Universidade Federal do Pará

Campus Universitário de Castanhal
Faculdade de Computação
Curso Bacharelado em Sistemas de Informação

Estruturas de Dados
Listas

Profª Penha Abi Harb
mpenha@ufpa.br

Listas

- Algumas aplicações requerem a ordem relativa dos dados, de acordo com algum critério.
- Exemplo:
  - ordem de chegada;
  - relações menor-maior;
  - outros.
- ESTRUTURAS DE LISTAS são aquelas que possibilitam o armazenamento de dados numa ordem dependente de alguma propriedade.

Listas

- Forma simples de interligar os elementos de um conjunto.
- Agrupa informações referentes a um conjunto de elementos que se relacionam entre si de alguma forma.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.

Listas

- Uma **lista** é uma forma de organização através da enumeração de dados para melhor visualização da informação.
- Exemplos:
  - Agenda ou Lista Telefônica;
  - Lista de Compras;
  - Bibliografia de uma Disciplina;
  - Ordem de chegada em um consultório, etc.

Listas:Filas,Pilhas

- Lista de compras
  - Arroz
  - Feijão
  - Macarrão
  - Tomate
- Fila de banco

- Pilhas de Livros

FIFO (first in, first out)

LIFO (last in, first out)

Representação de listas

- Há dois tipos possíveis de representação de listas lineares:
- Por Contiguidade (LISTA ESTÁTICA):
  - Garantia da precedência dos elementos pela contiguidade física na memória usando arranjos;
- Por Encadeamento (LISTA DINÂMICA)
  - Garantia da precedência dos elementos pelo seu encadeamento (contiguidade lógica) usando-se apontadores.

Operações com listas lineares

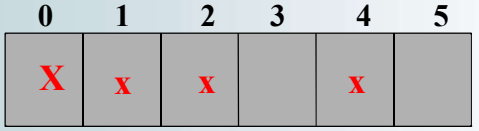
- acessar o k-ésimo item (nó) (para obter e/ou alterar seu conteúdo).
- inserir novo item (nó) antes/depois do k-ésimo nó.
- remover o k-ésimo item (nó).
- concatenar duas listas.
- determinar o tamanho (quantidade de nós).
- localizar o nó com determinado conteúdo.
- classificar os nós quanto ao seu conteúdo.

- Lista de compras:

1 - **Arroz**, 1 kilo
2 - **Feijão**, 2 kilos
3 - **Macarrão**, 1 pacote
4 - **Tomate**, qt 5

Listas por contiguidade

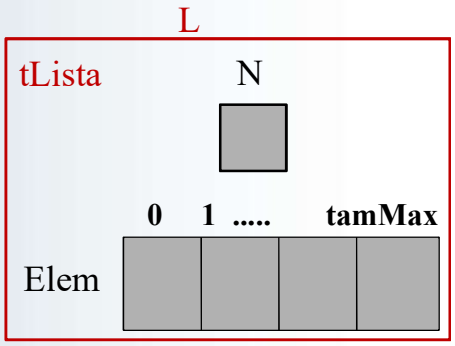
- Este tipo de representação é baseada em um arranjo unidimensional onde a precedência entre os elementos da lista é conseguida aproveitando o fato de que os elementos do arranjo estão em endereços contíguos da memória.



Lista com 1 elemento
Lista com 2 elementos
Lista com 3 elementos

Listas por contiguidade

```
const int TamMax = 100;

Struct (registro) tLista
      {
          int N;
          int Elem[TamMax];
       };
tLista L;
```



Rever registro
Rever Vetor de Objetos

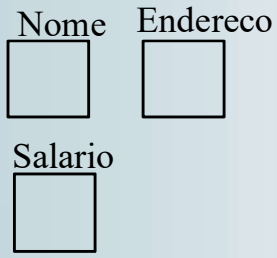
Registro - struct

- Coleção de dados que são de tipos diferentes
- Exemplo: ficha de cadastro de cliente
  - Nome: string
  - Endereço: string
  - Telefone: int
  - Salário: float
  - Idade: int

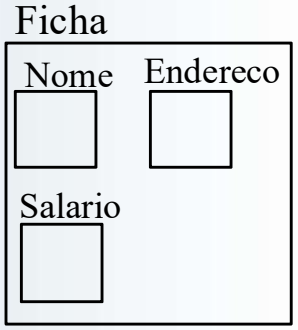
Ficha

Nome:_____
Endereço:_____
Telefone:_____
Salário:_____
Idade:_____

Registro - struct



Nome = “Engenharia”;
Endereco = “Castanhal”;



Ficha.Nome = “Engenharia”;
Ficha.Endereco = “Castanhal”;

Então, o que é um registro?

Registro – struct - Classe

- Java não possui um elemento registro como em outras linguagens (C/C++, pascal)
- Possui o elemento classe que pode ser utilizado como um registro
- Ex: cliente

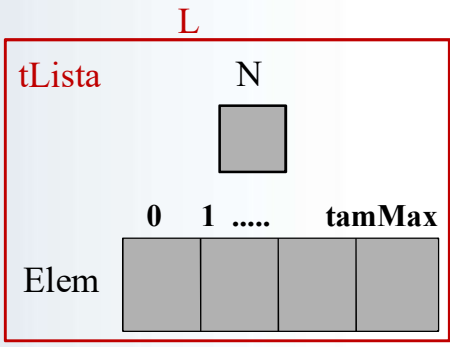
  conta

Registro – struct - Classe

- Ex Cliente
  - public class Cliente {
    - public String nome;
    - public String endereco;
    - public int telefone;
    - ...
    - }

Vetor de Registro

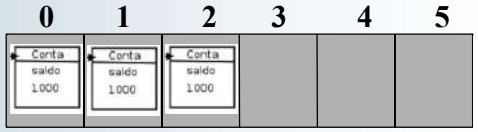
- Vetor é uma estrutura de dados capaz de armazena um conjunto de elementos do mesmo tipo.



inteiro

double

registro



Registro – struct - Classe

```
public class Cliente {
    public String nome;
    public String endereco;
    public int telefone;
    ...
    }
```

- Como utilizar:
- Cliente cliente1 = new Cliente();
- cliente1.nome = "José Gomes";
- cliente1.endereco = "Av. Jose Malcher";
- cliente1.telefone = “12345678";
- cliente1.salario = 1500.00;
- cliente1.idade = 30;

Rever registro - ok
Rever Vetor de Objetos

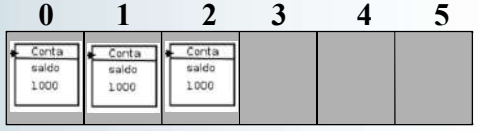
Vetor de Registro

- Vetor é uma estrutura de dados capaz de armazena um conjunto de elementos do mesmo tipo.

inteiro

double

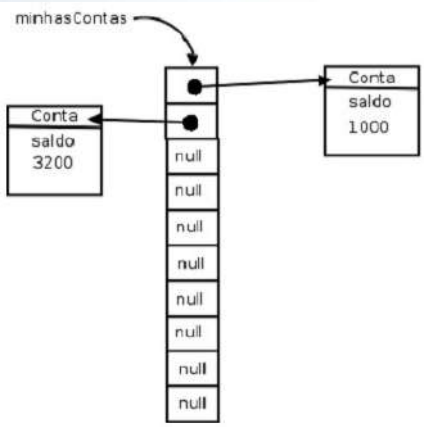
registro



Arrays de referencia ou vetor de registro

```
class Conta {
      int numero;
      String nome;
      double saldo;
      double limite;
```

- Conta[] minhasContas;
- minhasContas = new Conta[10];
- Conta contaNova = new Conta();
- contaNova.saldo = 1000.0;
- minhasContas[0] = contaNova;
- ou
- minhasContas[1] = new Conta();
- minhasContas[1].saldo = 3200.0;

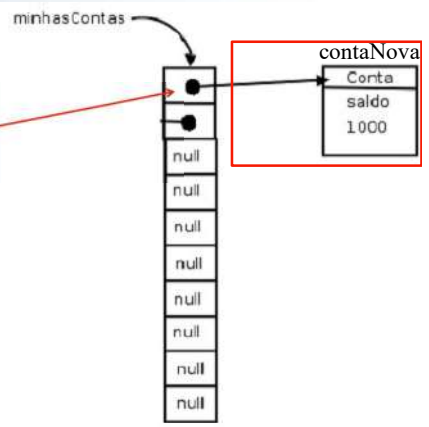


Arrays de referencia ou vetor de registro

```
class Conta {
      int numero;
      String nome;
      double saldo;
      double limite;
```

- Conta[] minhasContas;
- minhasContas = new Conta[10];
- Conta contaNova = new Conta();
- contaNova.saldo = 1000.0;
- minhasContas[0] = contaNova;

Cria um novo Objeto Conta de nome contaNova. Insere um valor em saldo (da contaNova) e associa a primeira posição do vetor ao objeto criado.



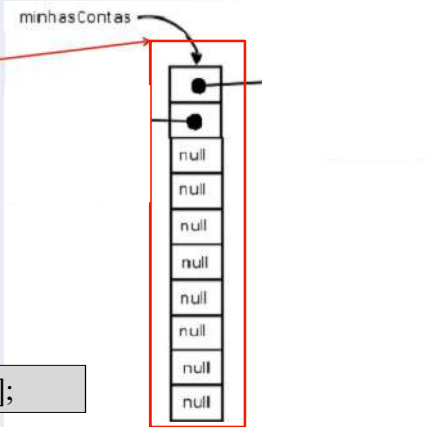
Arrays de referencia ou vetor de registro

```
class Conta {
      int numero;
      String nome;
      double saldo;
      double limite;
```

- Conta[] minhasContas;
- minhasContas = new Conta[10];

São criados 10 espaços para guardar uma referência a uma Conta.
Por enquanto, eles se referenciam para lugar nenhum (null).

Conta[] minhasContas = new Conta[10];

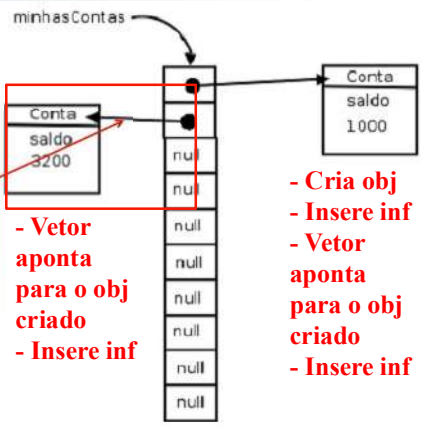


Arrays de referencia ou vetor de registro

```
class Conta {
      int numero;
      String nome;
      double saldo;
      double limite;
```

- Conta[] minhasContas;
- minhasContas = new Conta[10];
- Conta contaNova = new Conta();
- contaNova.saldo = 1000.0;
- minhasContas[0] = contaNova;
- ou
- minhasContas[1] = new Conta();
- minhasContas[1].saldo = 3200.0;

Podemos criar um novo Objeto já referenciando com a posição do Vetor.



Vetor de Registro

- Cria obj, Insere inf., Vetor aponta para o obj criado e Insere ou atualiza inf.

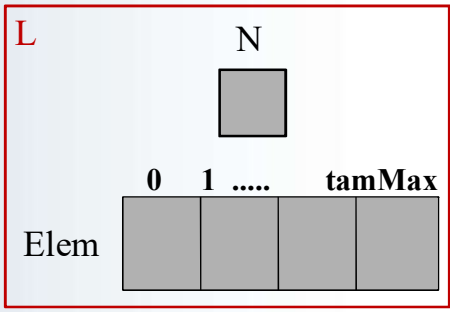
- Cliente[ ] clientes; variável clientes será vetor de objetos Cliente
- clientes = new Cliente[quantidade]; Cria o vetor
- Cliente c = new Cliente(); Cria o objeto c. De qual tipo?
- c.nome = “Maria"; Atribuindo informações ao obj c
- c.endereco = “rua Tamoios";
- c.telefone = “12345678";
- c.salario = 1500.00;
- c.idade = 30;
- **clientes[indice] = c;** ??????

Classe Cliente
nome endereco telefone salario idade

Listas por contiguidade

Relembrando

```
const int TamMax = 100;
Struct (registro) Lista
      {
          int N;
          int Elem[TamMax];
       };
Lista L;
```



Vetor de Registro

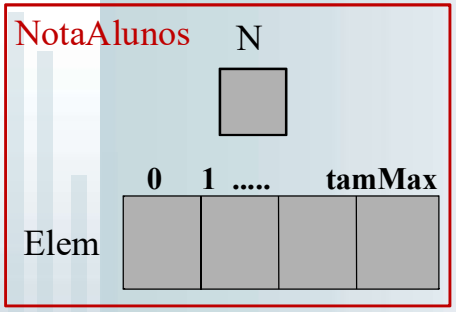
- Vetor aponta para o obj criado e Insere ou atualiza inf.

- Ou ....
- Cliente[ ] clientes;
- clientes = new Cliente[quantidade];

- clientes[indice].nome = “Maria";
- clientes[indice].endereco = “Rua Tamoios"
- clientes[indice].telefone = “1234567";
- clientes[indice].salario = 1500.00;
- clientes[indice].idade = 30;

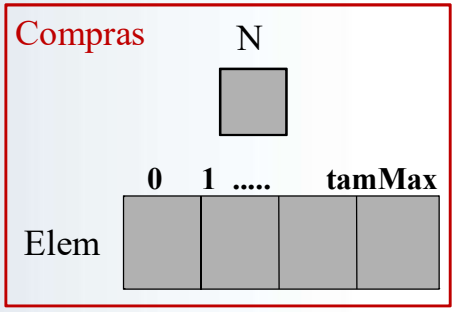
Listas por contiguidade

Lista NotaAlunos;



NotaAlunos.N = 1;
NotaAlunos.Elem[0] = 9;
NotaAlunos.N = 2;
NotaAlunos.Elem[2] = 8;

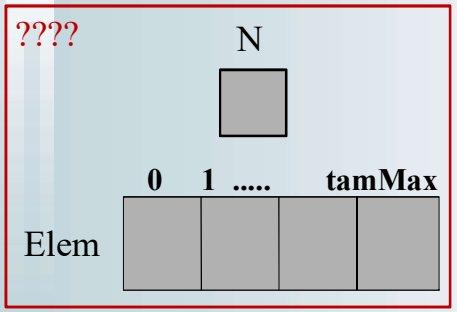
Lista Compras;



Compras.N = 1;
Compras.Elem[0] = “arroz”;
Compras.N = N+1;
Compras.Elem[1] = “sal”;

Listas por contiguidade

- Supor a criação de uma lista contigua para guardar combinações de números da mega sena. A lista contém 6 itens. Como ficaria a sintaxe da lista?



N vai existir?
Vetor Elem vai existir?

Quais informações Elem irá guardar?
Qual tamanho de Elem?

Numero, int

Listas por contiguidade

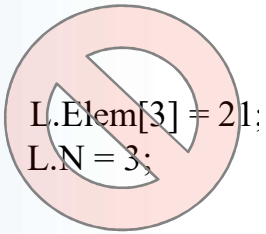
Classe para MEGASENA

```
public class Megasena {
  int N;
  int[] Elem = new int[6];
}
```

Duas classes
- Uma para declarar a LISTA
- Outra para manipular a LISTA

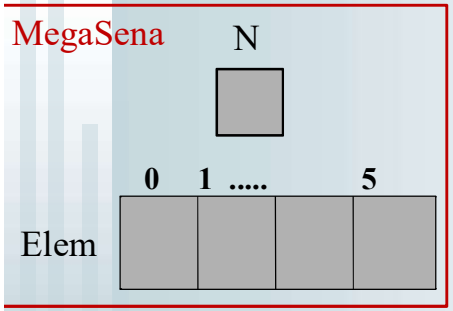
Classe principal

```
public static void main(String args[]) {
    // preencher a lista diretamente
    Megasena L = new Megasena();

    L.N = 1;
    L.Elem[0] = 12;

    L.N = 2;
    L.Elem[1] = 30;

    L.Elem[2] = 21;
    L.N = 3;
```



Listas por contiguidade

- Numero (inteiro). A lista terá 6 itens.
- Como ficaria a sintaxe da lista?

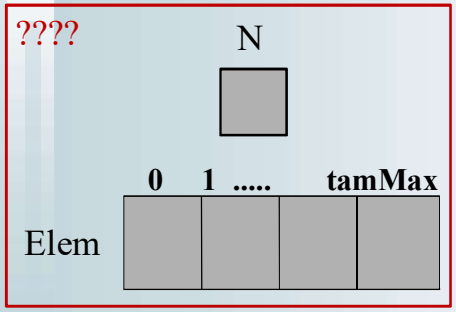


Classe para MEGASENA

```
public class Megasena {
  int N;
  int[] Elem = new int[6];
}
```

Listas por contiguidade

- Supor a criação de uma lista contigua para guardar as informações das galáxias do universo, como: Nome, magnitude e constelação. A lista poderá ter até 100 itens. Como ficaria a sintaxe da lista?



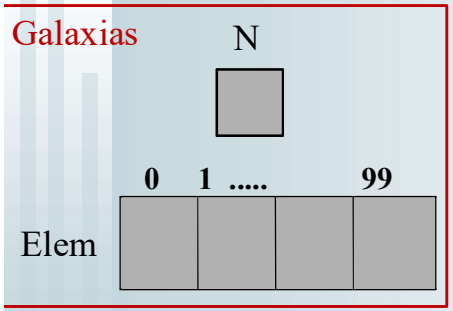
N vai existir?
Vetor Elem vai existir?

Quais informações Elem irá guardar?

Nome, string
Magnitude, float
Constelação, string

Listas por contiguidade

- Nome, magnitude e constelação. A lista poderá ter até 100 itens.
- Como ficaria a sintaxe da lista?



Classe para GALAXIAS

```
public class Galaxias {
   int N;
   infG[] Elem =
                new infG[100];
}
```

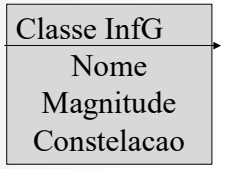
Listas por contiguidade

```
public static void main(String args[]) {

    Galaxias L = new Galaxias();

    infG g1 = new infG();
    g1.nome = "Grande Nuvem de Magalhães";
    g1.magnitude = 0.9;
    g1.cosntelacao = "Dorado";

    L.N = 1;
    L.Elem[0] = g1;

    infG g2 = new infG();
    g2.nome = "Via Láctea";
    g2.magnitude = -6.5;
    g2.cosntelacao = "Sagittarius";

    L.N = 2;
    L.Elem[1] = g2;
```

Listas por contiguidade

Classe para GALAXIAS

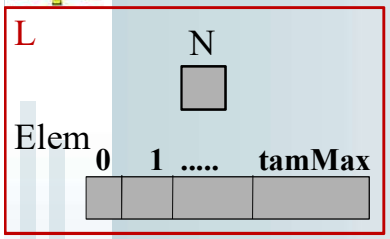
```
public class Galaxias {
   int N;
   infG[] Elem = new
            infG[100];}
```



Três classes
- Uma para declarar a LISTA
- Uma para declarar o tipo de vetor
- Outra para manipular a LISTA

```
public class InfG {
    String nome;
    float magnitude;
    String constelacao;
    }
```

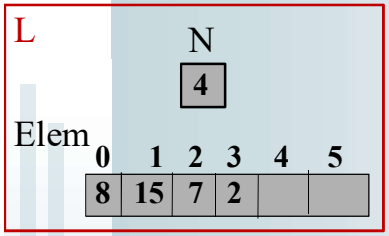
Listas por contiguidade



Classe para LISTA

```
public class Lista {
   int N;
   int[] Elem = new int[100] ;
}
```

Obter o k-ésimo elemento



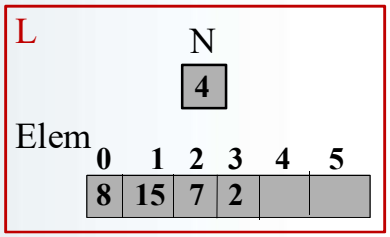
Criar um método para acessar o elemento da posição solicitada
- O que passar para o método como parâmetros??

K
L.Elem[k]

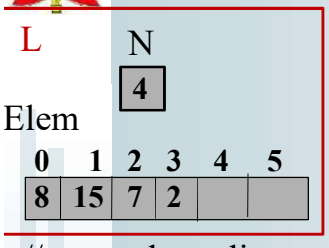
```
int Acesso(Lista L, int k) {
  // índice inválido
  if ((k < 0) || (k > (L.N-1) ))
    // código de erro
      return -9999;
  else
      return L.Elem[k];
}
```

Alterar o k-ésimo elemento

```
void Alterar(Lista L, int k, int Val)
        //k é a posição que vai ser alterada
{
  if ( (k < 0) || (k > (L.N-1) ) )  //índice inválido
       System.out("indice invalido);
  else {

       L.Elem[k] = Val;
  }
```



Obter o k-ésimo elemento




```
// preencher a lista
 Lista L = new Lista();
L.N = 5;
L.Elem[0] = 43;
L.Elem[1] = 52;
L.Elem[2] = 21;
L.Elem[3] = 15;
L.Elem[4] = 9;
```

Chamando o método

```
public static int Acesso(Lista L, int k){
  if ((k < 0) || (k > (L.N – 1) ) )
    return -9999;
  else
    return L.Elem[k];   }
```

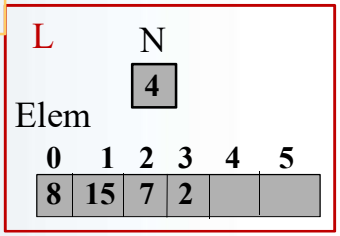
Chamando o método

```
int retorno = Acesso(L,1);
 if (retorno == -9999)
 System.out.println("Acesso inválido.
                Numero fora da lista!");
 else
 System.out.println("O numero da lista e: “
                                + retorno);
```

Inserir um novo elemento (não pode apagar o elemento já existente)

Tem espaço?

```
static void Inserir(Lista L, int k, int Val) {
    int i;
 if (L.N == TamMax) // espaço esgotado {
    System.out.println("Espaço esgotado!");
 } else if (k == L.N) { //apos o ultimo elemento
    L.Elem[k] = Val;
    L.N++;
 } else if ((k < 0) || (k > L.N - 1)) //indice invalido      {
    System.out.println("Indice inválido");
 } else {//deslocar elementos a direita
    for (i = L.N - 1; i >= k; i--)
        L.Elem[i + 1] = L.Elem[i];
    L.Elem[k] = Val;
    L.N++; //ajustar tam. Logico   }  }
```



Remover o k-ésimo elemento

```
void Remover(tLista L, int k) {
int i;
  if (k < 0 || k > L.N-1)  // índice inválido
        System.out("indice invalido);
  else {
    for (i = k; i<L.N-1; i++) //deslocar elementos esqu.
        L.Elem[i] = L.Elem[i+1];
    L.N--; //ajustar tamanho lógico
    }
}
```

Remover o ultimo elemento, simplesmente N--