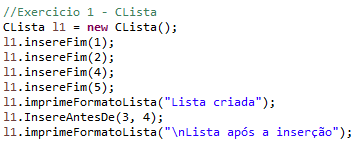
aux = tmp = **null**;

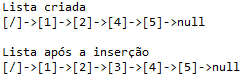
qtde++;

}

}

Teste e resultado:





2 – Crie na CLista o método void InsereDepoisDe(Object ElementoAInserir, Object Elemento) que insere o ElementoAInserir na posição posterior ao Elemento passado por parâmetro.

/\*\*

\* Método para inserir depois de - Exercicio 2

\* Insere o Item passado por parâmetro na posição posterior ao outro Item.

\* **@param** ElementoAInserir é o item a ser inserido na lista

\* **@param** Elemento indica a posição, posterior a este item

\* **@return** sem retorno

\*\*/

**public** **void** InsereDepoisDe(Object ElementoAInserir, Object Elemento) {

//Verifica se o elemento de referencia existe e guarda sua posição

**boolean** achou = **false**;

**int** pos = 0;

**for** (CCelula aux = primeira.prox; aux != **null** && !achou; aux = aux.prox) {

achou = Elemento.equals(aux.item);

pos++;

}

//Se existir, insere o outro na posição posterior

**if**(achou) {

CCelula aux = primeira;

**for**(**int** i = 0; i < pos; i++, aux = aux.prox);

CCelula tmp = **new** CCelula(ElementoAInserir);

tmp.prox = aux.prox;

aux.prox = tmp;

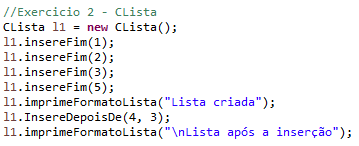
aux = tmp = **null**;

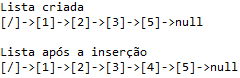
qtde++;

}

}

Teste e resultado:





3 – Crie na CLista o método void InsereOrdenado(int ElementoAInserir) que insere ElementoAInserir em ordem crescente (perceba que para funcionar corretamente, todos os elementos precisarão, necessariamente, ser inseridos através desse método).

/\*\*

\* Método para inserir em ordem crescente - Exercicio 3

\* Insere o Item passado por parâmetro em ordem crescente.

\* **@param** ElementoAInserir é o item a ser inserido na lista

\* **@return** sem retorno

\*\*/

**public** **void** InsereOrdenado(**int** ElementoAInserir) {

**int** pos = 0;

//Procura a posição do elemento

**for** (CCelula aux = primeira.prox; aux != **null**; aux = aux.prox)

**if**(ElementoAInserir > (**int**)aux.item)

pos++;

//Adiciona o elemento na posição

CCelula tmp = **new** CCelula(ElementoAInserir);

CCelula aux = primeira;

**for**(**int** i = 0; i < pos; i++, aux = aux.prox);

tmp.prox = aux.prox;

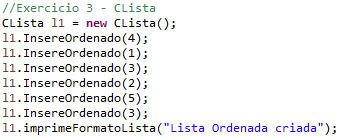
aux.prox = tmp;

aux = tmp = **null**;

qtde++;

}

Teste e resultado:





4 – Crie a função CListaDup ConcatenaLD(CListaDup L1, CListaDup L2) que concatena as listas L1 e L2 passadas por parâmetro, retornando uma lista duplamente encadeada.

CListDup A = new CListaDup ();

CListaDup B = new CListaDup ();

CListaDup AmaisB; // Apenas a referência foi declarada. Uma ListaDup auxiliar deverá ser criada

// dentro da função e retornado pela mesma

// código para preencher as CListaDup A, B

AmaisB = ConcatenaLD(A, B);

A = [19, 33, 2, 4]

B = [1, 2, 3, 4, 5]

AmaisB = [19, 33, 2, 4, 1, 2, 3, 4, 5]

/\*\*

\* Função apara concatenar duas listas duplamente encadeada - Exercício 4

\* Concatena as listas duplamente encadeadas passadas por parâmetro retornando o resultado.

\* **@param** L1 - lista duplamente encadeada 1

\* **@param** L2 - lista duplamente encadeada 2

\* **@return** L2 - lista duplamente encadeada resultado da concatenação

\*\*/

**public** **static** CListaDup ConcatenaLD(CListaDup L1, CListaDup L2) {

CListaDup L3 = **new** CListaDup();

//Concatena L1

**int** i = 1;

**for** (Object aux1 = L1.retornaIndice(i); aux1 != **null**; aux1 = L1.retornaIndice(++i))

L3.insereFim(aux1);

//Concatena L2

i = 1;

**for** (Object aux2 = L2.retornaIndice(i); aux2 != **null**; aux2 = L2.retornaIndice(++i))

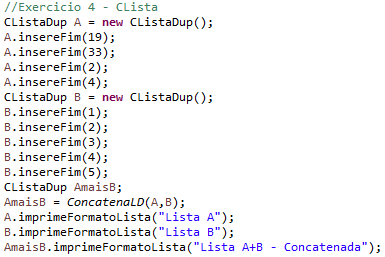
L3.insereFim(aux2);

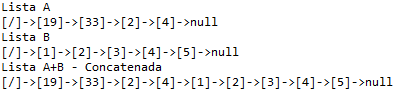
//Retorna Lista Concatenada

**return** L3;

}

Teste e resultado:





5 – Crie a função CFila ConcatenaFila(CFila F1, CFila F2) que concatena as filas F1 e F2 passadas por parâmetro.

\*obs.: Não consegui fazer sem a criação de um novo método, ou que clonasse a fila ou que retornasse o valor contido na posição i. De outra maneira acabaria destruindo as filas de entrada.

/\*\*

\* Retorna o Item contido na posição passada por parâmetro.

\*/

**public** Object retornaIndice(**int** posicao) {

**if** ((posicao >= 1) && (posicao <= qtde) && (frente != tras)) {

CCelula aux = frente.prox;

**for** (**int** i = 1; i < posicao; i++, aux = aux.prox);

**return** aux.item;

}

**return** **null**;

}

/\*\*

\* Função apara concatenar duas filas - Exercício 5

\* Concatena as filas passadas por parâmetro retornando o resultado.

\* **@param** L1 - fila duplamente encadeada 1

\* **@param** L2 - fila duplamente encadeada 2

\* **@return** L2 - fila duplamente encadeada resultado da concatenação

\*\*/

**public** **static** CFila ConcatenaFila(CFila F1, CFila F2) {

CFila F3 = **new** CFila();

//Concatena L1

**int** i = 1;

**for** (Object aux1 = F1.retornaIndice(i); aux1 != **null**; aux1 = F1.retornaIndice(++i))

F3.enfileira(aux1);

//Concatena L2

i = 1;

**for** (Object aux2 = F2.retornaIndice(i); aux2 != **null**; aux2 = F2.retornaIndice(++i))

F3.enfileira(aux2);

**return** F3;

}

Teste e resultado:

|  |  |
| --- | --- |
|  |  |

6 – Crie a função CPilha ConcatenaPilha(CPilha P1, CPilha P2) que concatena as pilhas P1 e P2 passadas por parâmetro.

\*obs.: Não consegui fazer sem a criação de um novo método, ou que clonasse a fila ou que retornasse o valor contido na posição i. De outra maneira acabaria destruindo as filas de entrada.

/\*\*

\* Clona a pilha invertida.

\*/

**public** CPilha cloneI() {

CPilha clone = **new** CPilha();

**for** (CCelula c = topo; c != **null**; c = c.prox)

clone.empilha(c.item);

**return** clone;

}

/\*\*

\* Função apara concatenar duas pilhas - Exercício 6

\* Concatena as pilhas duplamente encadeadas passadas por parâmetro retornando o resultado.

\* **@param** P1 - pilha duplamente encadeada 1

\* **@param** P2 - pilha duplamente encadeada 2

\* **@return** result - pilha duplamente encadeada resultado da concatenação

\*\*/

**public** **static** CPilha ConcatenaPilha(CPilha P1, CPilha P2) {

CPilha P3 = P1.cloneI();

CPilha P4 = P2.cloneI();

CPilha result = **new** CPilha();

//Concatena P1

**for** (Object aux1 = P3.desempilha(); aux1 != **null**; aux1 = P3.desempilha())

result.empilha(aux1);

//Concatena P2

**for** (Object aux2 = P4.desempilha(); aux2 != **null**; aux2 = P4.desempilha())

result.empilha(aux2);

**return** result;

}

Teste e resultado:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

8 – Crie na CListaDup o método int primeiraOcorrenciaDe(Object elemento) que busca e retorna o índice da primeira ocorrência do elemento passado por parâmetro. Caso o elemento não exista, sua função deve retornar um valor negativo. Obs: considere que o primeiro elemento está na posição 1.

/\*\*

\* Método que busca e retorna a posição da primeira ocorrencia

\* de um elemento passado por parâmentro.

\* **@return** pos - posição do elemento

\* se ele não existir, deve retornar o valor negativo -1;

\*/

**public** **int** primeiraOcorrenciaDe(Object elemento) {

**int** pos = 1;

**for** (CCelulaDup aux = primeira.prox; aux != **null**; aux = aux.prox, pos++)

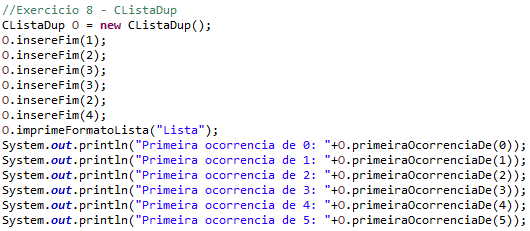
**if** (elemento.equals(aux.item))

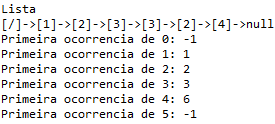
**return** pos;

**return** -1;

}

Teste e resultado:





9 – Crie na CListaDup o método int ultimaOcorrenciaDe(Object elemento) que busca e retorna o índice da última ocorrência do elemento passado por parâmetro. Caso o elemento não exista, sua função deve retornar um valor negativo. Obs: considere que o primeiro elemento está na posição 1.

/\*\*

\* Método que busca e retorna a posição da última ocorrencia

\* de um elemento passado por parâmentro.

\* **@return** posE - posição do elemento

\* se ele não existir, deve retornar o valor negativo -1;

\*/

**public** **int** ultimaOcorrenciaDe(Object elemento) {

**int** pos = 1;

**int** posE = -1;

**for** (CCelulaDup aux = primeira.prox; aux != **null**; aux = aux.prox, pos++)

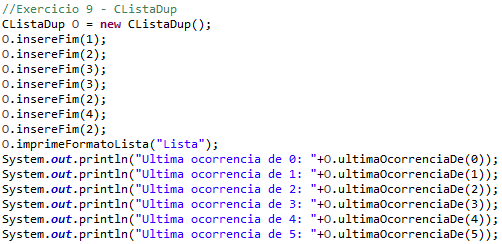
**if** (elemento.equals(aux.item))

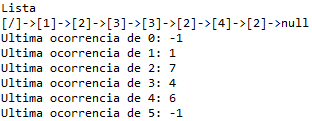
posE = pos;

**return** posE;

}

Teste e resultado:





11 – Crie na CLista o método void RemovePos(int n) que remove o elemento na n-ésima posição da lista.

/\*\*

\* Método para remover item na n-ésima posição.

\* Remove o item da posição passada por parâmetro.

\* **@param** n - n-ésima posição.

\* **@return** sem retorno.

\*\*/

**public** **void** RemovePos(**int** n) {

**if** ((n >= 1) && (n <= qtde) && (primeira != ultima)) {

CCelula aux = primeira;

**for**(**int** i = 0; i < n-1; i++)

aux = aux.prox;

aux.prox = aux.prox.prox;

**if** (aux.prox == **null**)

ultima = aux;

qtde--;

}

}

Teste e resultado:

|  |  |
| --- | --- |
|  |  |

12 – Crie na CListaDup o método void RemovePos(int n) que remove o elemento na n-ésima posição da lista.

/\*\*

\* Método para remover item na n-ésima posição - Exercicio 12

\* Remove o item da posição passada por parâmetro.

\* **@param** n - n-ésima posição.

\* **@return** sem retorno.

\*\*/

**public** **void** RemovePos(**int** n) {

**if** ((n >= 1) && (n <= qtde) && (primeira != ultima)) {

CCelulaDup aux = primeira;

**for**(**int** i = 0; i < n; i++)

aux = aux.prox;

aux.ant.prox = aux.prox;

**if** (aux.prox != **null**)

aux.prox.ant = aux.ant;

**else**

ultima.ant = aux;

qtde--;

}

}

Teste e resultado:

|  |  |
| --- | --- |
|  |  |

13 – Crie na CFila o método int qtdeOcorrencias(Object elemento) a qual retorna a quantidade de vezes que o elemento passado como parâmetro está armazenado na CFila.

/\*\*

\* Metodo que retorna a quantidade de vezes em

\* que o elemento passado por parâmetro está na fila.

\* **@param** elemento - elemento a ser pesquisado.

\* **@return** quantidade - quantidade de aparições.

\*/

**public** **int** qtdeOcorrencias(Object elemento) {

**int** quantidade = 0;

**for** (CCelula aux = frente.prox; aux != **null**; aux = aux.prox)

**if** (elemento.equals(aux.item))

quantidade++;

**return** quantidade;

}

Teste e resultado:

|  |  |
| --- | --- |
|  |  |

14 – Crie na CPilha o método void inverte() que inverte a ordem dos elementos da Pilha.

/\*\*

\* Método que inverte a ordem dos elementos da Pilha.

\*/

**public** **void** inverte() {

CPilha nova = **new** CPilha();

**for**(Object auxo = **this**.desempilha(); auxo != **null**; auxo = **this**.desempilha())

nova.empilha(auxo);

**this**.topo = nova.topo;

**this**.qtde = nova.qtde;

}

Teste e resultado:

|  |  |  |
| --- | --- | --- |
|  |  |  |

15 – Crie na CFila o método void inverte() que inverte a ordem dos elementos da Fila.

/\*\*

\* Método que inverte a ordem dos elementos da Fila.

\*/

**public** **void** inverte() {

CFila nova = **new** CFila();

**for** (**int** i = qtde; i > 0; i--) {

CCelula aux = frente.prox;

**for** (**int** j = 0;j < i-1; aux = aux.prox, j++);

nova.enfileira(aux.item);

}

**this**.frente = nova.frente;

**this**.tras = nova.tras;

**this**.qtde = nova.qtde;

}

Teste e resultado:

|  |  |
| --- | --- |
|  |  |

16 - Crie na CLista o método Object[] copiaParaVetor() que copia todos os elementos da Lista para um vetor.

/\*\*

\* Método que copia todos os elementos da Lista para um vetor.

\* **@return** vetor com elementos.

\*\*/

**public** Object[] copiaParaVetor() {

Object[] vector = **new** Object[qtde];

**int** i = 0;

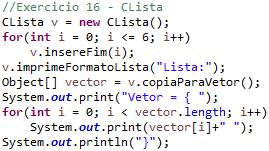
**for** (CCelula aux = primeira.prox; aux != **null**; aux = aux.prox, i++)

vector[i] = aux.item;

**return** vector;

}

Teste e resultado:





17 – Crie a função construtora CListaDup(Object[] VET) na classe CListaDup que receba um vetor como parâmetro e crie a lista duplamente encadeada com todos os elementos contidos nesse vetor.

/\*\*

\* Função construtora a partir de um vetor.

\* **@param** VET - vetor de objetos para construir a lista.

\*\*/

CListaDup(Object[] VET){

primeira = **new** CCelulaDup();

ultima = primeira;

**for**(**int** i = 0; i < VET.length; i++) {

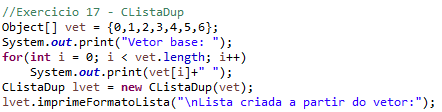
ultima.prox = **new** CCelulaDup(VET[i], ultima, **null**);

ultima = ultima.prox;

qtde++;

}

Teste e resultado:





18 – Crie a função void InvertePilha(CPilha P) que inverte a pilha P recebida como parâmetro. Use qualquer estrutura adicional que achar necessário.

/\*\*

\* Função para inverter uma pilha - Exercício 18

\* Inverte a pilha P recebida como parâmetro.

\* **@param** P - pilha a ser invertida

\*\*/

**public** **static** **void** InvertePilha(CPilha P) {

CPilha Pc = P.clone();

//Poderia ser substituido pelo Limpar da questão 20

**for** (Object aux = P.desempilha(); aux != **null**; aux = P.desempilha());

**for** (Object aux = Pc.desempilha(); aux != **null**; aux = Pc.desempilha())

P.empilha(aux);

}

|  |
| --- |
| /\*\*  \* Clona a pilha.  \*/  **public** CPilha clone() {  CPilha clone = **new** CPilha();  CPilha cloneI = **new** CPilha();  **for** (CCelula c = topo; c != **null**; c = c.prox)  cloneI.empilha(c.item);  **for** (CCelula c = cloneI.topo; c != **null**; c = c.prox)  clone.empilha(c.item);  **return** clone;  } |
|  |

19 – Crie a função void InverteFila(CFila F) que inverte a fila F recebida como parâmetro. Use qualquer estrutura adicional que achar necessário.

/\*\*

\* Função para inverter uma fila - Exercício 19

\* Inverte a fila F recebida como parâmetro.

\* **@param** F - fila a ser invertida

\*\*/

**public** **static** **void** InverteFila(CFila F) {

CFila Fc = F.clone();

//Poderia ser substituido pelo Limpar da questão 20

**for** (Object aux = F.desenfileira(); aux != **null**; aux = F.desenfileira());

**int** i = Fc.quantidade();

**for** (Object aux1 = Fc.retornaIndice(i); aux1 != **null**; aux1 = Fc.retornaIndice(--i))

F.enfileira(aux1);

}

Teste e resultado:

/\*\*

\* Método que clona a Fila.

\*/

**public** CFila clone() {

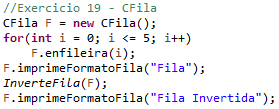
CFila clone = **new** CFila();

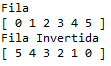
**for** (CCelula aux = frente.prox; aux != **null**; aux = aux.prox)

clone.enfileira(aux.item);

**return** clone;

}





20 – Cria o método void Limpar() para todas as classes (CLista, CListaDup, CFila e CPilha), o qual deve remover todos os itens da estrutura.

/\*\*

\* Método limpa a Lista Dupla.

\*\*/

**public** **void** Limpar() {

primeira.prox = **null**;

ultima = primeira;

qtde = 0;

}

/\*\*

\* Método limpa a Lista.

\*\*/

**public** **void** Limpar() {

primeira.prox = **null**;

ultima = primeira;

qtde = 0;

}

/\*\*

\* Método limpa a Pilha.

\*\*/

**public** **void** Limpar() {

topo = **null**;

qtde = 0;

}

/\*\*

\* Método limpa a Fila.

\*\*/

**public** **void** Limpar() {

frente.prox = **null**;

tras = frente;

qtde = 0;

}

|  |  |
| --- | --- |
|  |  |

21 – Crie a função construtora CFila(Object[] vetor) na classe CFila que receba um vetor de Object como parâmetro e crie a fila com todos os elementos do vetor.

/\*\*

\* Função construtora a partir de um vetor.

\* **@param** VET - vetor de objetos para construir a fila.

\*\*/

CFila(Object[] VET){

frente = **new** CCelula();

tras = frente;

**for**(**int** i = 0; i < VET.length; i++) {

tras.prox = **new** CCelula(VET[i]);

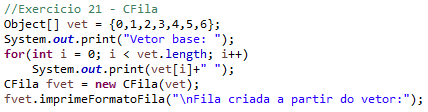
tras = tras.prox;

qtde++;

}

}

Teste e resultado:





22 – Crie a função construtora CFila(CPilha P) na classe CFila que receba uma Pilha como parâmetro e crie a fila com todos os elementos da Pilha de forma que a ordem de retirada dos elementos seja a mesma ordem de retirada dos elementos da Pilha.

23 – Crie a função construtora CFila(CFila F) na classe CFila que crie a fila com todos os elementos da Fila F recebida como parâmetro.

24 – Crie na classe CLista o método void InsereEspelhado(Object item), o qual insere o elemento no início e no final da lista. Assim, as chamadas para inserir os elementos 1, 2 e 3 deveriam resultar na seguinte lista [3 2 1 1 2 3].

25 – Crie na classe CFila o método void RemoverApos(Object item), o qual remove TODOS os elementos que seguem o item passado como parâmetro.

26 – Crie a função construtora CPilha(CPilha P) na classe CPilha que recebe a Pilha P passada como parâmetro e copia todos os seus elementos (sem destruí-la) para a nova pilha que está sendo criada.

27 - Crie a função public void VaiProFundo(CPilha P, Object elemento) que empilha o elemento passado como parâmetro no fundo da CPilha P, ao invés de no topo.

28 - Crie a função public void FuraFIla(CFila F, Object elemento) que insere o elemento no início da Fila F. Obs: você pode utilizar outras estruturas auxiliares que julgue necessárias.

29 - Crie na classe CFila o método public void FuraFIla(Object elemento) que insere o elemento no início da Fila.

Exercícios obrigatórios (5):

\* 7 – A classe RandomQueue é uma Fila que retorna elementos aleatórios ao invés de sempre retornar o primeiro elemento. Crie a classe RandomQueue com os seguintes métodos:

class RandomQueue {

RandomQueue() { } // Construtora – cria uma RandomQueue vazia

bool IsEmpty() { } // Retorna true se a RandomQueue estiver vazia

void Enqueue(Object item) { } // Adiciona um item

Object Dequeue() { } // Remove e retorna um elemento aleatório da RandomQueue

Object Sample() { } // Retorna um elemento aleatório sem removê-lo da RandomQueue

}

**import** java.util.Random;

/\*\*

\*

\* **@coauthor** Luana Campos Takeishi

\* **@version** 1.01 2021/5/29

\* Lista 2 - 2o semestre AEDII

\*

\*/

**public** **class** RandomQueue {

**private** CCelula comeco; // Celula cabeca.

**private** CCelula fim; // Ultima celula.

**private** **int** q;

/\*\*

\* Função construtora.

\* Cria a célula cabeça e faz as referências

\* comeco e fim apontarem para ela.

\* Fila aleatória vazia.

\*/

**public** RandomQueue() {

comeco = **new** CCelula();

fim = comeco;

}

/\*\*

\* Verifica se a fila está vazia.

\* **@return** Retorna TRUE se a fila estiver vazia

\* e FALSE se conter elementos.

\*/

**public** **boolean** IsEmpty() {

**return** comeco == fim;

}

/\*\*

\* Insere um novo Item no fim da fila.

\* **@param** item - object com o valor a

\* ser inserido no final da fila.

\*/

**public** **void** Enqueue(Object item) {

fim.prox = **new** CCelula(item);

fim = fim.prox;

q++;

}

/\*\*

\* Retira e retorna um elemento aleatório da fila.

\* **@return** item retirado aleatoriamente da fila,

\* se a fila estiver vazia, retorna null.

\*/

**public** Object Dequeue() {

Random rand = **new** Random();

**int** posicao = rand.nextInt(q);

CCelula item = **null**;

**if** (comeco != fim) {

**int** i = 0;

CCelula aux = comeco;

**while** (i < posicao) {

aux = aux.prox;

i++;

}

item = aux.prox;

aux.prox = aux.prox.prox;

**if** (aux.prox == **null**)

fim = aux;

q--;

}

**return** item.item;

}

/\*\*

\* Retorna um elemento aleatório da fila sem removê-lo.

\* **@return** item aleatório da fila,

\* se a fila estiver vazia, retorna null.

\*/

**public** Object Sample() {

Random rand = **new** Random();

**int** posicao = rand.nextInt(q);

CCelula item = **null**;

**if** (comeco != fim) {

item = comeco.prox;

**for** (**int** i = 1; i <= posicao; i++, item = item.prox);

}

**return** item.item;

}

/\*\*

\* Imprime todos os elementos simulando formato de lista:

\* [/]->[x]->[y]->[z]->null

\*/

**public** **void** imprimeFormatoLista(String titulo) {

System.***out***.println(titulo);

System.***out***.print("[/]->");

**for** (CCelula aux = comeco.prox; aux != **null**; aux = aux.prox)

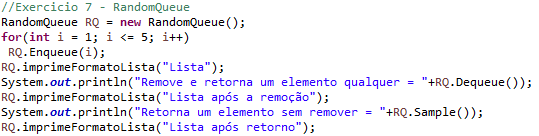
System.***out***.print("[" + aux.item + "]->");

System.***out***.println("null\n");

}

}

Testes e resultados:



|  |  |
| --- | --- |
|  |  |
|  |  |

\* 10 – Deque (Double-ended-queue) é um Tipo Abstrato de Dados (TAD) que funciona como uma Fila e como uma Pilha, permitindo que itens sejam adicionados em ambos os extremos. Implemente a classe Deque, usando duplo encadeamento, com os seguintes métodos:

class Deque {

Deque() { } // Construtora – cria uma Deque vazia

boolean isEmpty() { } // Retorna true se a Deque estiver vazia

int size() { } // Retorna a quantidade de itens da Deque

void pushLeft(Object item) { } // Adiciona um item no lado esquerdo da Deque

void pushRight(Object item) { } // Adiciona um item no lado direito da Deque

Object popLeft() { } // Remove e retorna um item do lado esquerdo da Deque

Object popRight() { } // Remove e retorna um item do lado direito da Deque

}

/\*\*

\*

\* **@author** Luana Campos Takeishi

\* **@version** 1.01 2021/5/29

\* Lista 2 - 2o semestre AEDII

\*

\*/

**public** **class** Deque {

**private** CCelulaDup esquerda; // Referencia a ultima celula da esquerda (cabeça)

**private** CCelulaDup direita; // Referencia a primeira celula da direita (rabo)

**private** **int** q;

/\*\*

\* Função construtora. Cria as células cabeça e rabo

\* e faz as referências direita e esquerda

\* apontarem entre si.

\*/

**public** Deque() {

esquerda = **new** CCelulaDup();

direita = **new** CCelulaDup();

esquerda.prox = direita;

direita.ant = esquerda;

}

/\*\*

\* Verifica se a Double-ended-queue está vazia.

\* **@return** TRUE se a deque estiver vazia

\* e FALSE caso tenha elementos.

\*/

**public** **boolean** isEmpty() {

**return** esquerda.prox == direita & direita.ant == esquerda;

}

/\*\*

\* Metodo que retorna a quantidade de itens da deque.

\*

\* **@return q -** quantidade de itens da deque;

\*/

**public** **int** size() {

**return** q;

}

/\*\*

\* Insere um novo Item no "começo" - lado esquerdo da Deque.

\* **@param** item - elemento a ser inserido no começo - esquerda.

\*/

**public** **void** pushLeft(Object item) {

**if** (isEmpty())

esquerda.prox = direita.ant = **new** CCelulaDup(item, esquerda, direita);

**else**

esquerda.prox = esquerda.prox.ant = **new** CCelulaDup(item, esquerda, esquerda.prox);

q++;

}

/\*\*

\* Insere um novo Item no "fim" - lado direito da Deque.

\* **@param** item - elemento a ser inserido no fim - direita.

\*/

**public** **void** pushRight(Object item) {

**if** (isEmpty()) {

esquerda.prox = direita.ant = **new** CCelulaDup(item, esquerda, direita);

**else**

direita.ant = direita.ant.prox = **new** CCelulaDup(item, direita.ant, direita);

q++;

}

/\*\*

\* Remove o Item no "começo" - lado esquerdo da Deque.

\* **@param** item - elemento a ser removido do começo - esquerda.

\*/

**public** Object popLeft() {

**if** (!isEmpty()) {

CCelulaDup aux = esquerda.prox;

esquerda.prox = esquerda.prox.prox;

esquerda.prox.prox.ant = esquerda.prox;

q--;

**return** aux.item;

}

**return** **null**;

}

/\*\*

\* Remove o Item no "fim" - lado direito da Deque.

\* **@param** item - elemento a ser removido do fim - direita.

\*/

**public** Object popRight() {

**if** (!isEmpty()) {

CCelulaDup aux = direita.ant;

direita.ant.ant.prox = direita;

direita.ant = direita.ant.ant;

q--;

**return** aux.item;

}

**return** **null**;

}

/\*\*

\* Imprime Deque.

\*/

**public** **void** imprimeFormatoLista(String titulo) {

System.***out***.println(titulo);

System.***out***.print("[esq]<->");

**for** (CCelulaDup aux = esquerda.prox; aux != **null** && aux != direita; aux = aux.prox)

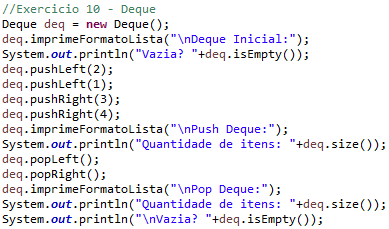
System.***out***.print("[" + aux.item + "]<->");

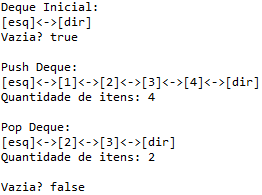
System.***out***.println("[dir]");

}

}

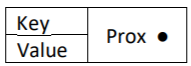
Teste e resultado:





\* 30 – Crie as classes CCelulaDicionario e CDicionario conforme a interface abaixo:

A classe CDicionario é muito semelhante à classe CLista. A principal diferença fica por conta da célula, que ao invés de ter apenas o valor do item e a referência para a próxima célula, tem também uma chave associada ao valor adicionado.



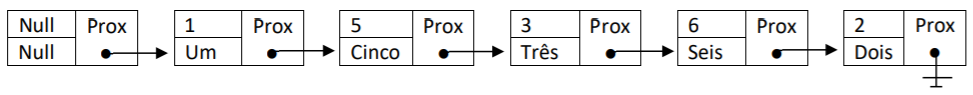
Algumas observações sobre sua classe:

• A construtora de sua classe CDicionario deve criar uma célula cabeça;

• O método Adicionar deve adicionar o novo emento (chave/valor) na última posição do dicionário. Atenção: sua classe não deve permitir a inserção de elementos com chaves duplicadas;

• O método RecebeValor deve localizar e retonar o valor associado à chave passada por parâmetro. Caso a chave não exista, o método deve retornar null.

Exemplo de um Dicionario cuja chave é um número inteiro e o valor é o valor por extenso.



Agora, usando sua classe CDicionario, crie um dicionário com URL’s e IP’s dos websites abaixo e mais 5 à sua escolha. O seu dicionário terá a URL como chave e o IP correspondente como valor (por exemplo, se digitarmos como chave a URL www.google.com, seu programa deve retornar o IP 74.125.234.81). O seu programa deve permitir que o usuário digite uma URL e deve imprimir o IP correspondente. Para descobrir o IP de um website, basta digitar ping + URL do website (exemplo: ping www.google.com).



/\*\*

\* **@author** Luana Campos Takeishi

\* **@version** 1.00 2021/5/30

\*/

**public** **class** CCelulaDicionario {

// Atributos

**public** Object key, value;

**public** CCelulaDicionario prox;

// Construtora que anula os três atributos da célula

**public** CCelulaDicionario() {

key = value = prox = **null**;

}

// Construtora que inicializa key e value com os argumentos passados

// por parâmetro e anula a referência à próxima célula

**public** CCelulaDicionario(Object chave, Object valor) {

key = chave;

value = valor;

prox = **null**;

}

// Construtora que inicializa todos os atribulos da célula com os argumentos

// passados por parâmetro

**public** CCelulaDicionario(Object chave, Object valor, CCelulaDicionario proxima) {

key = chave;

value = valor;

prox = proxima;

}

}

/\*\*

\*

\* **@author** Luana Campos Takeishi

\* **@version** 1.00 2021/5/30

\*

\*/

**public** **class** CDicionario {

**private** CCelulaDicionario primeira, ultima;

/\*\*

\* Função Construtora.

\* Aloca a célçula cabeça e aponta as referências à ela.

\*/

**public** CDicionario() {

primeira = ultima =**new** CCelulaDicionario();

}

/\*\*

\* Método que verifica se o dicionário está vazio.

\* **@return** true - caso dicionário vazio.

\* **@return** false - se contem elementos.

\*/

**public** **boolean** vazio() {

**return** primeira == ultima;

}

/\*\*

\* Método que adiciona o par chave/valor

\* na última posição do dicionário.

\* Não permite chaves duplicadas.

\* **@param** chave e valor a serem adicionados.

\*/

**public** **void** adiciona(Object chave, Object valor) {

**boolean** existe= **false**;

**for** (CCelulaDicionario aux = primeira.prox; aux != **null** && !existe; aux = aux.prox)

existe = chave.equals(aux.key);

**if**(!existe) {

ultima.prox = **new** CCelulaDicionario(chave, valor);

ultima = ultima.prox;

}

}

/\*\*

\* Método que retorna o valor associado a chave.

\* **@param** chave a ser pesquisada.

\* **@return** valor associado.

\* Caso não exista, retorna null;

\*/

**public** Object recebeValor(Object chave) {

**for** (CCelulaDicionario aux = primeira.prox; aux != **null**; aux = aux.prox)

**if**(chave.equals(aux.key))

**return** aux.value;

**return** **null**;

}

/\*\*

\* Método que imprimir discionário.

\*/

**public** **void** imprimeFormatoDicio(String titulo) {

System.***out***.println(titulo);

**for** (CCelulaDicionario aux = primeira.prox; aux != **null**; aux = aux.prox)

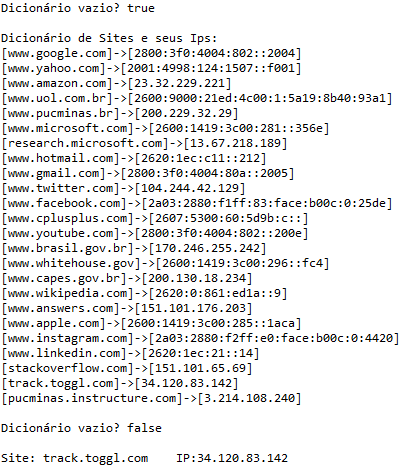
System.***out***.println("[" + aux.key + "]->[" + aux.value + "]");

}

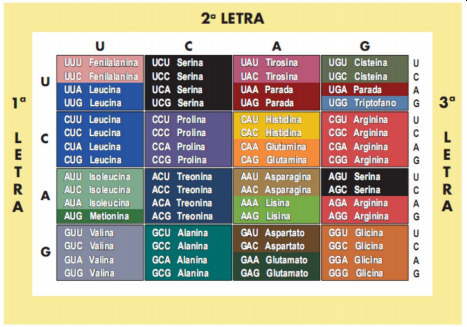
}

Teste e resultado:





31 – Um biólogo precisa de um programa que traduza uma trinca de nucleotídeos em seu aminoácido correspondente. Por exemplo, a trinca de aminoácidos ACG é traduzida como o aminoácido Treonina, e GCA em Alanina. Crie um programa em Java que use a sua classe CDicionario para criar um dicionário do código genético. O usuário deve digitar uma trinca (chave) e seu programa deve mostrar o nome (valor) do aminoácido correspondente. Use a tabela a seguir para cadastrar todas as trincas/aminoácidos.



//Exercicio 30 - CCelulaDicionario e CDicionario

CDicionario amino = **new** CDicionario();

//Cadastro

amino.adiciona("UUU","Fenilalanina"); amino.adiciona("UUC","Fenilalanina"); amino.adiciona("UUA","Leucina"); amino.adiciona("UUG","Leucina");

amino.adiciona("CUU","Leucina"); amino.adiciona("CUC","Leucina"); amino.adiciona("CUA","Leucina"); amino.adiciona("CUG","Leucina");

amino.adiciona("AUU","Isoleucina"); amino.adiciona("AUC","Isoleucina"); amino.adiciona("AUA","Isoleucina"); amino.adiciona("AUG","Metionina");

amino.adiciona("GUU","Valina"); amino.adiciona("GUC","Valina"); amino.adiciona("GUA","Valina"); amino.adiciona("GUG","Valina");

amino.adiciona("UCU","Serina"); amino.adiciona("UCC","Serina"); amino.adiciona("UCA","Serina"); amino.adiciona("UCG","Serina");

amino.adiciona("CCU","Prolina"); amino.adiciona("CCC","Prolina"); amino.adiciona("CCA","Prolina"); amino.adiciona("CCG","Prolina");

amino.adiciona("ACU","Treonina"); amino.adiciona("ACC","Treonina"); amino.adiciona("ACA","Treonina"); amino.adiciona("ACG","Treonina");

amino.adiciona("GCU","Alanina"); amino.adiciona("GCC","Alanina"); amino.adiciona("GCA","Alanina"); amino.adiciona("GCG","Alanina");

amino.adiciona("UAU","Tirosina"); amino.adiciona("UAC","Tirosina"); amino.adiciona("UAA","Ocre/Parada"); amino.adiciona("UAG","Âmbar/Parada");

amino.adiciona("CAU","Histidina"); amino.adiciona("CAC","Histidina"); amino.adiciona("CAA","Glutamina"); amino.adiciona("CAG","Glutamina");

amino.adiciona("AAU","Asparagina"); amino.adiciona("AAC","Asparagina"); amino.adiciona("AAA","Lisina"); amino.adiciona("AAG","Lisina");

amino.adiciona("GAU","Ácido aspártico");amino.adiciona("GAC","Ácido aspártico"); amino.adiciona("GAA","Ácido glutâmico");amino.adiciona("GAG","Ácido glutâmico");

amino.adiciona("UGU","Cisteína"); amino.adiciona("UGC","Cisteína"); amino.adiciona("UGA","Opala/Parada"); amino.adiciona("UGG","Triptofano");

amino.adiciona("CGU","Arginina"); amino.adiciona("CGC","Arginina"); amino.adiciona("CGA","Arginina"); amino.adiciona("CGG","Arginina");

amino.adiciona("AGU","Serina"); amino.adiciona("AGC","Serina"); amino.adiciona("AGA","Arginina"); amino.adiciona("AGG","Arginina");

amino.adiciona("GGU","Glicina"); amino.adiciona("GGC","Glicina"); amino.adiciona("GGA","Glicina"); amino.adiciona("GGG","Glicina");

amino.imprimeFormatoCodao("Codão - código genético:");

Object i = 0;

System.***out***.println("\nDicionário de código genético:");

System.***out***.println("Entre com a trinca para receber o nome do aminoácido\nou com outro valor para parar a execução.");

**while**(i != **null**) {

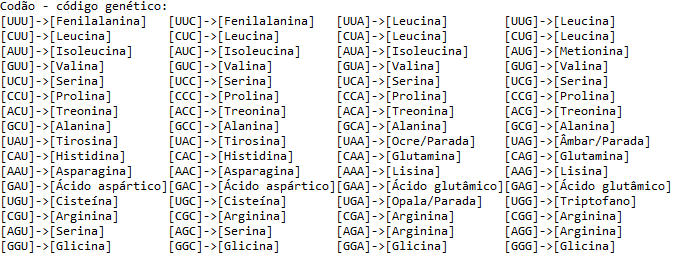
System.***out***.println("\nDigite a trinca:");

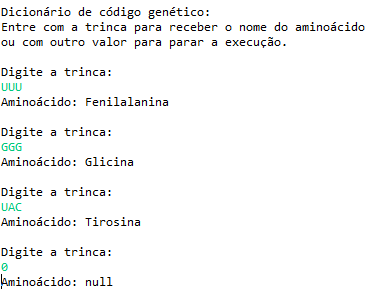
i = amino.recebeValor(scan.nextLine());

System.***out***.println("Aminoácido: " + i);

}

Teste e resultado:





\* 32 – Crie a classe CListaSimples que é uma lista simplesmente encadeada sem célula cabeça e que possui apenas os métodos definidos na interface abaixo. Atenção: não podem ser acrescentados novos atributos ou métodos às classes CListaSimples e/ou CCelula abaixo.

/\*\*

\*

\* **@author** Luana Campos Takeishi

\* **@version** 1.00 2021/6/01

\*

\*/

**class** CCelulaSimples {

**public** Object item;

**public** CCelulaSimples prox;

}

**public** **class** CListaSimples {

**private** CCelulaSimples primeira, ultima;

/\*\*

\* Função Construtora.

\*/

**public** CListaSimples() {

//Vazio

}

/\*\*

\* Método que verifica se a lista está vazia.

\* **@return** true - caso lista vazia.

\* **@return** false - se contem elementos.

\*/

**public** **boolean** vazia() {

**return** primeira == ultima;

}

/\*\*

\* Método que insere o elemento no começo.

\* **@param** valorItem - item a ser inserido.

\*/

**public** **void** insereComeco(Object valorItem) {

CCelulaSimples aux = **new** CCelulaSimples();

aux.item = valorItem;

aux.prox = primeira;

primeira = aux;

**if** (primeira.prox == **null**)

ultima = primeira;

}

/\*\*

\* Método que remove o elemento no começo.

\* **@return** valorItem - item a ser removidoo.

\*/

**public** Object removeComeco(){

**if** (primeira != ultima) {

CCelulaSimples aux = primeira;

primeira = aux.prox;

**return** aux.item;

}

**return** **null**;

}

/\*\*

\* Método que insere o elemento no fim.

\* **@param** valorItem - item a ser inserido.

\*/

**public** **void** insereFim(Object valorItem) {

ultima.prox = **new** CCelulaSimples();

ultima.prox.item = valorItem;

ultima = ultima.prox;

}

/\*\*

\* Método que remove o elemento no fim.

\* **@return** valorItem - item a ser removido.

\*/

**public** Object removeFim() {

**if** (primeira != ultima) {

CCelulaSimples aux = primeira;

**while** (aux.prox != ultima)

aux = aux.prox;

CCelulaSimples aux2 = aux.prox;

ultima = aux;

ultima.prox = **null**;

**return** aux2.item;

}

**return** **null**;

}

/\*\*

\* Método que imprime a lista.

\*/

**public** **void** imprime() {

**for** (CCelulaSimples aux = primeira; aux != **null**; aux = aux.prox)

System.***out***.print(aux.item + " ");

System.***out***.println("");

}

/\*\*

\* Método que verifica se um elemento está contido na lista.

\* **@param** elemento - item a ser procurado.

\* **@return** true - lista contem elemento.

\* **@return** false - lista não contem elemento.

\*/

**public** **boolean** contem(Object elemento){

**boolean** contem = **false**;

**for** (CCelulaSimples aux = primeira; aux != **null** && !contem; aux = aux.prox)

contem = elemento.equals(aux.item);

**return** contem;

}

}

\*obs: mudei o nome da célula para CCelulaSimples, pois como CCelula conflitava com a classe CCelula que estava na pasta conjunta em que fiz os exercícios.

Teste e resultado:

