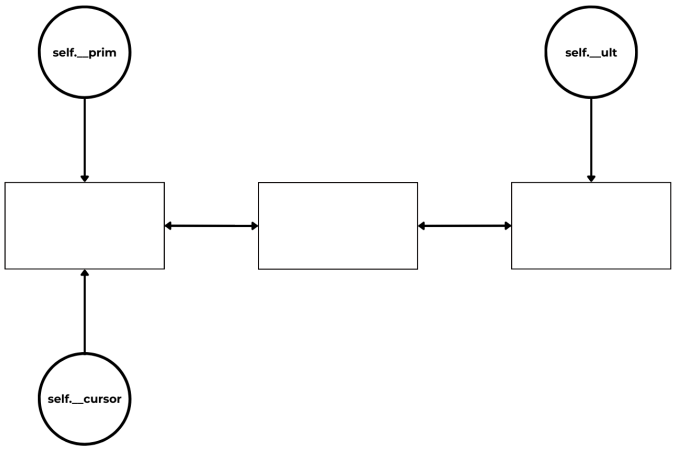
<u>Relatório do Trabalho 1</u> - Lista Duplamente Encadeada

Alunos: Eduardo Boçon e Jiliard Peifer.

<u>Desenvolvendo a estrutura da nossa lista duplamente encadeada</u>

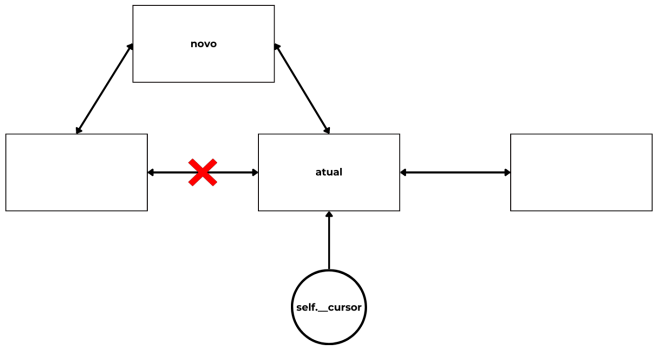


- Desenho para ajudar no raciocínio dos ponteiros na lista.

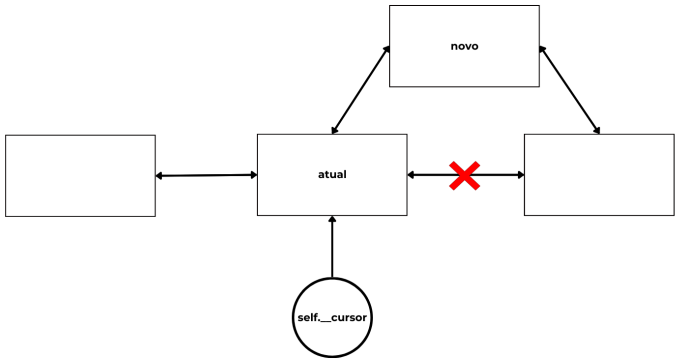
<u>MÉTODOS</u>

- **(devolve elemento) acessarAtual()**
  - Retorna o dado do atualmente apontado pelo cursor.

- **(não devolve nada) InserirAntesDoAtual(el)**
  - Cria um objeto novo.
  - Se o anterior do atual não for vazio, torna o anterior do atual o anterior do novo. Essa verificação acontece para casos onde o atual seja o primeiro.
  - Define o atual como o próximo do novo. Cortando as antigas conexões existentes.
  - Soma um para a quantidade de elementos na lista.
  - Aponta o cursor para o novo objeto.

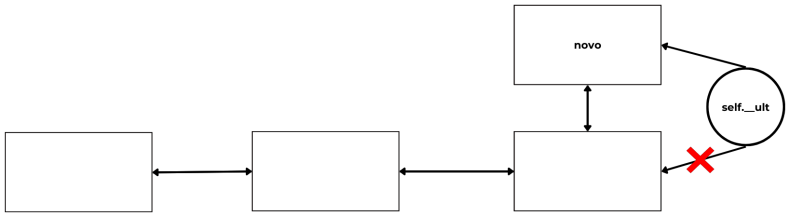


- **(não devolve nada) InserirDepoisDoAtual(el)**
    - Cria um objeto novo.
    - Se o próximo do atual não for vazio, torna o próximo do atual o próximo do novo. Essa verificação acontece para casos onde o atual seja o último.
    - Define o atual como o anterior do novo. Cortando as antigas conexões existentes.
    - Soma um para a quantidade de elementos na lista.
    - Aponta o cursor para o novo objeto.

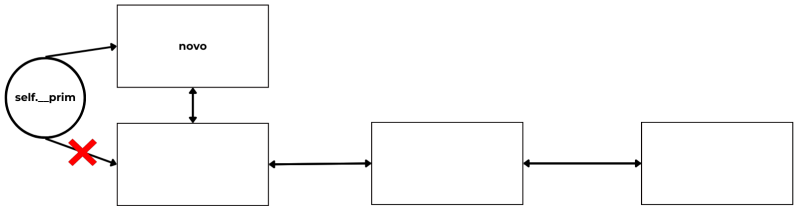


- **(não devolve nada) inserirComoUltimo(el)**
    - Cria um objeto novo, se o último estiver vazio (ou seja, lista vazia), o novo será o primeiro, e ele estará sendo apontado pelo cursor.
    - Caso exista um último, o novo será definido como próximo do último, ou seja, o último.
    - O novo sempre será apontado como último.
    - Soma um para a quantidade de elementos na lista.

- Aponta o cursor para o novo objeto.

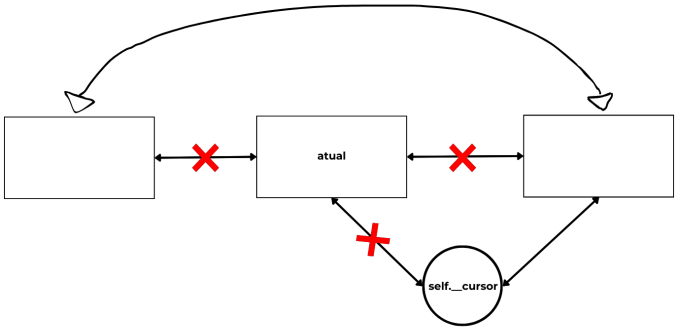


- **(não devolve nada) inserirComoPrimeiro(el)**
  - Cria um objeto novo, se o primeiro estiver vazio (ou seja, lista vazia), o novo será o último, e ele estará sendo apontado pelo cursor.
  - Caso exista um primeiro, o próximo do novo será o primeiro, ou seja, o novo se torna o primeiro.
  - Bota o novo primeiro como anterior do antigo primeiro.
  - O novo sempre será apontado como primeiro.
  - Soma um para a quantidade de elementos na lista.
  - Aponta o cursor para o novo objeto.

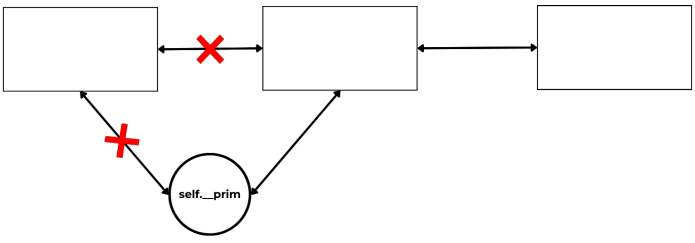


- **(não devolve nada) inserirNaPosicao(el, pos)**
  - O cursor vai para o primeiro, e avança ‘pos’ - 1 posições, tornando-o atual.
  - Insere o elemento ‘el’ antes do atual.
  - Soma um para a quantidade de elementos na lista.

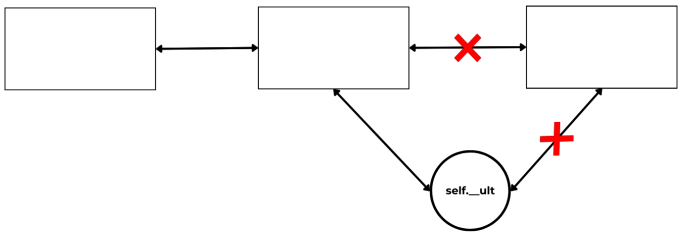
- **(não devolve nada) ExcluirAtual()**
    - Pega o próximo do atual, e define ele como próximo do anterior do atual.
    - Pega o anterior do atual, e define ele como anterior do próximo do atual.
    - Avança o cursor uma posição, ou seja, aponta pro próximo.
    - Caso o atual seja o primeiro ou o último elemento, apenas executa o devido método para excluí-lo.
    - Subtrai um para a quantidade de elementos na lista.



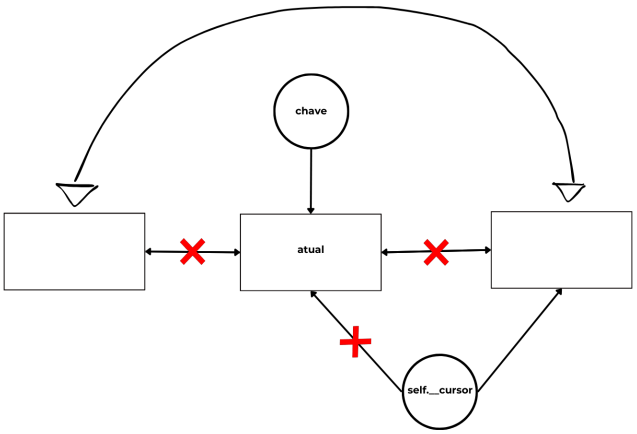
- **(não devolve nada) ExcluirPrim()**
    - Se vazio, retorna uma exceção.
    - Se não for vazio, o próximo do primeiro torna-se o primeiro.
    - Define o anterior desse novo primeiro como vazio, excluindo assim o antigo primeiro.
    - Subtrai um para a quantidade de elementos na lista.
    - Aponta o cursor para o novo primeiro.



- **(não devolve nada) ExcluirUlt()**
    - Se vazio, retorna uma exceção.
    - Se não for vazio, o anterior do último torna-se o último.
    - Define o próximo desse novo último como vazio, excluindo assim o antigo último.
    - Subtrai um para a quantidade de elementos na lista.
    - Aponta o cursor para o novo último.



- **(não devolve nada) ExcluirElemento(el)**
    - Direciona o cursor para o primeiro.
    - Pega o mesmo objeto ‘el’, e define ele como o próximo do último, criando um ‘fake’ no final da lista.
    - Avança continuamente de um em um, até achar o objeto ‘el’, seja ele o procurado esperado ou o ‘fake’ adicionado.
    - Se o cursor estiver apontando para o próximo do último, significa que o ‘el’ encontrado é ‘fake’, portanto ‘el’ não existe na lista e é lançada uma exceção.
    - Se o cursor não estiver apontando para o próximo do último, significa que o ‘el’ original foi encontrado, portanto será excluído e subtraído um para a quantidade de elementos na lista .
    - Define o próximo do último como vazio, excluindo o objeto ‘fake’ da lista.



- **(não devolve nada) ExcluirDaPos(pos)**
    - Vai para o primeiro e avança ‘pos’ - 1 posições.
    - ‘pos’ - 1 → Por exemplo, se começamos na posição 1 (primeiro) e queremos chegar na posição 4, é necessário avançar 3 posições de 1 até 4, logo, ‘pos’ - 1.
    - Exclui o atualmente apontado pelo cursor, que está na posição ‘pos’.
    - Subtrai um para a quantidade de elementos na lista.

- **(boolean) Buscar(el)**
    - Cria um objeto novo.
    - Direciona o cursor para o primeiro.
    - Pega o mesmo objeto ‘el’, e define ele como o próximo do último, criando um ‘fake’ no final da lista.
    - Avança continuamente de um em um, até achar o objeto ‘el’, seja ele o procurado esperado ou o ‘fake’ adicionado.
    - Se o cursor estiver apontando para o próximo do último, significa que o ‘el’ encontrado é ‘fake’, portanto ‘el’ não existe na lista, ‘busca’ é definido como False, pois ‘el’ não foi encontrado.
    - Se o cursor não estiver apontando para o próximo do último, significa que o ‘el’ original foi encontrado, portanto ‘busca’ é definido como True, pois ‘el’ foi encontrado.
    - Define o próximo do último como vazio, excluindo o objeto ‘fake’ da lista, e retorna ‘busca’ com seu devido valor boolean.

- **__avancarKPosicoes(k)**
    - Aponta o cursor para o próximo do atual ‘k’ vezes.
    - Caso o cursor chegue no último elemento, o método acaba e o cursor permanece no último, independente se não foram avançadas ‘k’ posições.

- **__retrocederKPosicoes(k)**

    - Aponta o cursor para o anterior do atual ‘k’ vezes.
    - Caso o cursor chegue no primeiro elemento, o método acaba e o cursor permanece no primeiro, independente se não foram retrocedidas ‘k’ posições.

- **__irParaOPrimeiro()**
    - Verifica, continuamente, se o anterior do atual é vazio.
    - Enquanto não for, o cursor irá apontar para o anterior do atual, até chegar no primeiro (que não possui anterior) e parar.

- **__irParaOUltimo()**
    - Verifica, continuamente, se o próximo do atual é vazio.
    - Enquanto não for, o cursor irá apontar para o próximo do atual, até chegar no último (que não possui próximo) e parar.

- **(boolean) __Vazio()**
    - Se o primeiro for vazio, retorna True, ou seja, não existe um primeiro elemento, portanto a lista está vazia.
    - Se existir um primeiro elemento, retorna False, ou seja, existe um primeiro elemento, portanto tem elemento na lista e ela não está vazia.

- **(boolean) __Cheio()**
    - Se a quantidade de elementos na lista for maior ou igual ao tamanho definido da lista, retorna True, ou seja, a lista está cheia.
    - Se a quantidade de elementos na lista for menor que o tamanho definido da lista, retorna False, ou seja, a lista não está cheia.

- **(INT) posiçãoDe(el)**
    - (Informa a posição sequencial do elemento com aquela chave, contado desde o primeiro.)
    - Cria um objeto novo.
    - Direciona o cursor para o primeiro.
    - Pega o mesmo objeto ‘el’, e define ele como o próximo do último, criando um ‘fake’ no final da lista.
    - Define um ‘contador’ com um inteiro para acompanhar o número da posição em que o cursor está (começa como ‘1’, pois o cursor começa na posição 1, o primeiro).
    - Avança continuamente de um em um, até achar o objeto ‘el’, seja ele o procurado esperado ou o ‘fake’ adicionado.
    - Enquanto avança posições, também irá incrementar ‘+1’ no ‘contador’ criado, para cada uma das posições avançadas.
    - Se o cursor estiver apontando para o próximo do último, significa que o ‘el’ encontrado é ‘fake’, portanto ‘el’ não existe na lista, exclui o objeto ‘fake’ da lista e é lançada uma exceção.

- Se o cursor não estiver apontando para o próximo do último, significa que o ‘el’ original foi encontrado, portanto o objeto ‘fake’ é excluído e será retornado o inteiro contido em ‘contador’, com a respectiva posição de ‘el’ na lista.