CMP1054 - Estrutura de Dados I

Listas Lineares em vetores

Java

Listas Lineares

- Uma das formas mais simples de interligar os elementos de um conjunto.
- Estrutura em que as operações inserir, retirar e localizar são definidas.
  - Itens podem ser acessados, inseridos ou retirados de uma lista.
- Duas listas podem ser concatenadas para formar uma lista única, ou uma pode ser partida em duas ou mais listas.
- Podem crescer ou diminuir de tamanho durante a execução de um programa, de acordo com a demanda.

Listas Lineares

- Aplicabilidade:
  - Com uso de alocação dinâmica de memoria (encadeamento de objetos)
    - Adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.

Definição de Listas Lineares

- Seqüência de zero ou mais itens $x_1 , x_2 , \cdots , x_n$, na qual x_i é de um determinado tipo e n representa o tamanho da lista linear.
- Sua principal propriedade estrutural envolve as posições relativas dos itens em uma dimensão.
  - Assumindo $n \geq 1$, x_1 é o primeiro item da lista e x_n é o último item da lista.
  - x_i precede x_{i+1} para $i = 1, 2, \cdots , n-1$
  - x_i sucede x_{i-1} para $i = 2, 3, \cdots , n$
  - O elemento x_i é dito estar na ***i*-ésima** posição da lista.

TAD Listas Lineares

- O conjunto de operações a ser definido depende de cada aplicação.
- Um conjunto de operações necessário a uma maioria de aplicações é:
  - Criar uma lista linear vazia.
  - Inserir um novo item no final ou imediatamente após o último item.
  - Retirar um item dado uma chave de busca.
  - Combinar duas ou mais listas lineares em uma lista única.
  - Partir uma lista linear em duas ou mais listas.
  - Fazer uma cópia da lista linear.
  - Localizar um item, dado uma chave de busca, para examinar e/ou alterar o conteúdo de seus componentes.
  - Pesquisar a ocorrência de um item com um valor particular em algum componente.

Implementações de Listas Lineares

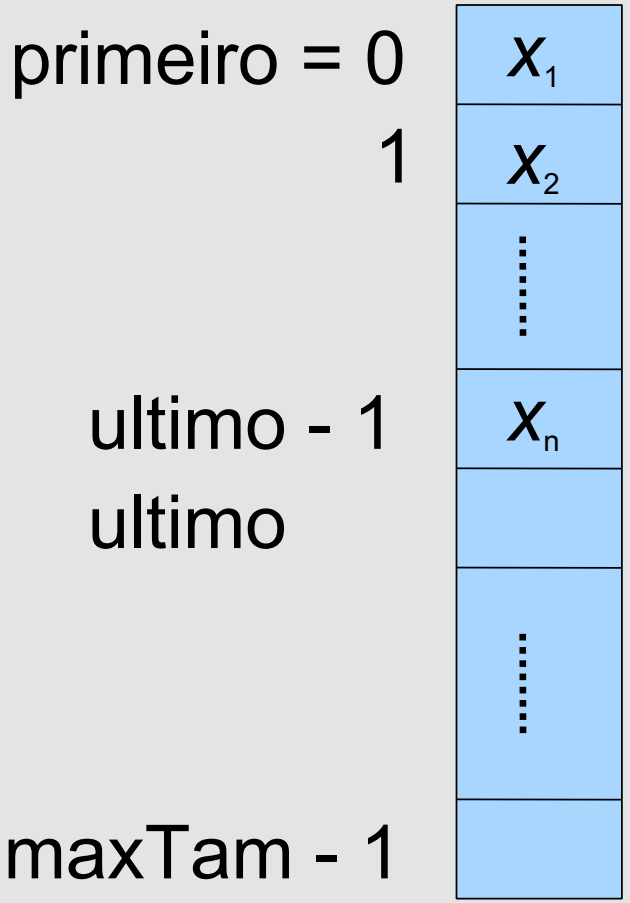
- Várias estruturas de dados podem ser usadas para representar listas lineares, cada uma com vantagens e desvantagens particulares.
- As duas representações mais utilizadas são as implementações por meio de:
  - arranjos
  - estruturas auto-referenciadas (encadeamento de objetos)
- Outra ----> Cursores

Exemplo de conjunto de operações (será usado no curso)

1. **Lista (*MaxTam*)**. Cria uma lista vazia.
2. **insereFinal(Item x)**. Insere ***x*** após o último item da lista.
3. **retira(Item *x*)**. Retorna o item ***x*** presente na lista, retirando-o da lista e deslocando os itens a partir da posição dele no vetor para as posições anteriores.
4. **listaVazia()**. Esta função retorna ***true*** se lista vazia; senão retorna ***false***.
5. **listaCheia()**. Esta função retorna ***true*** se lista cheia; senão retorna ***false***.
6. **getLista()**. concatena os itens da lista, na ordem de ocorrência, numa string.
7. **pesquisa(*Item x*)**. Verifica se um determinado item (***chave***) está na lista e retorna a sua posição.

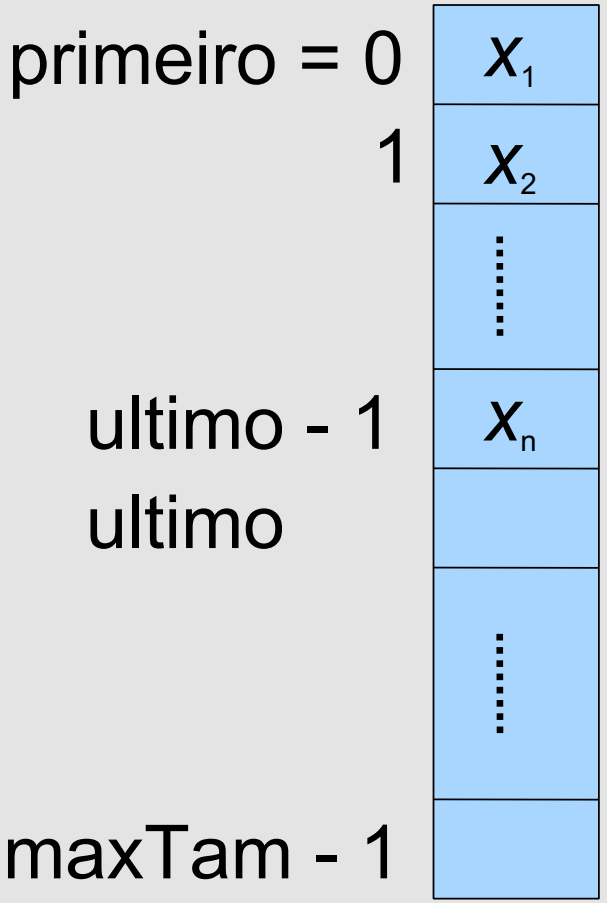
Implementação de Listas por meio de estruturas Arranjos

- Os itens da lista são armazenados em posições contíguas de memória.
- A lista pode ser percorrida em qualquer direção.
- A inserção de um novo item pode ser realizada após o ***último*** item com custo constante.
- A inserção de um novo item no meio da lista requer um deslocamento de todos os itens localizados após o ponto de inserção.
- Retirar um item do início (ou do meio) da lista requer um deslocamento de itens para preencher o espaço deixado vazio.



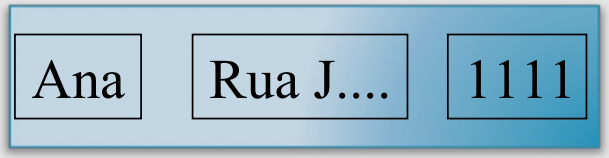
Implementação de Listas por meio de estruturas Arranjos

- Os itens são armazenados em um arranjo de tamanho suficiente para armazenar a lista.
- O campo ***ultimo*** aponta para a posição seguinte a do último elemento da lista.
- O ***i-ésimo*** item da lista está armazenado na ***i-ésima-1*** posição do arranjo,
  - $0 \leq i < ultimo$.
- A constante ***maxTam*** define o tamanho máximo permitido para a lista.



Item

- A lista é composta de Item (Veja um exemplo a seguir).
  - classe Contato (Item)
    - Atributos
      - nome (string)
      - endereço (string)
      - cpf (long int) --> atributo chave
    - Métodos
      - getNome, getEndereco, getCPF
      - setNome, setEndereco, setCPF
      - getContato
      - Contato

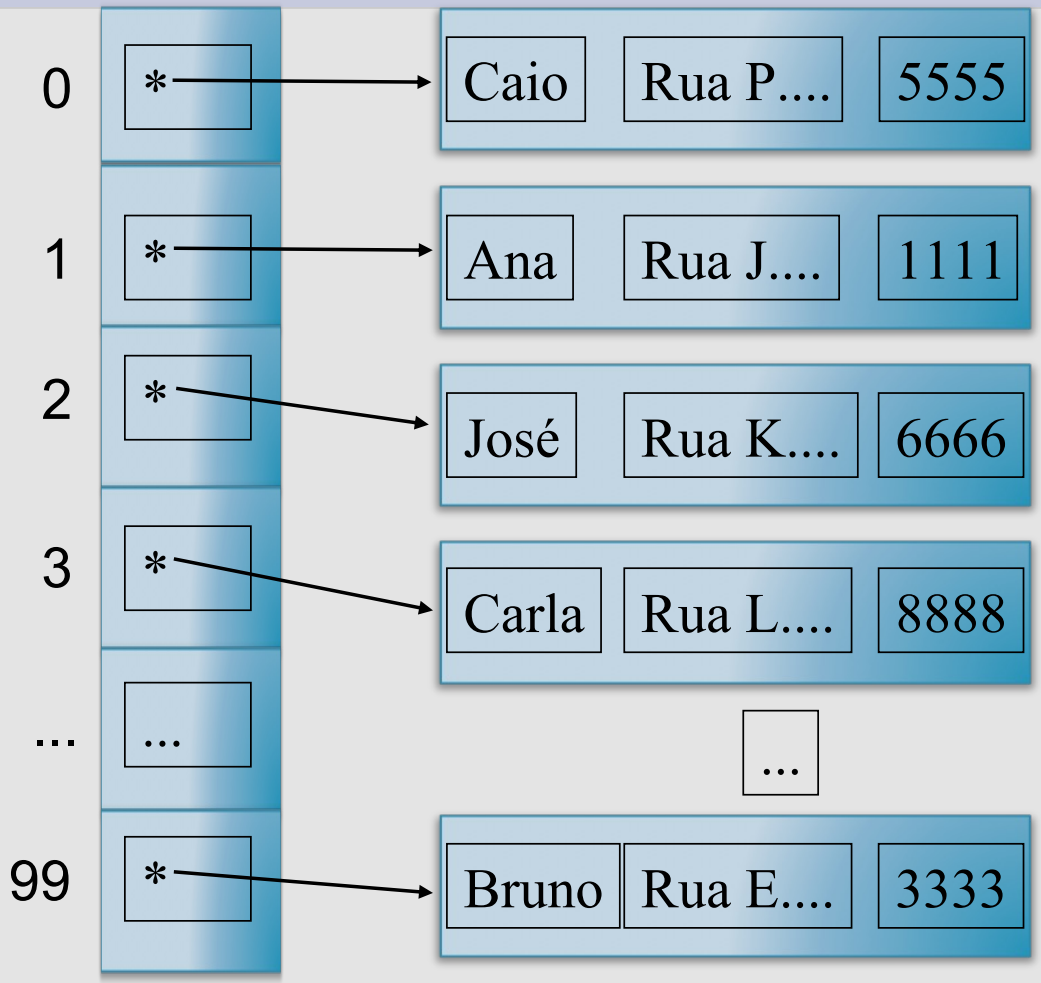


Lista

- A classe Lista
  - Atributos
    - ultimo (int)
    - maxTam (int)
    - vetor (Contato [])
  - Métodos
    - getMaxTam
    - setMaxTam
    - Lista
    - listaVazia
    - listaCheia
    - insereFinal
    - pesquisa
    - retira
    - getLista

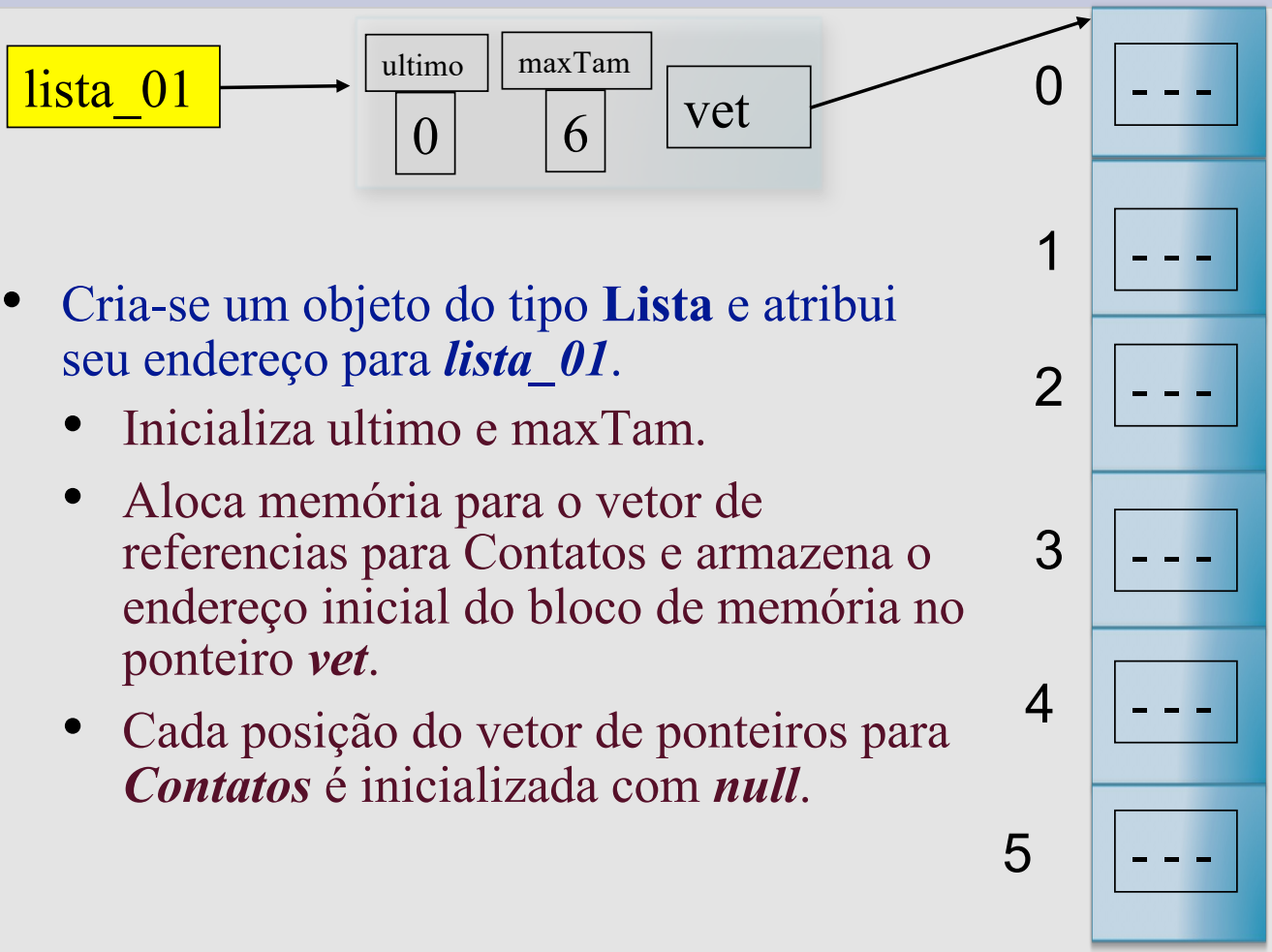
Vetor de referencias para objetos

- classe Contato (Item)
  - Atributos
    - Nome (string)
    - Endereço (string)
    - Cpf (long long int)



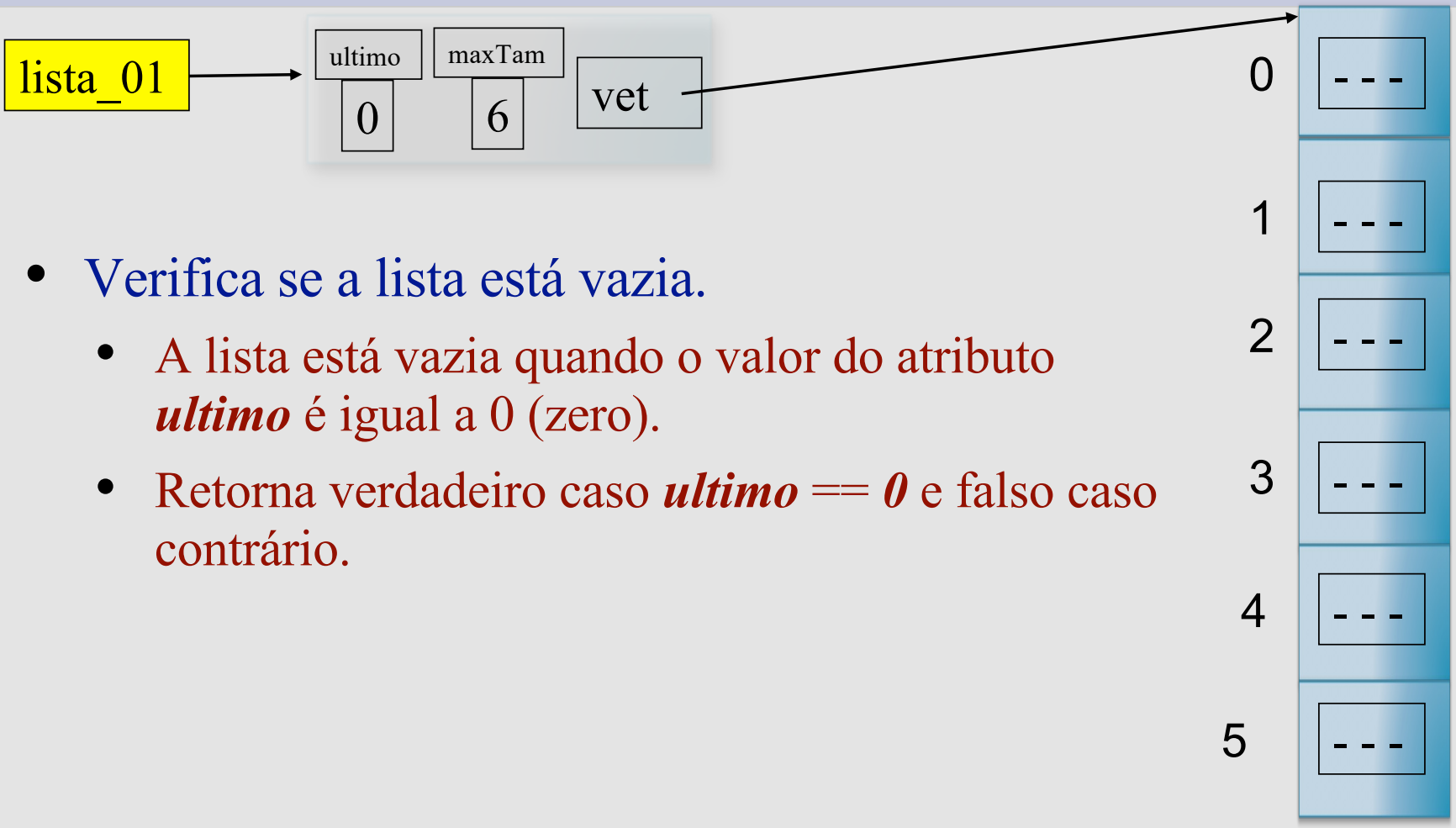
Lista() + setMaxTam(6)

- Construtor + setMaxTam(6).
  - Cria uma lista vazia.



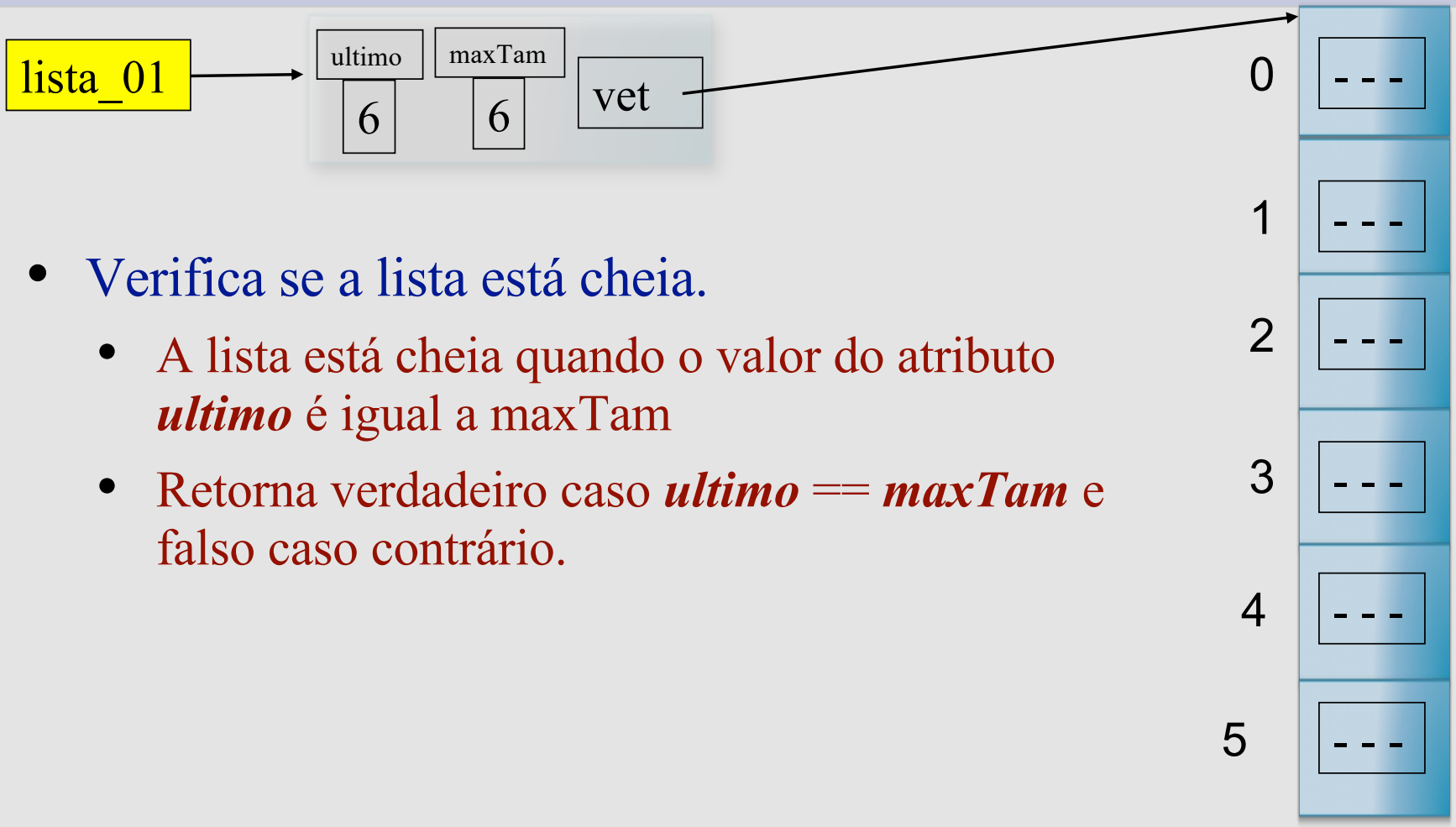
- Cria-se um objeto do tipo **Lista** e atribui seu endereço para ***lista_01***.
  - Inicializa ultimo e maxTam.
  - Aloca memória para o vetor de referencias para Contatos e armazena o endereço inicial do bloco de memória no ponteiro ***vet***.
  - Cada posição do vetor de ponteiros para ***Contatos*** é inicializada com ***null***.

listaVazia()



- Verifica se a lista está vazia.
  - A lista está vazia quando o valor do atributo ***ultimo*** é igual a 0 (zero).
  - Retorna verdadeiro caso ***ultimo*** == ***0*** e falso caso contrário.

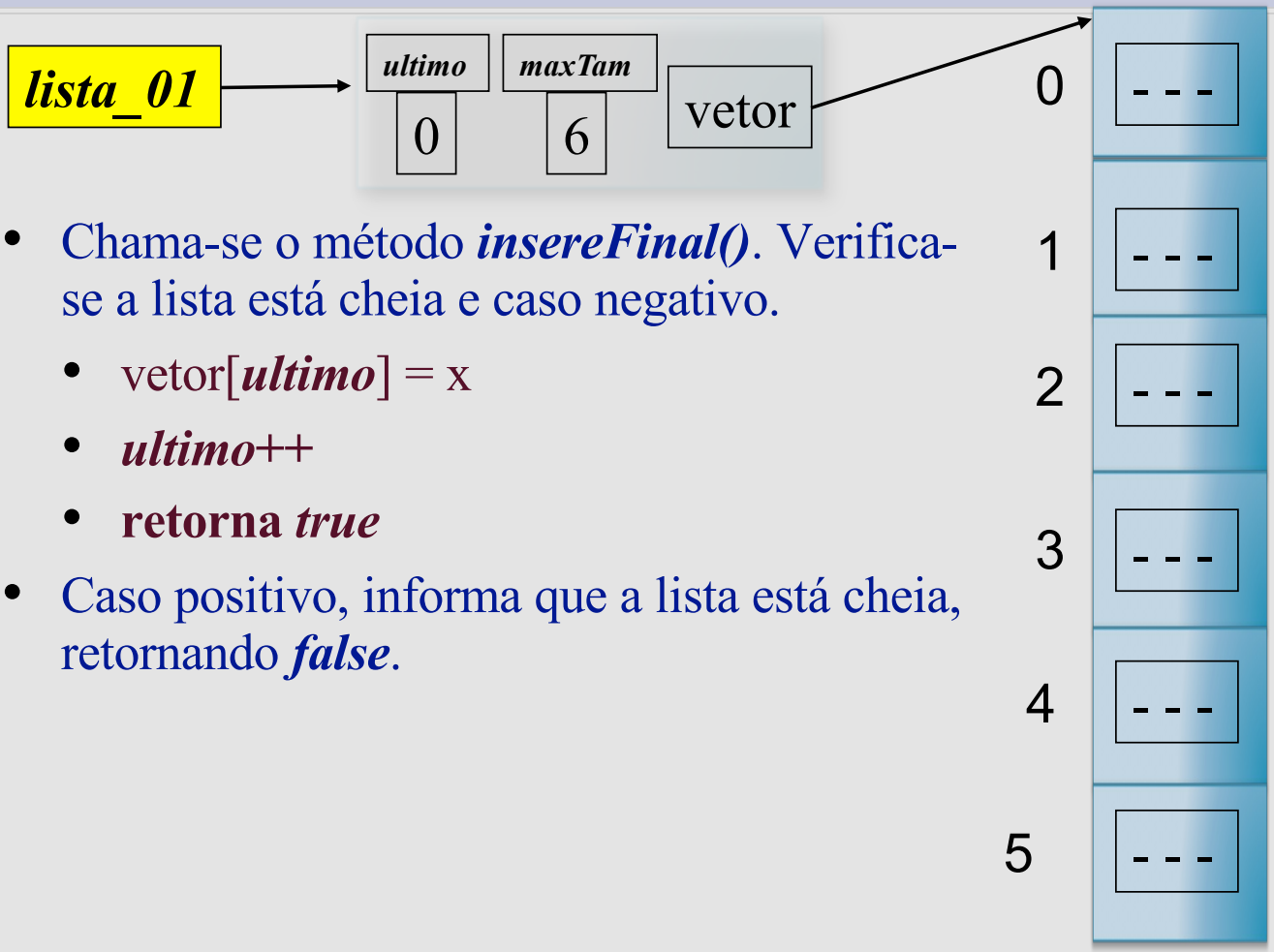
listaCheia()



- Verifica se a lista está cheia.
  - A lista está cheia quando o valor do atributo ***ultimo*** é igual a maxTam
  - Retorna verdadeiro caso ***ultimo*** == ***maxTam*** e falso caso contrário.

insereFinal(Contato x)

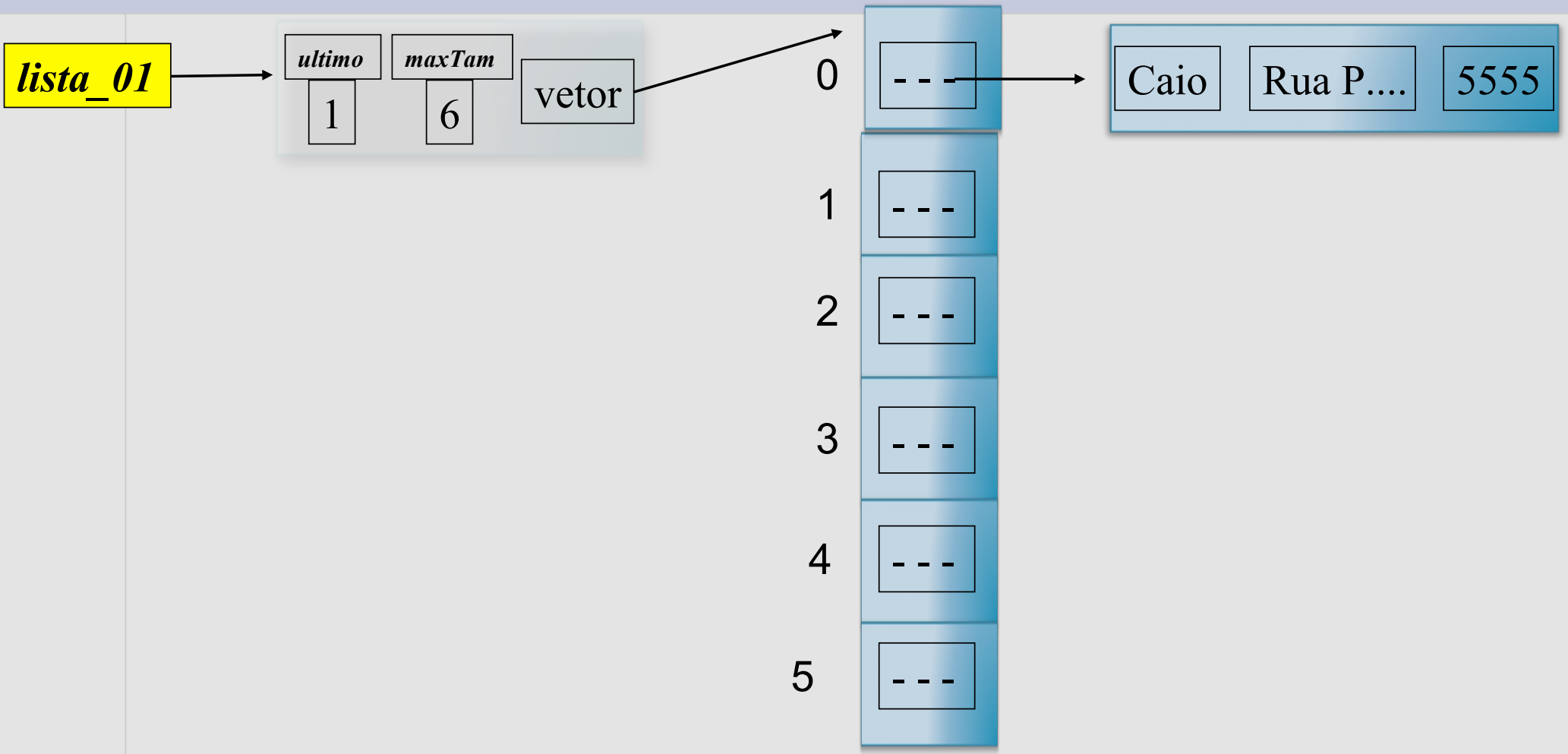
- Insere um item no final da lista.



- Chama-se o método ***insereFinal()***. Verifica-se a lista está cheia e caso negativo.
  - vetor[***ultimo***] = x
  - ***ultimo***++
  - **retorna *true***
- Caso positivo, informa que a lista está cheia, retornando ***false***.

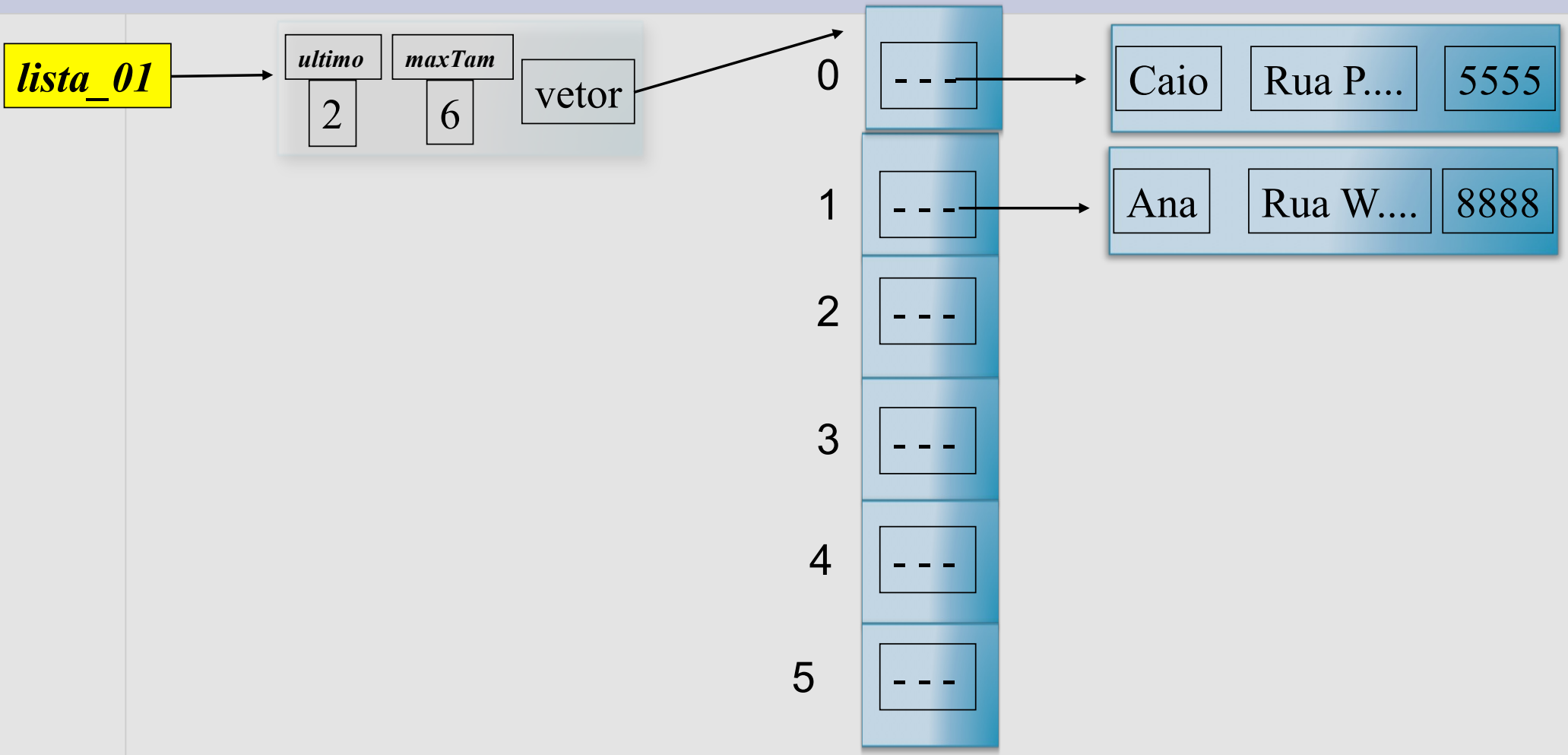
insereFinal(Contato x)

- Insere um item no final da lista.



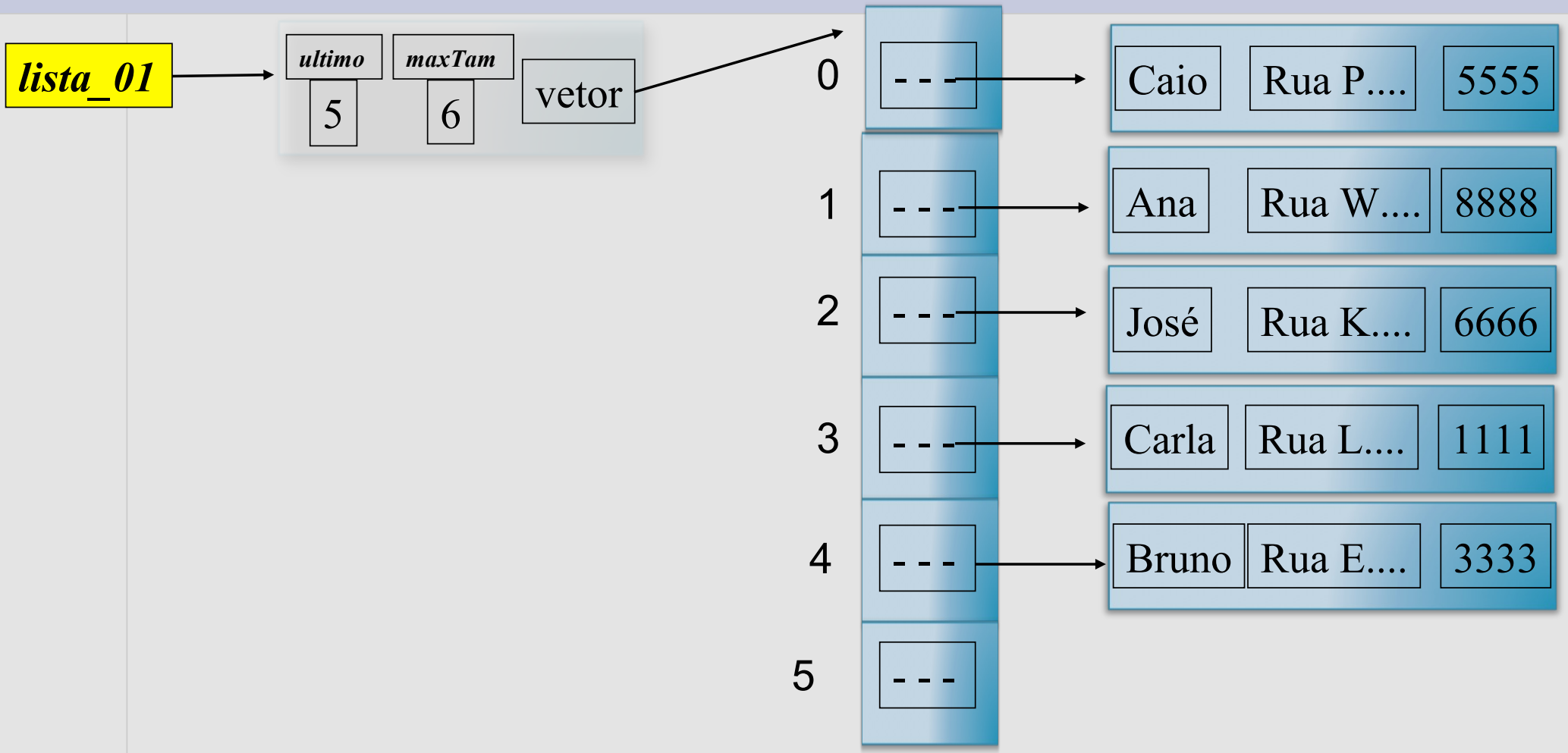
insereFinal(Contato x)

- Insere um item no final da lista.



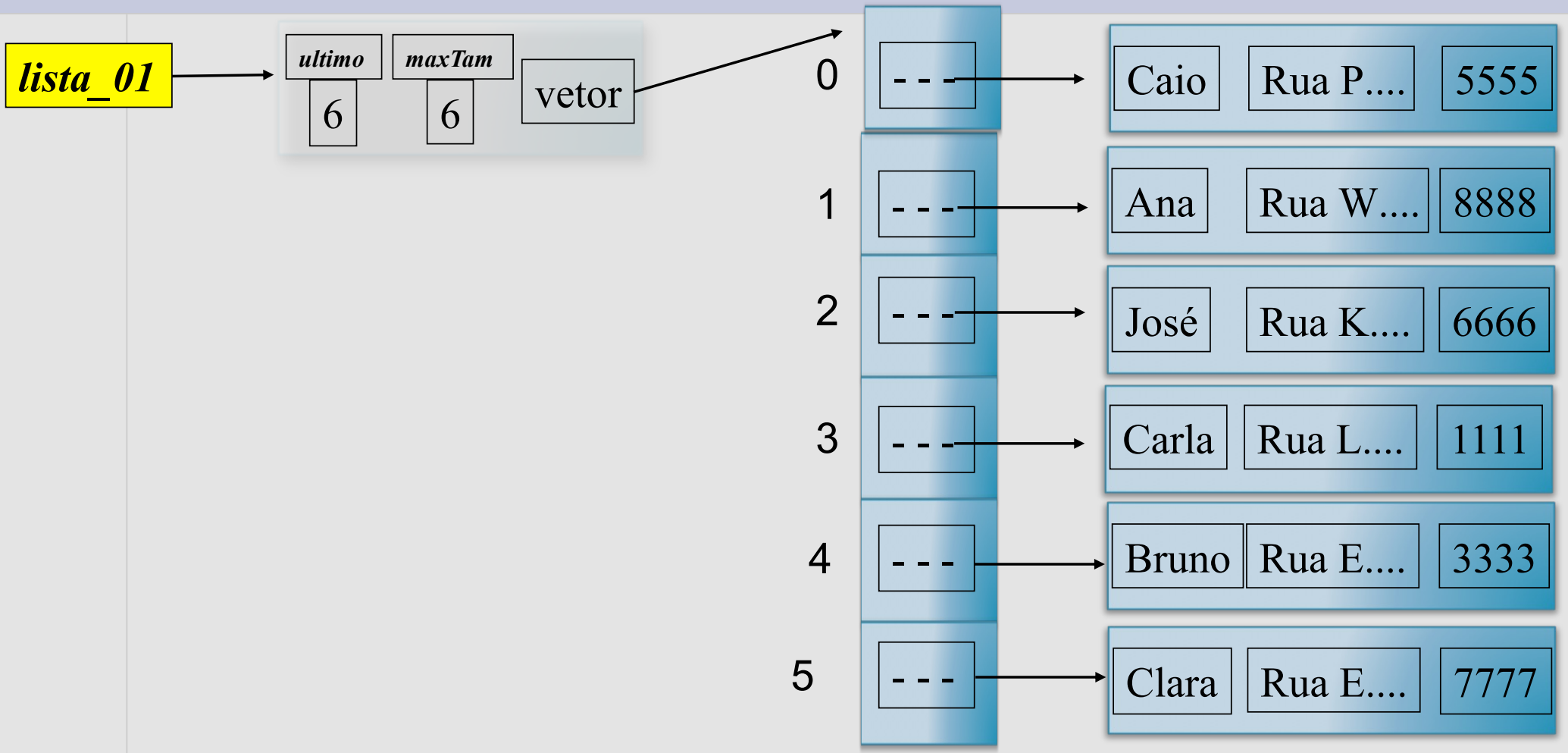
Lista após 5 inserções.

- Insere um item no final da lista.

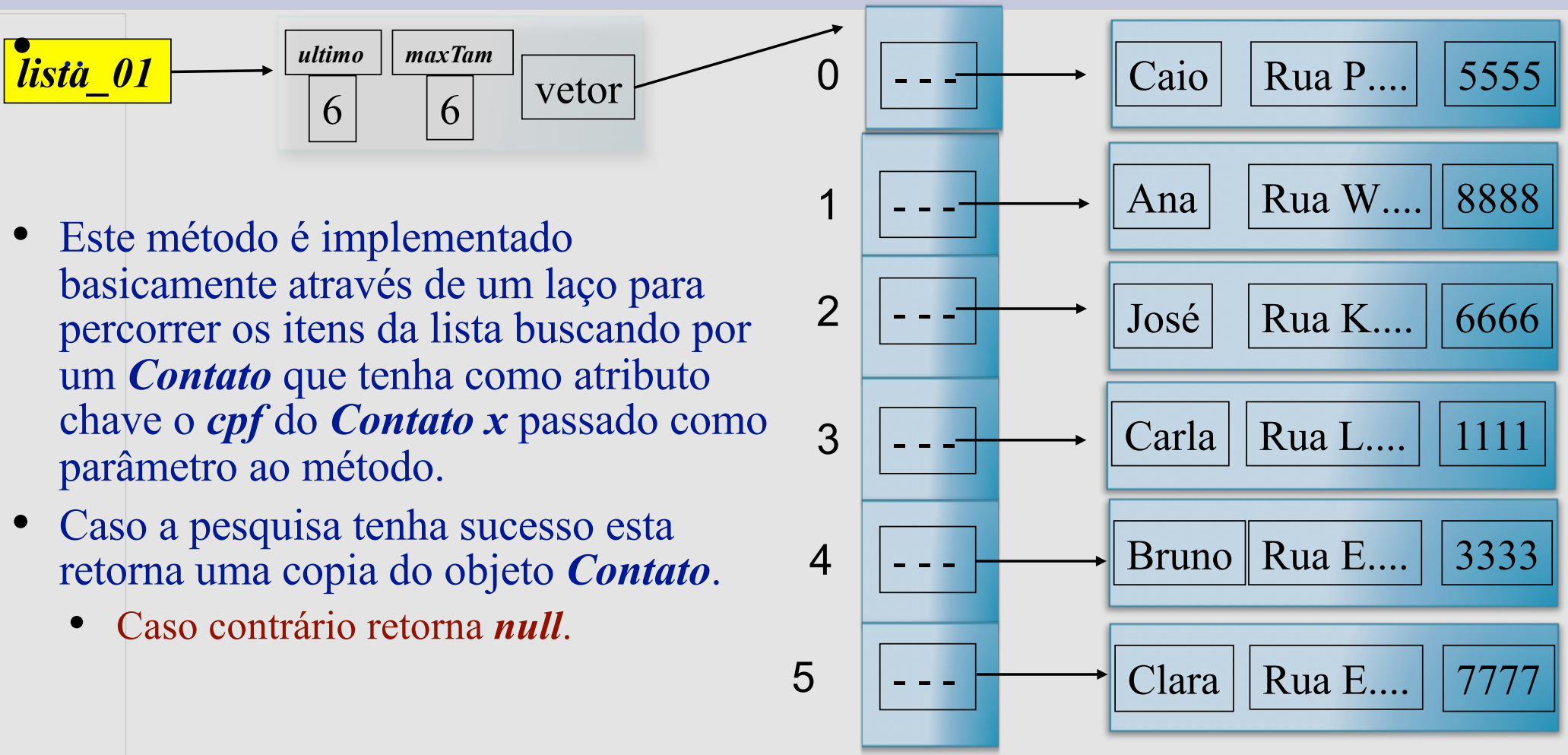


Lista após 6 inserções.

- Insere um item no final da lista.

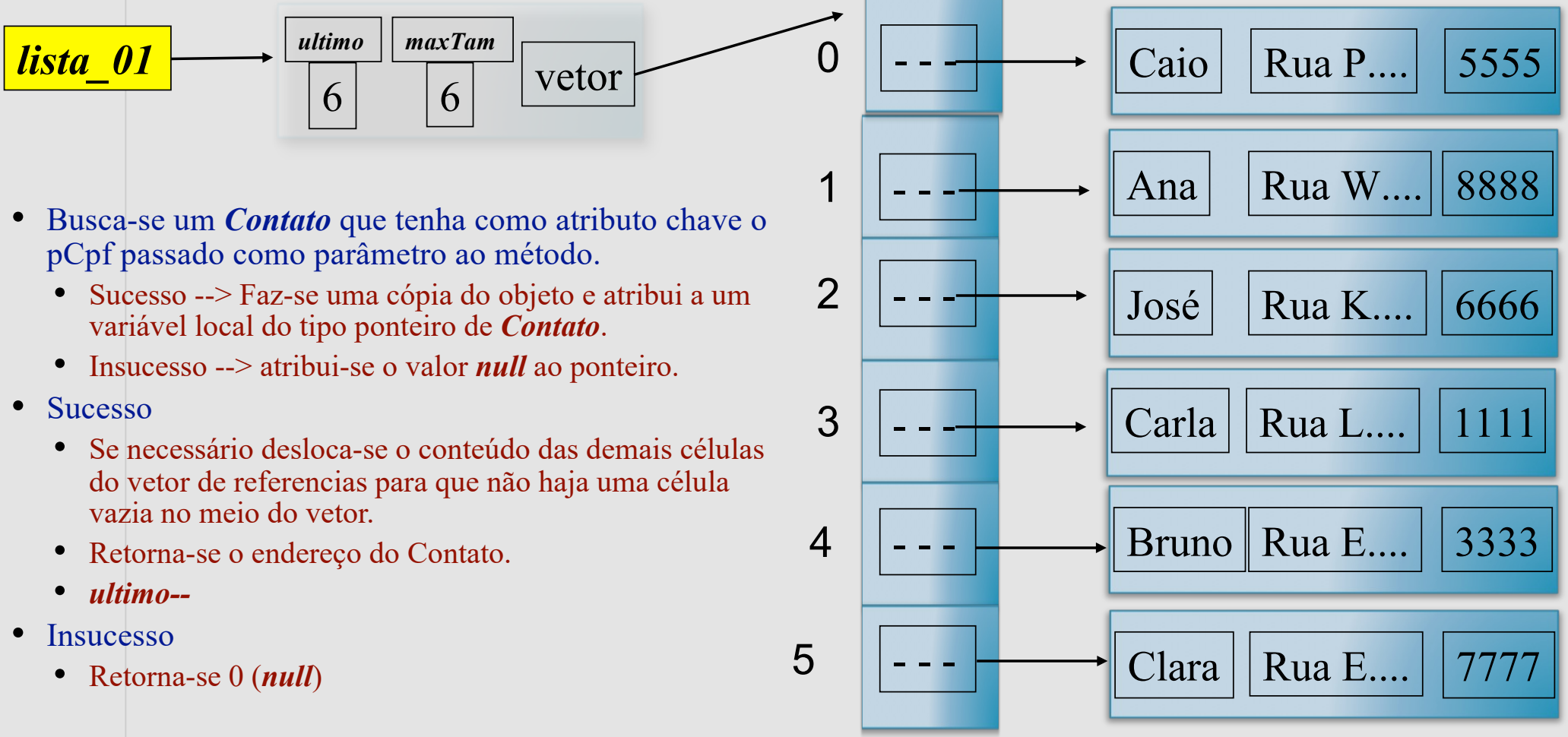


pesquisa(Contato x)
pCpf == 1111



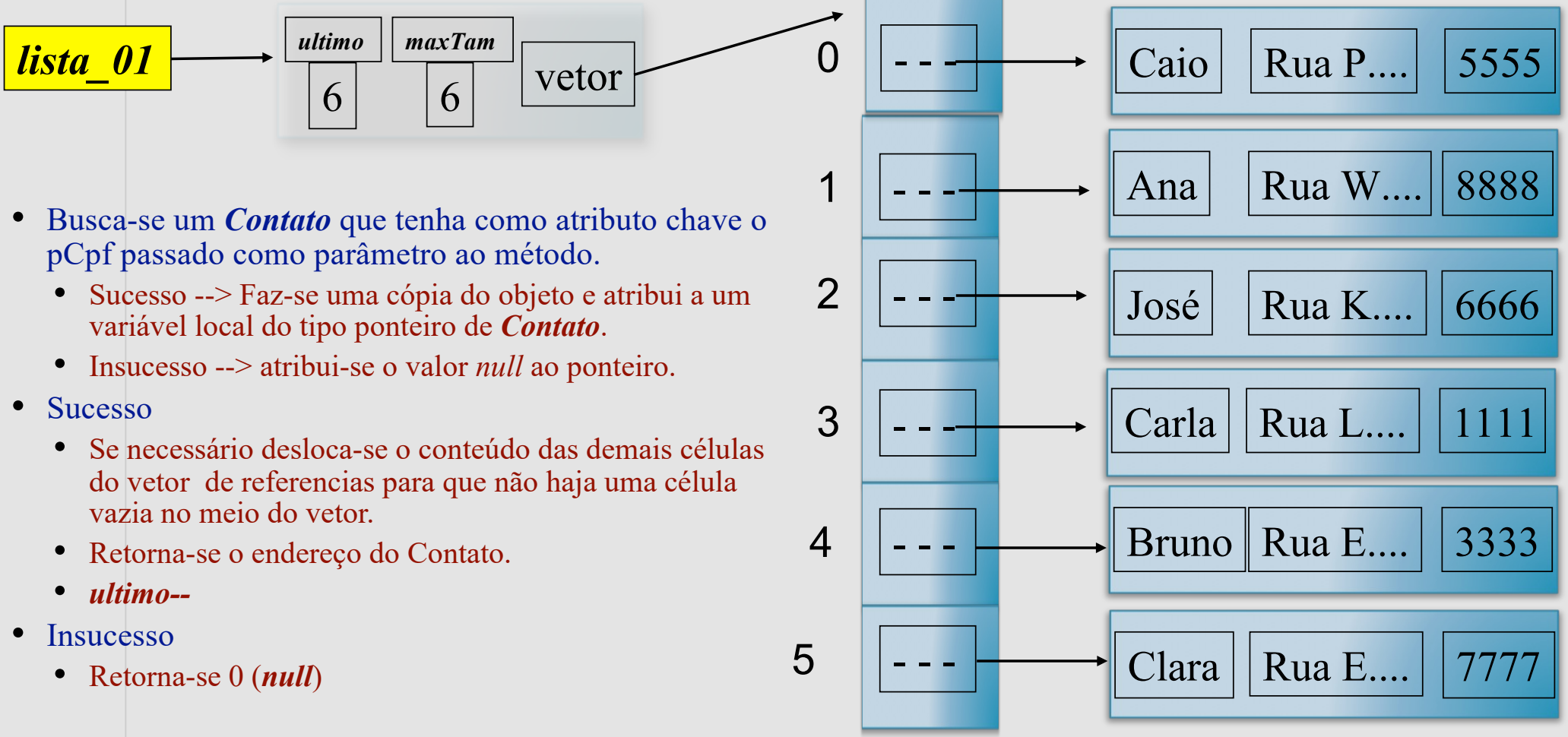
- Este método é implementado basicamente através de um laço para percorrer os itens da lista buscando por um ***Contato*** que tenha como atributo chave o ***cpf*** do ***Contato x*** passado como parâmetro ao método.
- Caso a pesquisa tenha sucesso esta retorna uma copia do objeto ***Contato***.
  - Caso contrário retorna ***null***.

retira(Contato x)
pCpf == 6666



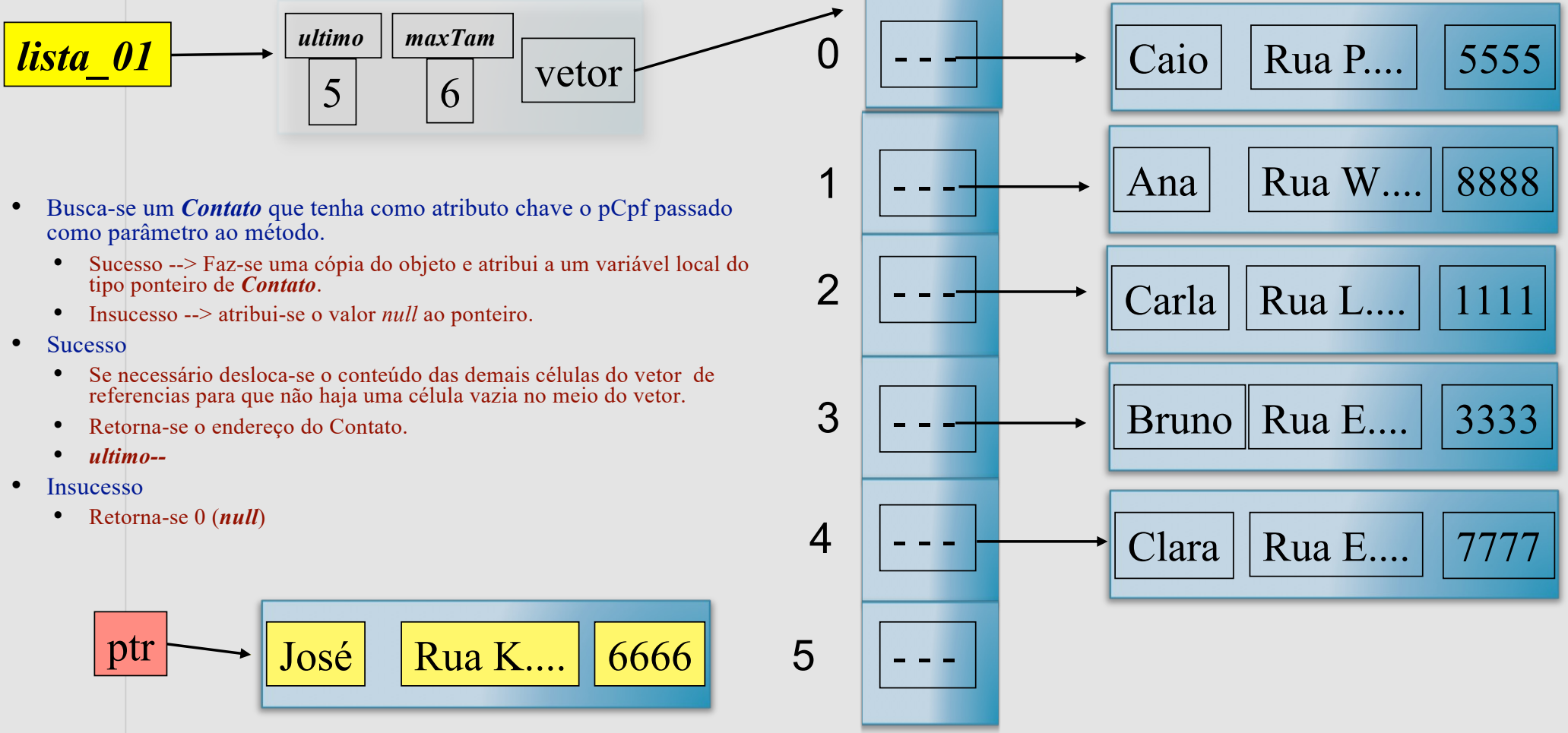
- Busca-se um ***Contato*** que tenha como atributo chave o pCpf passado como parâmetro ao método.
  - Sucesso --> Faz-se uma cópia do objeto e atribui a um variável local do tipo ponteiro de ***Contato***.
  - Insucesso --> atribui-se o valor ***null*** ao ponteiro.
- Sucesso
  - Se necessário desloca-se o conteúdo das demais células do vetor de referencias para que não haja uma célula vazia no meio do vetor.
  - Retorna-se o endereço do Contato.
  - ***ultimo--***
- Insucesso
  - Retorna-se 0 (***null***)

retira(Contato x)
pCpf == 6666



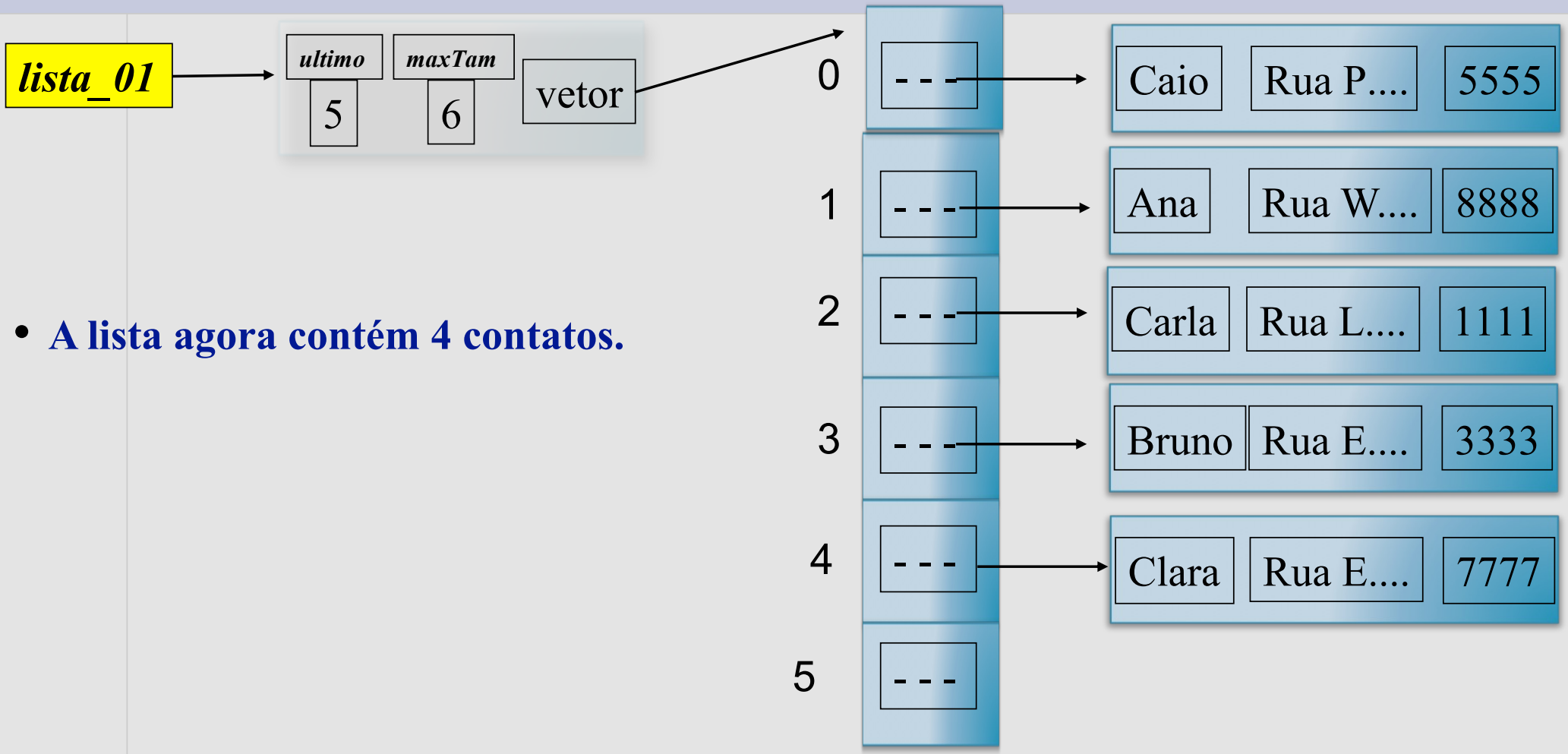
- Busca-se um ***Contato*** que tenha como atributo chave o pCpf passado como parâmetro ao método.
  - Sucesso --> Faz-se uma cópia do objeto e atribui a um variável local do tipo ponteiro de ***Contato***.
  - Insucesso --> atribui-se o valor *null* ao ponteiro.
- Sucesso
  - Se necessário desloca-se o conteúdo das demais células do vetor de referencias para que não haja uma célula vazia no meio do vetor.
  - Retorna-se o endereço do Contato.
  - ***ultimo--***
- Insucesso
  - Retorna-se 0 (***null***)

retira(Contato x)
pCpf == 6666



- Busca-se um ***Contato*** que tenha como atributo chave o pCpf passado como parâmetro ao método.
  - Sucesso --> Faz-se uma cópia do objeto e atribui a um variável local do tipo ponteiro de ***Contato***.
  - Insucesso --> atribui-se o valor *null* ao ponteiro.
- Sucesso
  - Se necessário desloca-se o conteúdo das demais células do vetor de referencias para que não haja uma célula vazia no meio do vetor.
  - Retorna-se o endereço do Contato.
  - ***ultimo--***
- Insucesso
  - Retorna-se 0 (***null***)

lista após retirada do Contato com cpf 6666



- **A lista agora contém 4 contatos.**

Clase TestaLista (Base)

- Métodos
  - Contato criarContato()
  - void main()
    - Criar a lista
    - Alocar memória para a lista (tamanho do vetor)
    - Menu de escolhas (colocar na lista, pesquisar, retirar da lista, mostrar lista)

UML

Classe Contato
- nome: String
- enderco: String
- cpf: long
+ Contato(String, String, long)
+ getContato(): String

Classe TestaLista
- objLista: Lista
+ criarContato(): Contato
+ main(): void

Classe Lista
- ultimo: int
- maxTam: int (>=2)
- vetor: Contato[]
+ setMaxTam(int): bool
+ getMaxTam(): int
+ Lista()
+ listaVazia(): bool
+ listaCheia(): bool
+ insereFinal(Contato): bool
+ retira(Contato): Contato
+ pesquisa(Contato): Contato
+ getLista(): String