Algoritmos e Estrutura de Dados - IV

Unidade 1 - Revisão - Estruturas de Dados Básicas

Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

Bibliografia

- **Head First Java, 2nd Edition by Kathy Sierra and Bert Bates**
- Core Java Fundamentals – Horstmann / Cornell – PTR- Volumes 1 e 2 – 8th Edition
- Java How to Program - 9th Edition by Paul Deitel and Harvey Deitel
- Beginning Java 2 – Ivor Horton – 1999 WROX
- Java2 – The Complete Reference – 7th Edition – Herbert Schildt – Oracle Press
- Inside the Java 2 – Virtual Machine Venners – McGrawHill
- Understanding Object-Oriented Programming with JAVA – Timothy Budd – Addison Wesley
- Effective Java, 2nd Edition by Joshua Bloch
- Thinking in Java (4th Edition) by Bruce Eckel

Bibliografia

- **Data Structures and Algorithms in Java, Fifth Edition, John Wiley & Sons, Michael Goodrich, Roberto Tamassia, 2010**
- Estrutura de Dados e Algoritmos – Bruno R. Preiss, Editora Campus, 2001
- Estrutura de Dados e Algoritmos em Java – Robert Lafore, Editora Ciência Moderna, 2005
- Algoritmos e Estrutura de Dados – Niklaus Wirth – Editora Prentice Hall do Brasil, 1989

Programação Orientada a Objetos

- Um programa orientado a objeto é um conjunto de objetos que trocam mensagens para, ao final do processamento, resolver o problema do usuário.

- Cada objeto tem uma **funcionalidade** que é exposta aos usuários (interface) e a implementação é, em geral, escondida dos mesmos (encapsulamento).

Ojetos são criado por meio de classes...

OOP em Java

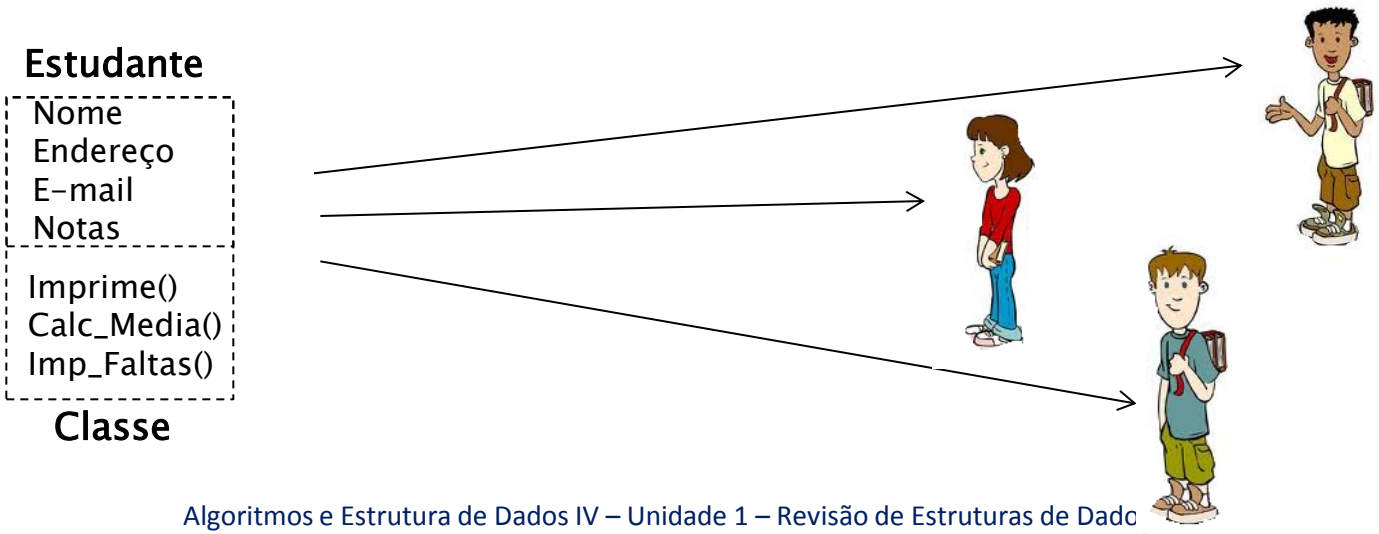
- Todo código escrito em Java está dentro de uma **classe**.
- A plataforma disponibiliza centenas de classes por meio de **API**.
- Você ainda pode criar classes para descrever objetos do domínio do problema de suas aplicações.

Documentação API - Java

http://docs.oracle.com/javase/8/docs/api/

Objetos

- Os dados (propriedades, atributos) de um objeto são os **campos instância**.
- Os procedimentos que operam os dados do objeto são os **métodos**.
- Um objeto específico (instância de uma classe) tem seus campos instância (valores) particulares e isto os tornam distintos de outros objetos (**individualidade**).
- O conjunto de valores de um objeto específico constitue o seu **estado**.

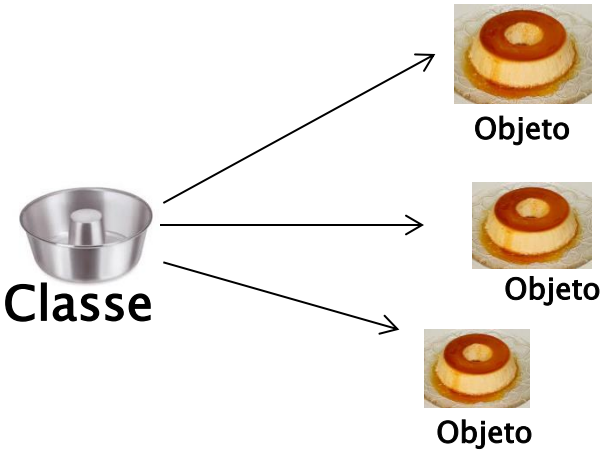


Construção de Objetos

- Em Java, usamos ***construtores*** para criar novas instâncias.
- Um ***construtor*** é um método especial cujo propósito é criar e inicializar objetos.
- Construtores têm sempre o mesmo nome da classe.
- Para criarmos objetos, combinamos o construtor com o operador **new**.

Definindo classes

- Um objeto é um repositório de dados e sempre é obtido a partir de uma classe.
- Objetos têm propriedades (dados) e desempenham funções.
- Os dados do objeto são definidos por meio de campos (variáveis instância).
- Exemplo: **x** é uma variável instância que faz referência a um objeto do tipo Aluno (classe Aluno).
- **x.nota** é um atributo do objeto.
- **x.media()** é um método.



Definindo classes

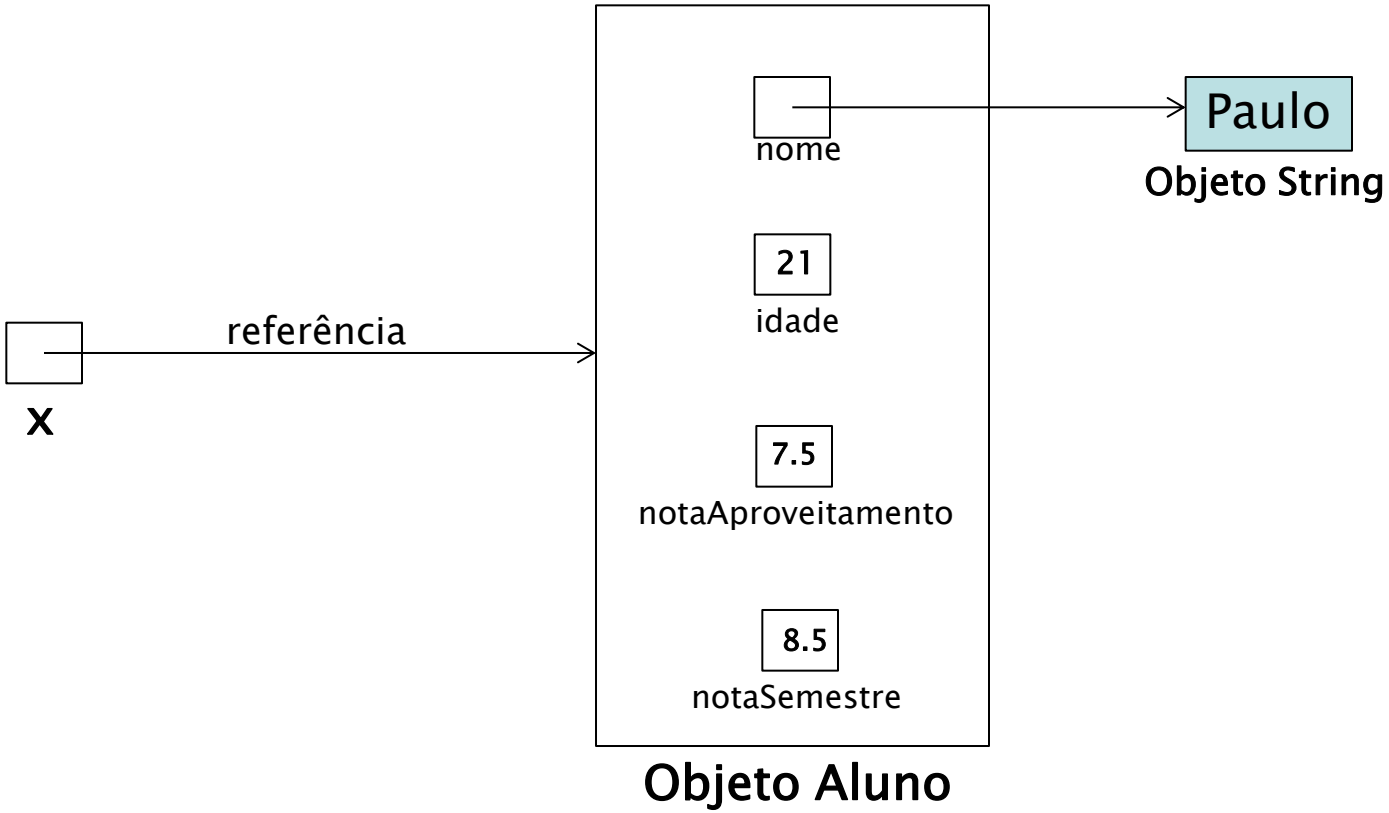
```
package uscs;
public class Aluno {

        String   nome;
        int      idade;
        double   notaAproveitamento;
        double   notaSemestre;

        public  double retornaMediaSemestre( )  {
                return (notaAproveitamento + notaSemestre)/2;
        }

        public void imprimeDadosAluno() {
                System.out.println("Eu sou " + nome + " e tenho " + idade + " anos.");
        }
}
```

Criando-se objetos

```
package uscs;
public class Teste_Aluno {

        public  static void main (String[] args )  {
                Aluno x = new Aluno();
                x.nome = "Paulo";
                x.idade = 21;
                x.notaAproveitamento = 7.5;
                x.notaSemestre = 8.5;
                x.imprimeDadosAluno();
        }

}
```

Saída: Eu sou Paulo e tenho 21 anos.

Criando-se objetos



Exercício

1. Criar uma classe Java chamada **Aluno** para modelar estudantes. A classe deve possuir os seguintes atributos de dados (propriedades):

nome – Nome do estudante – (tipo String)
codmat – Código de Matrícula – (tipo int)
cpf – (tipo String)
sexo – (tipo char)
notaP1 – (float)
notaP2 – (float)
notaP3 – (float)

A classe deve conter métodos para construir objetos e um método **ImprimeAluno()** que irá imprimir os dados do estudante. Adicionalmente a classe deve conter um método chamado **ImprimeSexo()** que irá imprimir "Masculino" se o sexo for 'M' e "Feminino" se o sexo for 'F'.

A função **ImprimeSexo()** também imprime o nome do estudante.

Codificar também a função **MediaAluno()** que retorna a média aritmética das duas maiores notas dentre as notas P1, P2 e P3. (Exemplo: notas 2, 6 e 8 => considerar para a média as notas 6 e 8.)

Finalmente, codificar a função **Resultado()** que retorna "Aprovado" se a média for >= 6.0 ou "Reprovado" se a média for inferior a 6.0 e a função **imprimeMedia()** que imprime a média do aluno.

A classe deve ser criada dentro de um package chamado **uscs**.

Exercício

Criar uma classe Java **TesteAluno** que possui um método **main()** para instanciar estudantes.

Criar um objeto referenciado pela variável **X1** por meio do construtor com os parâmetros: nome = “Paulo”, codmat = 55123, cpf=”800912345-12” , sexo = ‘M’, nota_P1 = 7.0, nota_P2=6.0 e nota_P3 = 8.0.

Criar um segundo objeto referenciado pela variável **X2** por meio do construtor com os parâmetros: nome = “Ana”, codmat = 991239, cpf=”500876123-15” , sexo = ‘F’, nota_P1 = 2.0, nota_P2=6.0 e nota_P3 = 9.0.

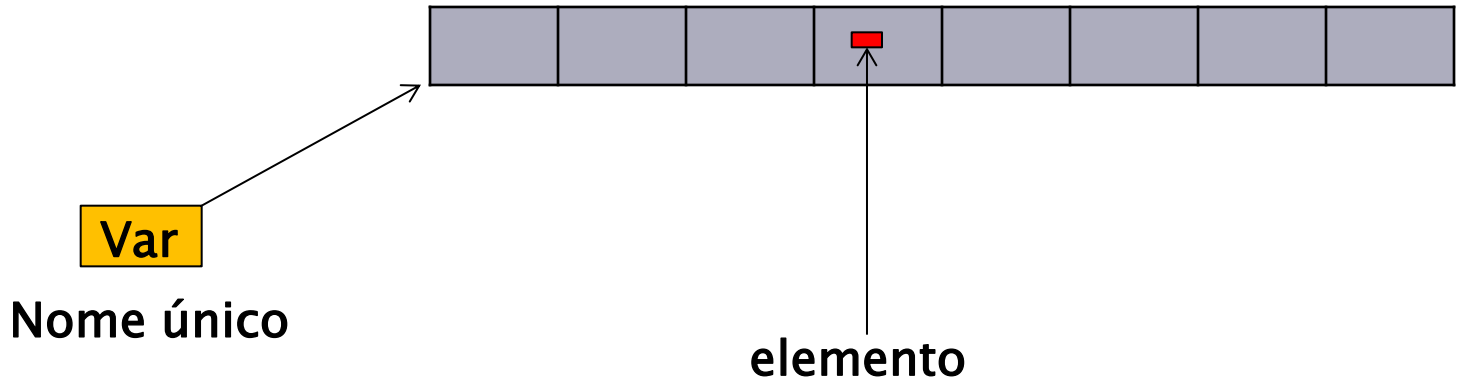
Para cada objeto chamar a função **ImprimeAluno()** para imprimir os dados dos dois objetos criados, e as funções **imprimeMedia()** e **Resultado()** .

Executar para cada objeto criado a função **ImprimeSexo().**

A classe deve ser criada dentro de um package chamado **uscs**.

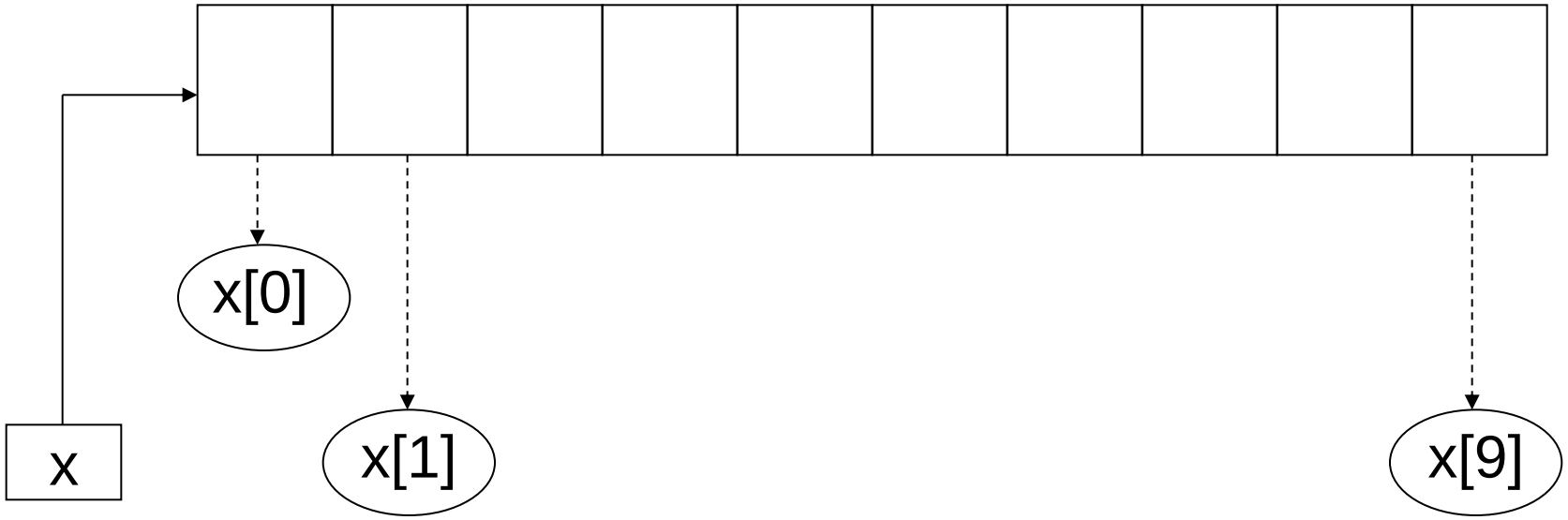
Estrutura Linear – array

- Um **array** é um conjunto de variáveis do mesmo tipo a qual atribuímos um nome único;
- Cada informação (dado) no array é chamada de elemento do array;
- Para fazermos referência à um elemento de um array devemos usar o nome do array em conjunto com um número inteiro chamado **índice**;
- O primeiro elemento do array tem índice **0**, o segundo **1**, e assim por diante.



Array - Instanciação

```
int[ ] x = new int[10];
```



Inicializando arrays

- Podemos inicializar um array explicitando os valores em tempo de declaração.
- Com este procedimento o tamanho do array e, consequente alocação de memória, é definido.

```
int [ ] x = {2,3,5,7,11,13,17 } ;
```

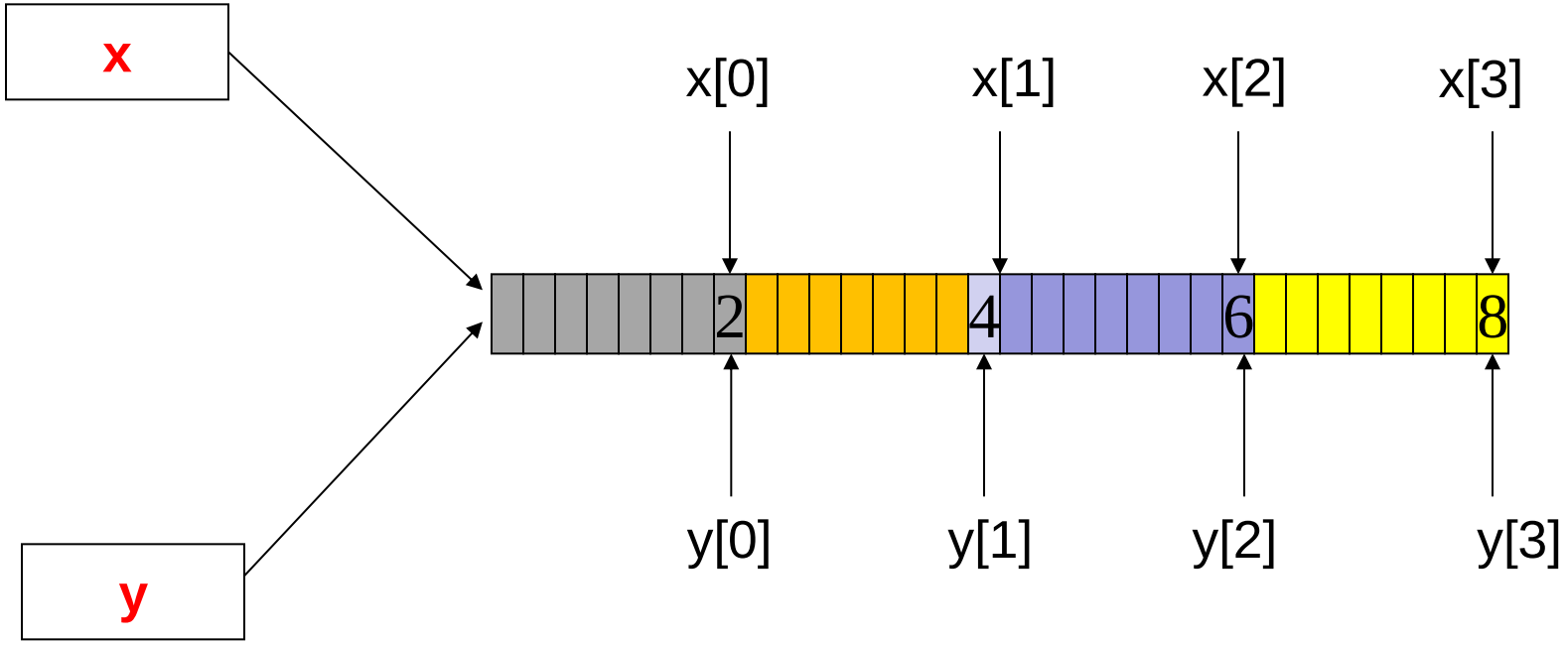
ou

```
int[ ] x = new int [ ]  { 2,3,5,7,11,13,17 } ;
```

O array acima tem 7 elementos inteiros.

Variáveis array

```
long [ ] x = {2L, 4L, 6L, 8L};
long [ ] y = x;
```



- Foram criadas duas variáveis array, porém temos apenas **um** array alocado em memória.

Populando arrays

```
double [ ] par = new double[50];
for (int i=0; i < 50; i++)
    par[i] = 100.0 *Math.random();
```

- Utilizamos elementos de array da mesma forma que usamos variáveis do mesmo tipo de dados.

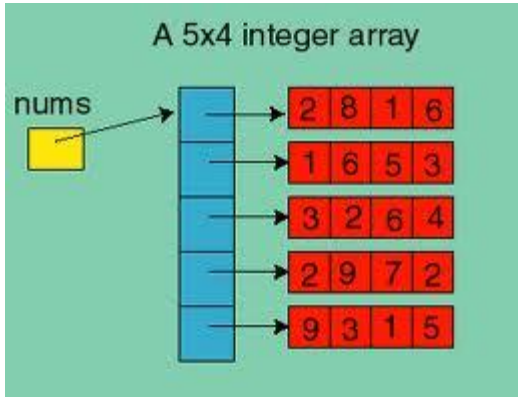
Imprimindo os elementos do array

```
package uscs;

public class ArrayPrint {

	public static void main(String[] args) {
		int[] x = new int[50];

		for (int i=0; i< x.length; i++) {
			x[i]=i;
			System.out.println(x[i]);
		}
	}

}
```

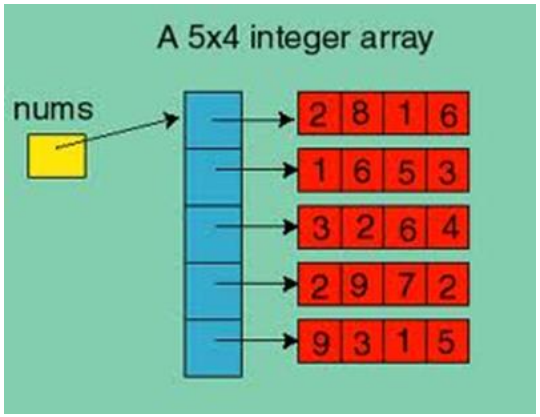
Array de arrays

- Usam mais de um índice para acessar os elementos do array.
- São usados para tabelas e outros arranjos mais complexos.
- São também chamados de arrays bidimensionais, uma vez que têm duas dimensões
- No exemplo, o primeiro índice se refere à quantidade de linhas e o segundo índice corresponde à quantidade de colunas da tabela.

```
int [ ] [ ]    nums = new int [5][4];
```



Carga de Array de arrays




```
package uscs;

public class ArrayArray {

        public static void main(String[] args) {

                int[][] nums  = new int[5][4];

                for (int r=0; r < nums.length; r++) {

                        for (int c=0; c < nums[r].length; c++) {
                                nums [r][c] = (int) (Math.random() * 10);
                                System.out.print(" " + nums[r][c]);
                        }
                        System.out.println("");
                }
        }
}
```

String

- É uma classe standard em Java a qual disponibiliza funcionalidades para o tratamento de listas de caracteres.
- Conceitualmente, strings são sequências de caracteres Unicode.
- Todo literal String (entre “ “) é uma instância da classe **String**.

Literais String

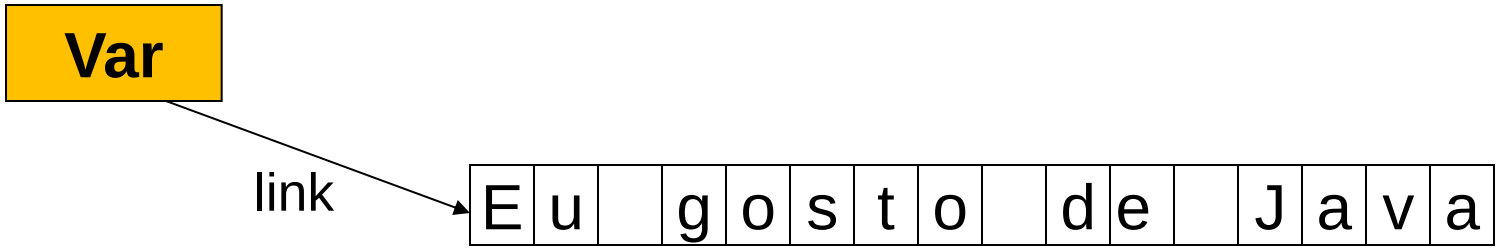
- Correspondem à uma seqüência de caractereres delimitados por “ ”.
- Exemplo:

  “Eu gosto de estudar na USCS !”

- O exemplo acima é um objeto constante da classe **String** que o compilador cria para usarmos no programa.

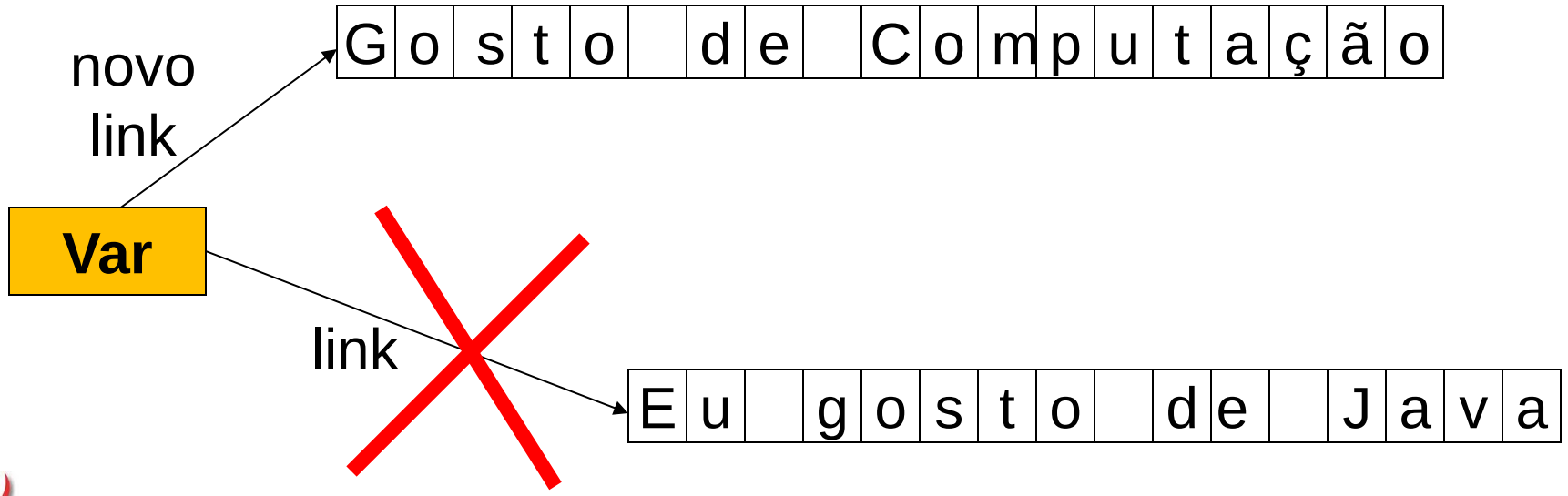
Criação de Strings

String Var = “Eu gosto de Java”;



Variáveis do tipo String

String Var = “Eu gosto de Java”;
Var = “Gosto de Computação”;



Criação de literais String

- Instâncias da classe **String** podem ser feitas **com** ou **sem** o operador **new**.

```
String S2 = "Linguagem Java";
```

```
String S1 = new String("Linguagem Java");
```

Concatenação de Strings

- Na linguagem Java, o sinal de + efetua concatenação de Strings.
- Toda vez que se concatena Strings, um novo String é instanciado.

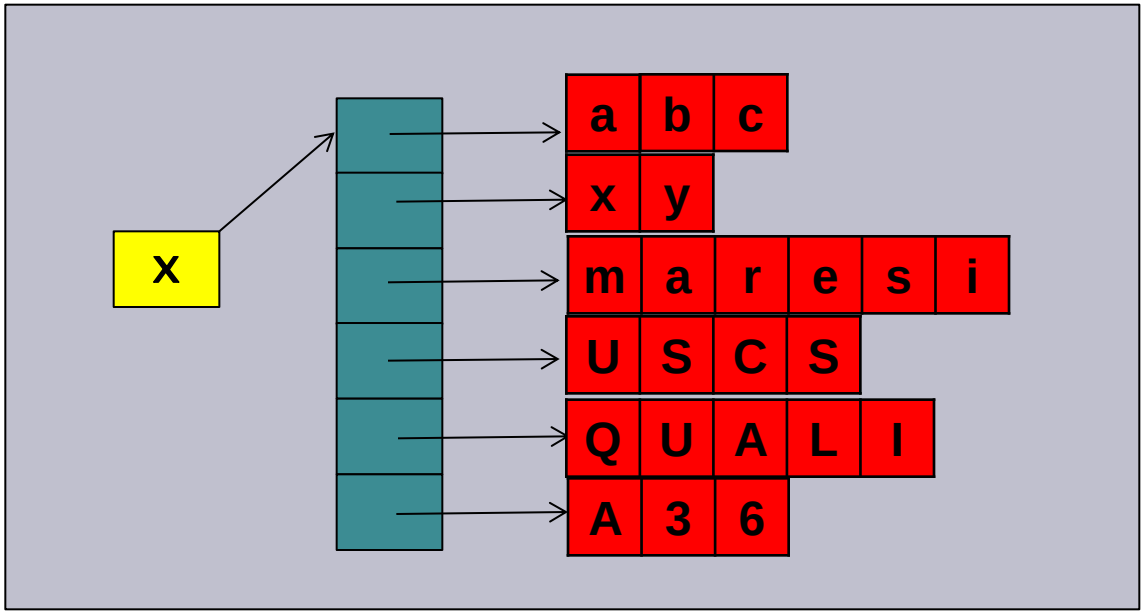
```
String str1 = new String(“ Resultado da “);
String str2 = new String(“ operacao “);

String str = str1 + str2;
```

concatenação

Array de Strings

- Tendo em vista que variáveis **String** são objetos, podemos então criar arrays de **Strings**.

String [ ] x = new String[6];



Array de Strings

```
public class SuperEquipes  {
  public static void main(String[] args)    {
        String[] equipes = {
                "Santos","Corinthians",
                "São Paulo","Portuguesa",
                "Palmeiras","São Caetano"
        };
        System.out.println("Seu time favorito hoje =  "
                                +
             equipes[(int)(equipes.length*Math.random())]);
  }
}
```

Entrando com dados pela console

- A classe **Scanner**, definida no package **java.util**, permite a leitura de dados a partir da “standard input stream”.

Obs. No eclipse, tecle <Ctrl> + <Shift> + O para import...

Entrando com dados pela console

```
import java.util.Scanner;

public class Scanner01 {

	public static void main(String[] args) {

		Scanner in = new Scanner(System.in);
		System.out.print("Qual o seu nome ? ");
		String nome = in.nextLine();

		System.out.print("Quantos anos você tem ?");
		int idade = in.nextInt();

		System.out.println("Olá " + nome +
			"," + "você tem " + idade + " anos...");
	}
}
```

Usando o método main para entrada de dados


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

            System.out.println("parametro1=" +
                  args[0] + " parametro2=" + args[1]);
      }
}
```

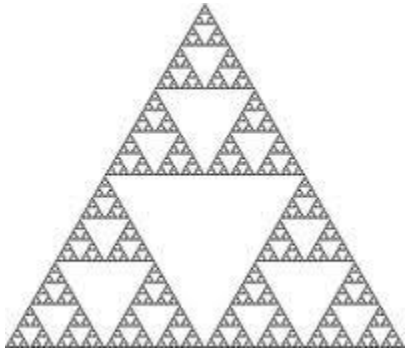
- linha de comando: java Entrada_Main USCS Java
- Impressao: parametro1=USCS parametro2=Java

Convertendo argumentos String para inteiros

- main considera todos os argumentos **Strings**.
- Para avaliar um argumento como **int**, use **Integer.parseInt().**

```
Integer.parseInt(variavel)
```

```
int x = Integer.parseInt(args[0]);
```

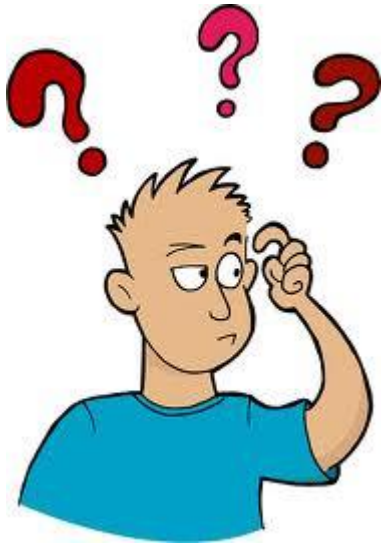


Recursão

- Repetição de instruções pode ser obtida por meio de loops (instruções for ou while).
- Outra forma de se implementar repetições é por meio de **Recursão**.
- Recursão ocorre quando uma função faz chamada de si própria.

Função Fatorial

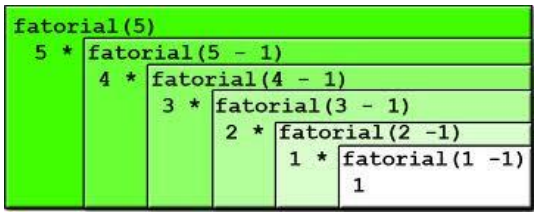
- Exemplo: 5! = 5.4.3.2.1 = 120



Será que a função fatorial pode ser definida de forma recursiva ?

Fatorial – Definição Recursiva

- Observe que fatorial(5) = 5.(4.3.2.1) = 5.fatorial(4)

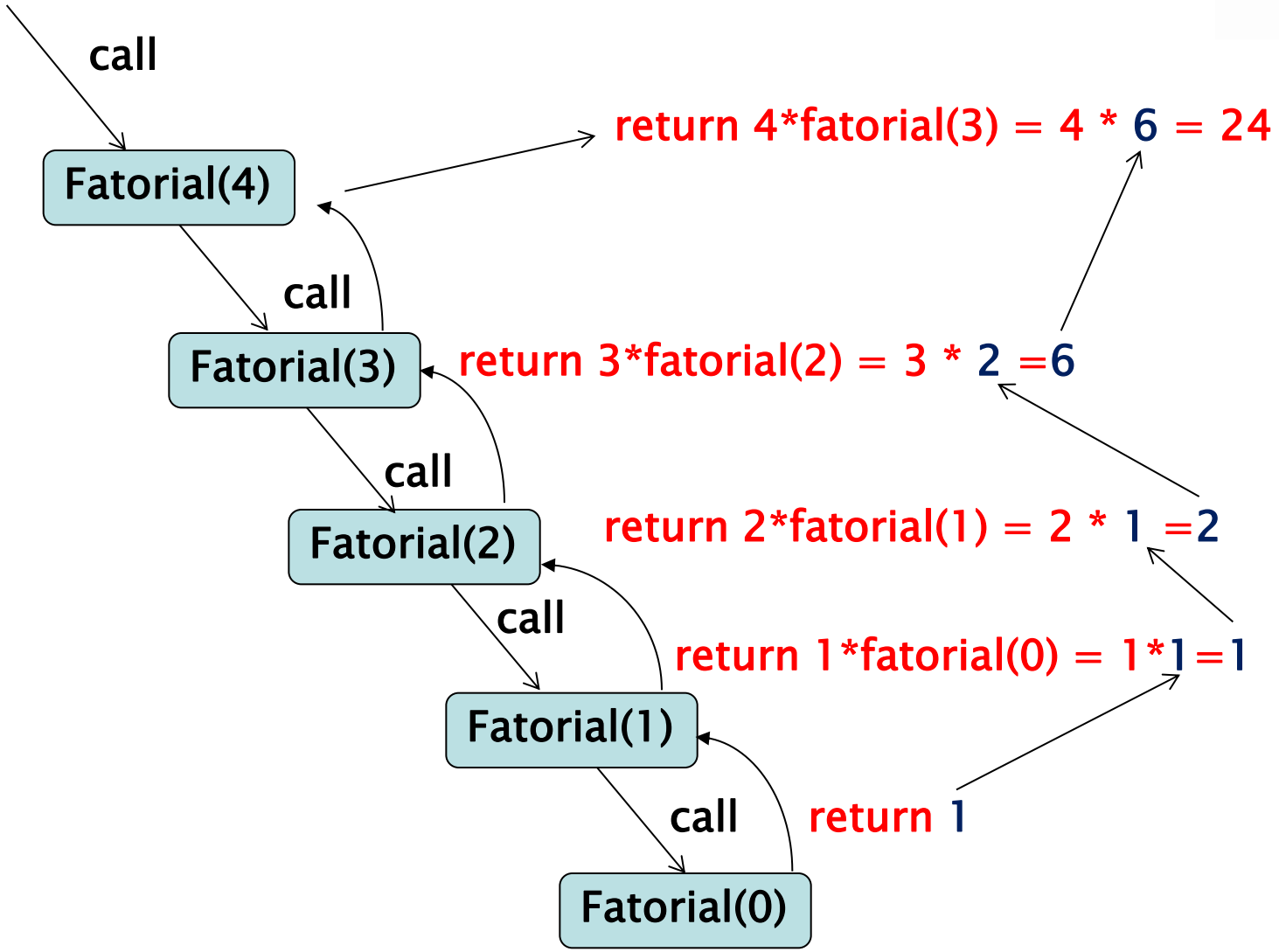


$$\text{fatorial}(n) = \begin{cases} 1 & \text{se } n = 0 \\ n.\ \text{fatorial}\,(n-1) & \text{se } n \geq 1 \end{cases}$$

Função Fatorial

```
package uscs;

public class Fat {

        public static void main(String[] args) {
                int n=4;
                System.out.println("Fatorial(" + n + ") = " +
                fatorial(n) );

        }

        public static int fatorial(int n) {
                if (n==0)
                        return 1; //caso básico
                else
                        return(n*fatorial(n-1)); //caso recursivo
        }

}
```

Trace de Recursão

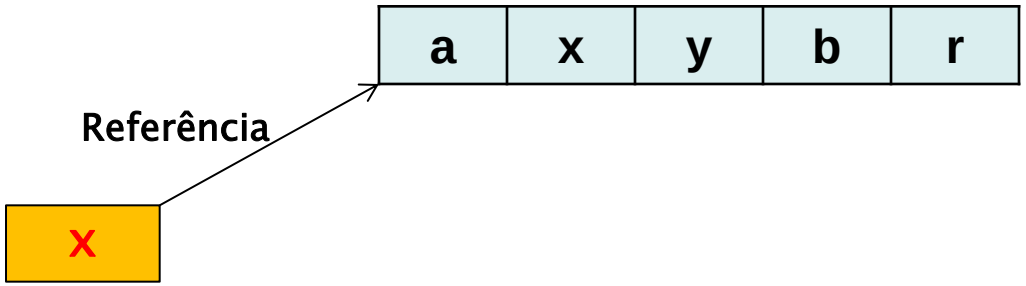


Listas

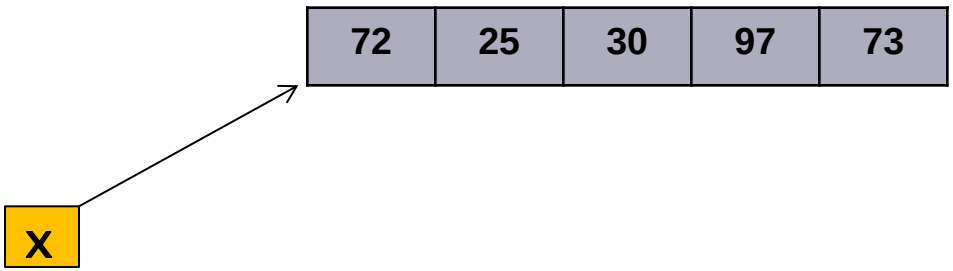
- Uma lista ou sequência é uma estrutura de dados abstrata que implementa uma **coleção ordenada de valores**, onde o mesmo valor pode ocorrer mais de uma vez.

- Uma lista é um **tipo abstrato de dados** (especificação de um conjunto de dados e operações que podem ser executadas sobre esses dados) .

Implementando listas com arrays...

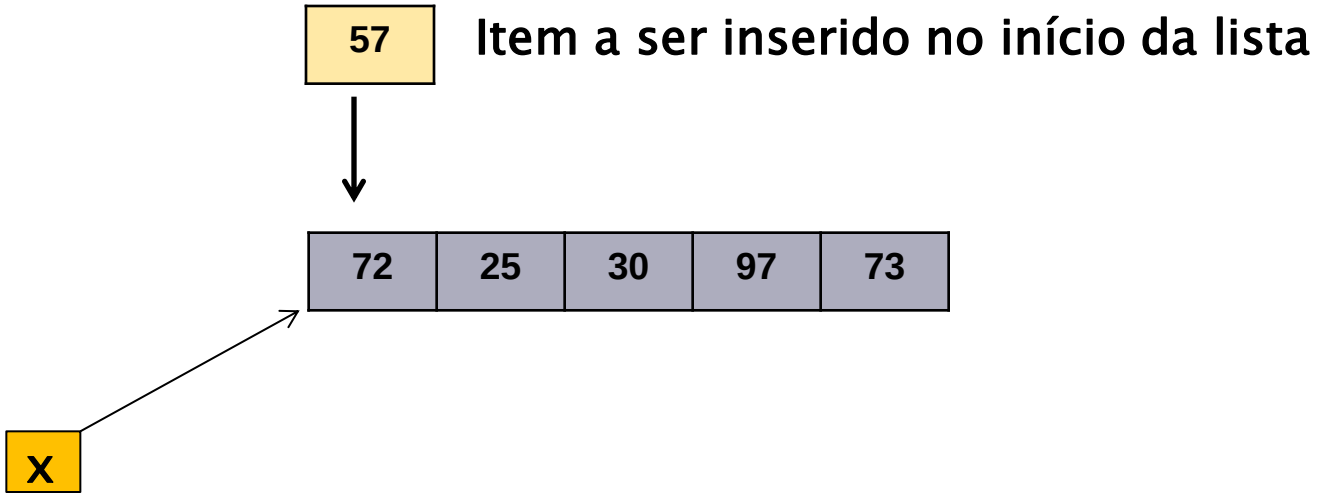


Exemplo: lista de 5 valores aleatórios entre 0 e 100

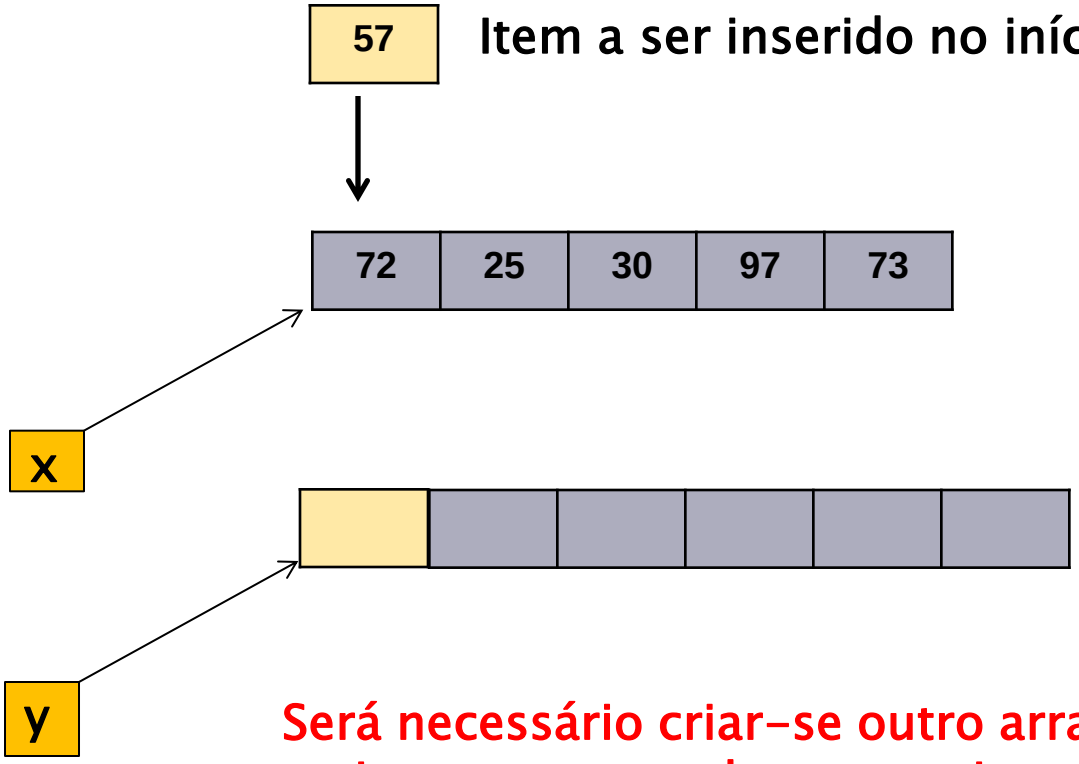


Mas, há alguns inconvenientes em se implementar listas com arrays...

Considere a necessidade de se inserir um item no início ou metade da lista ...



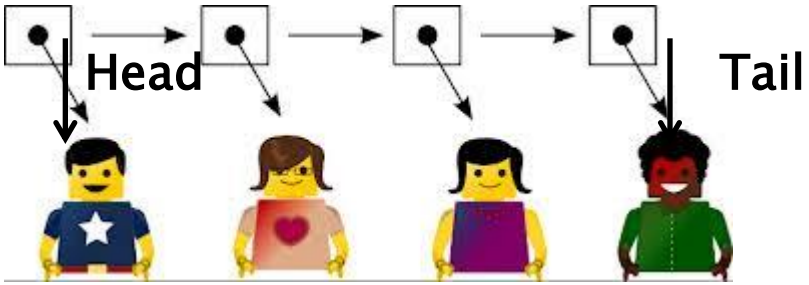
Primeiro, arrays têm tamanho fixo...



Será necessário criar-se outro array com tamanho maior para acomodar o novo item...

Listas Ligadas

- É um conjunto de nós que são definidos de forma **recursiva**.
- Cada nó tem um item de dado e uma **referência** ao próximo nó.
- O primeiro e último nó são chamados **HEAD** e **TAIL** respectivamente.



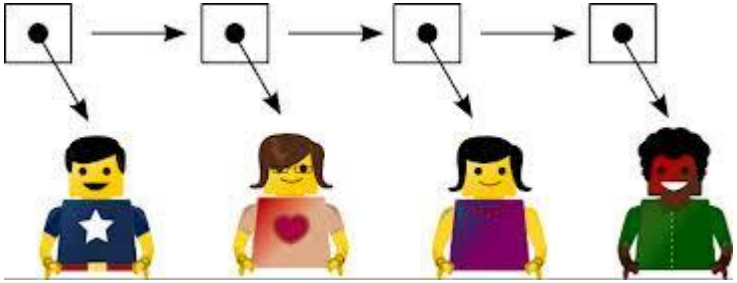
Vantagens sobre listas implementadas com arrays

- A inserção de um item no meio da lista leva **tempo constante**, caso você tenha a referência ao prévio nó.
- Listas ligadas podem crescer até o limite de memória oferecido pela máquina virtual.

Desvantagens sobre listas implementadas com arrays

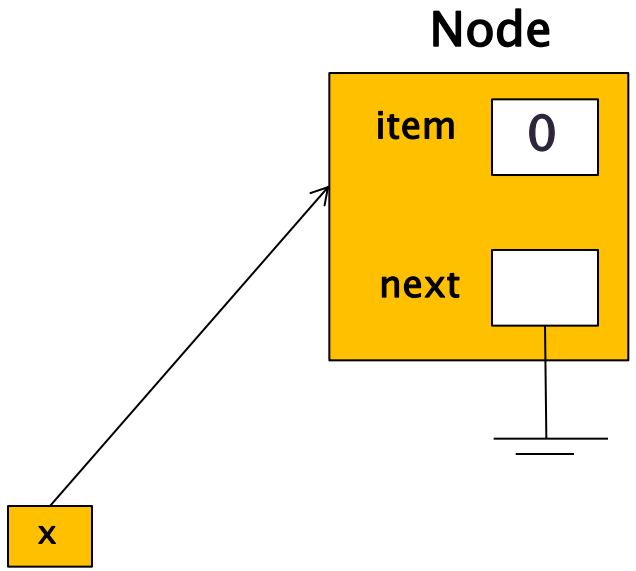
- A busca do nth elemento de um array é de tempo constante (índice).

- A busca do nth elemento de uma lista ligada é proporcional a n, sendo n o tamanho da lista. (A pesquisa se inicia a partir do HEAD até se encontrar de forma exaustiva o item procurado).



Implementação de Nós

```
package uscs;

public class Node {

        int item;
        Node next;

}
```

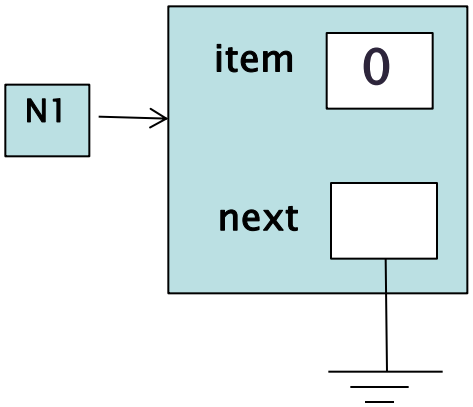


Criação de Nós

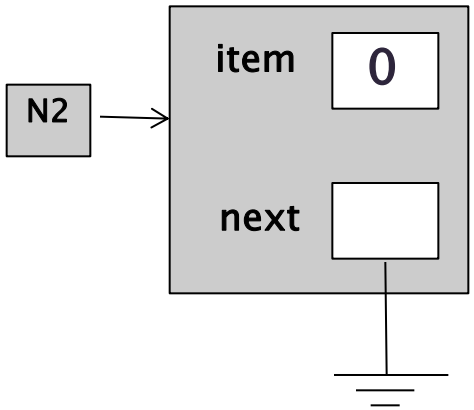
```
package uscs;

public class Test_ListNode {

        public static void main(String[] args) {

                Node N1;
                N1 = new Node();
                N1.item = 8;

                Node N2;
                N2 = new Node();
                N2.item = 5;

                Node N3;
                N3 = new Node();
                N3.item = 9;

        }
}
```

Criação de Nós

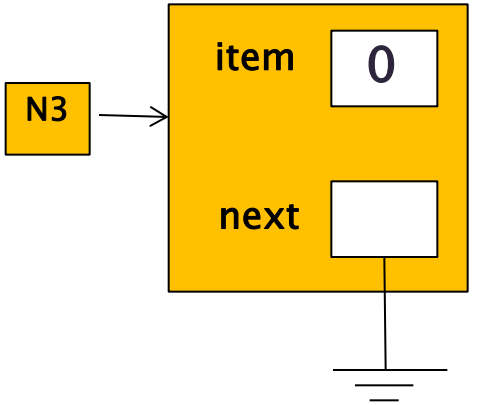
```
Node  N1;
N1 = new Node();
```




```
Node  N2;
N2 = new Node();
```

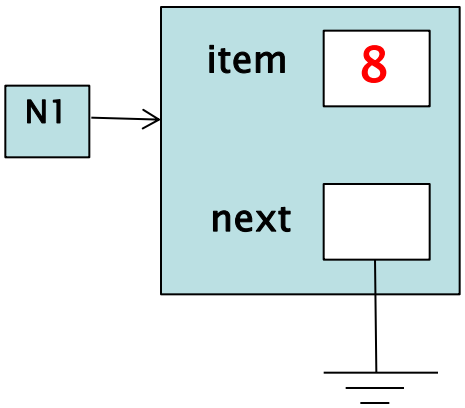



```
Node  N3;
N3 = new Node();
```

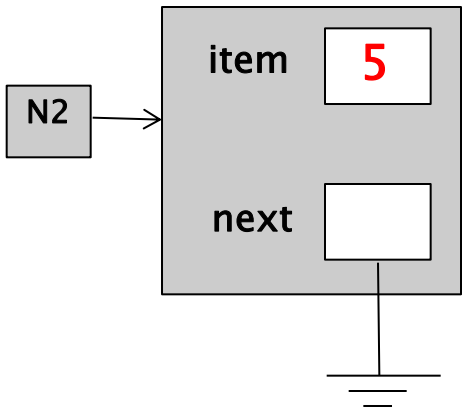


Armazenando valores nos Nós

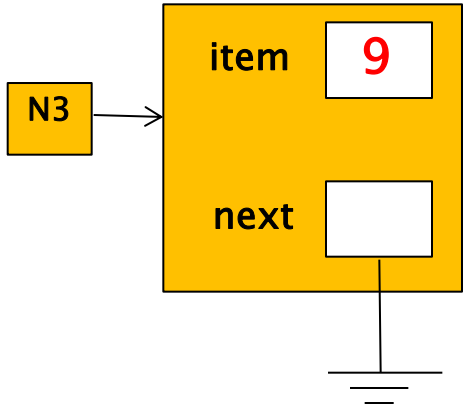
```
N1.item = 8;
```




```
N2.item = 5;
```

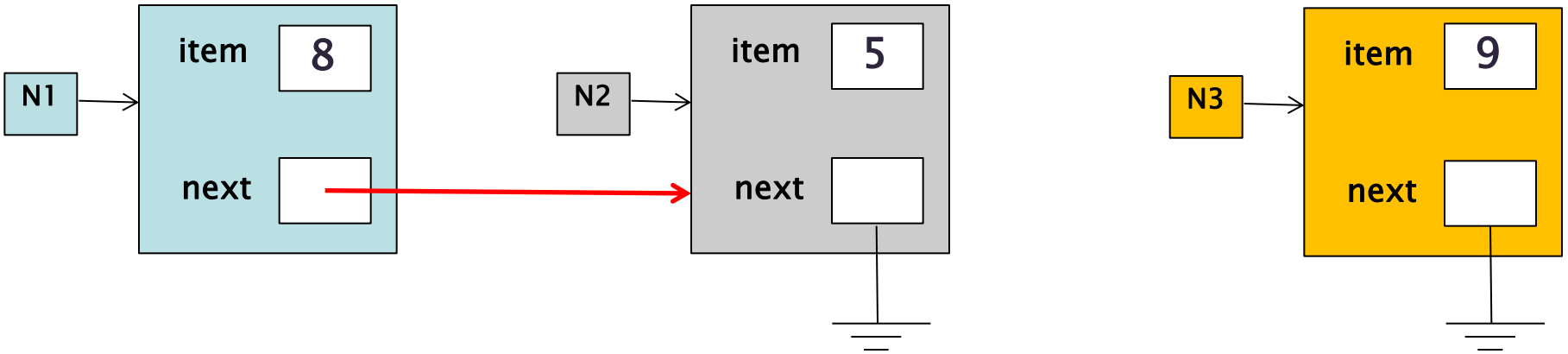



```
N3.item = 9;
```



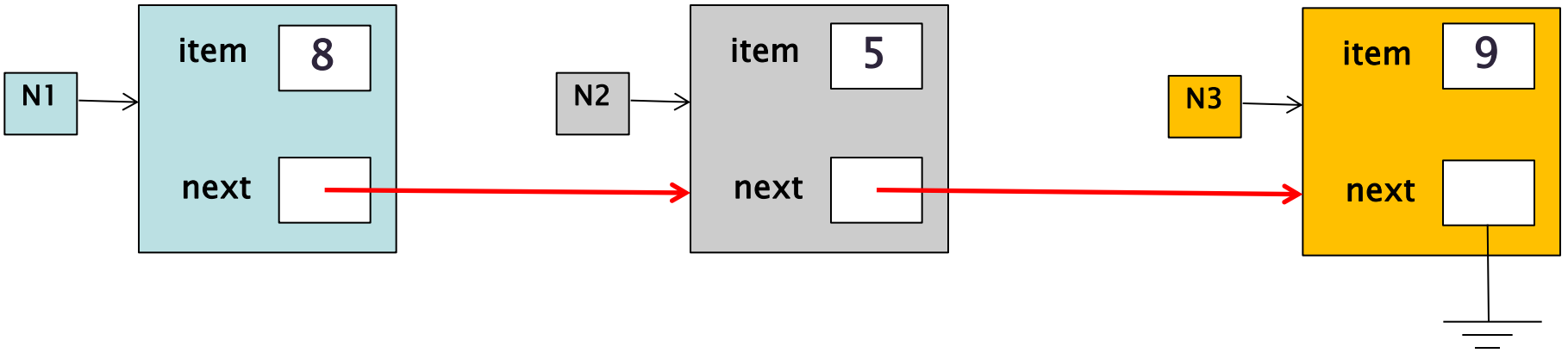
Encadeamento de Nós

```
N1.next = N2;
```



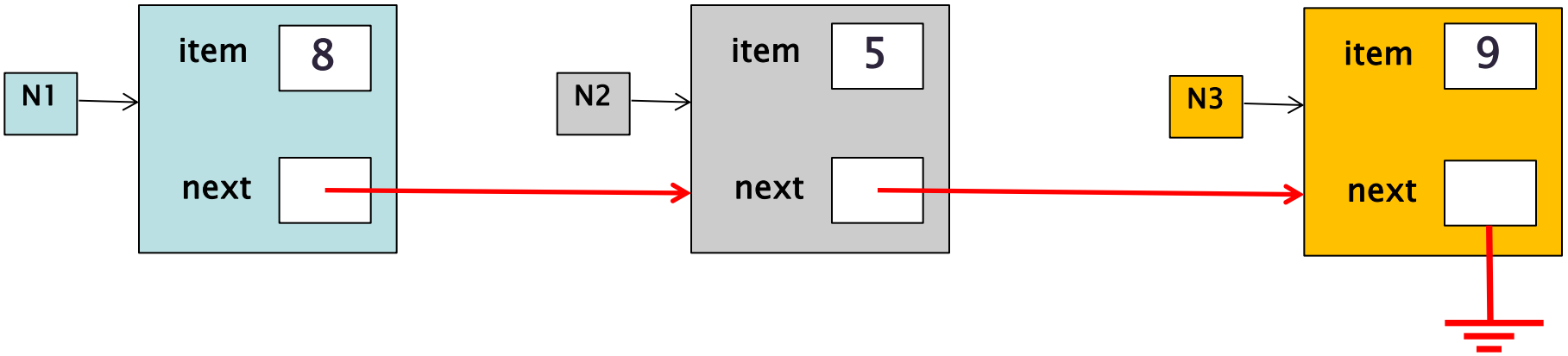
Encadeamento de Nós

```
N2.next = N3;
```

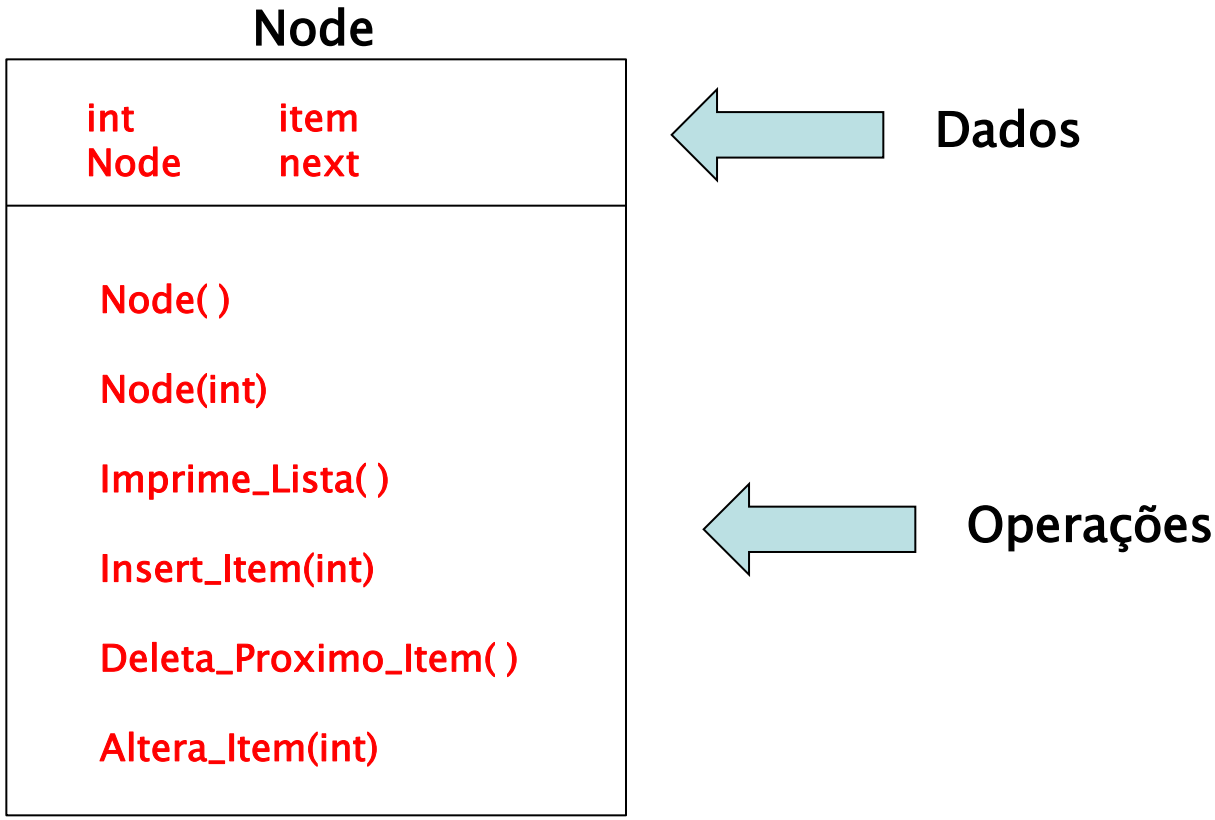


Encadeamento de Nós

```
N3.next = null;
```



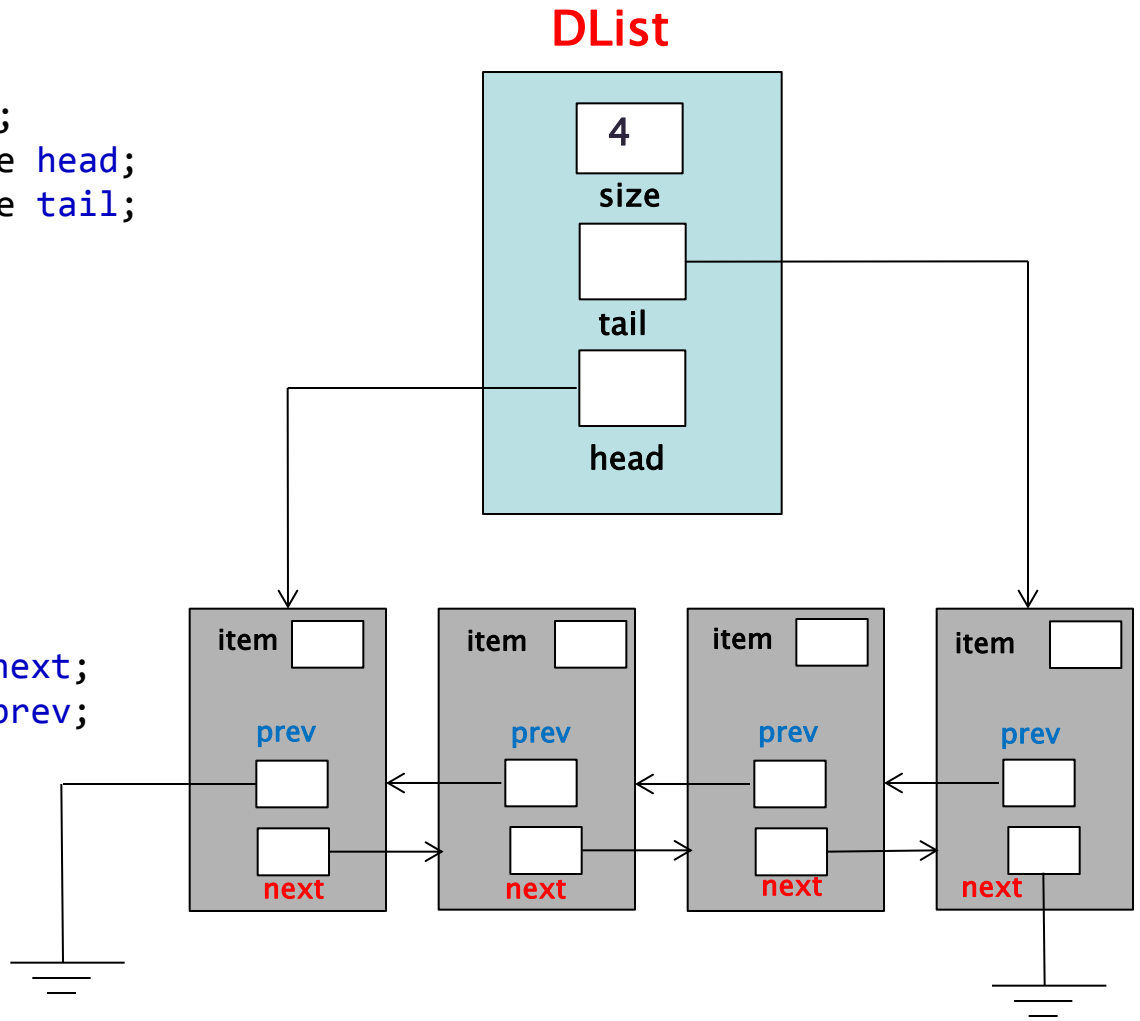
Tipo Abstrato de Dados



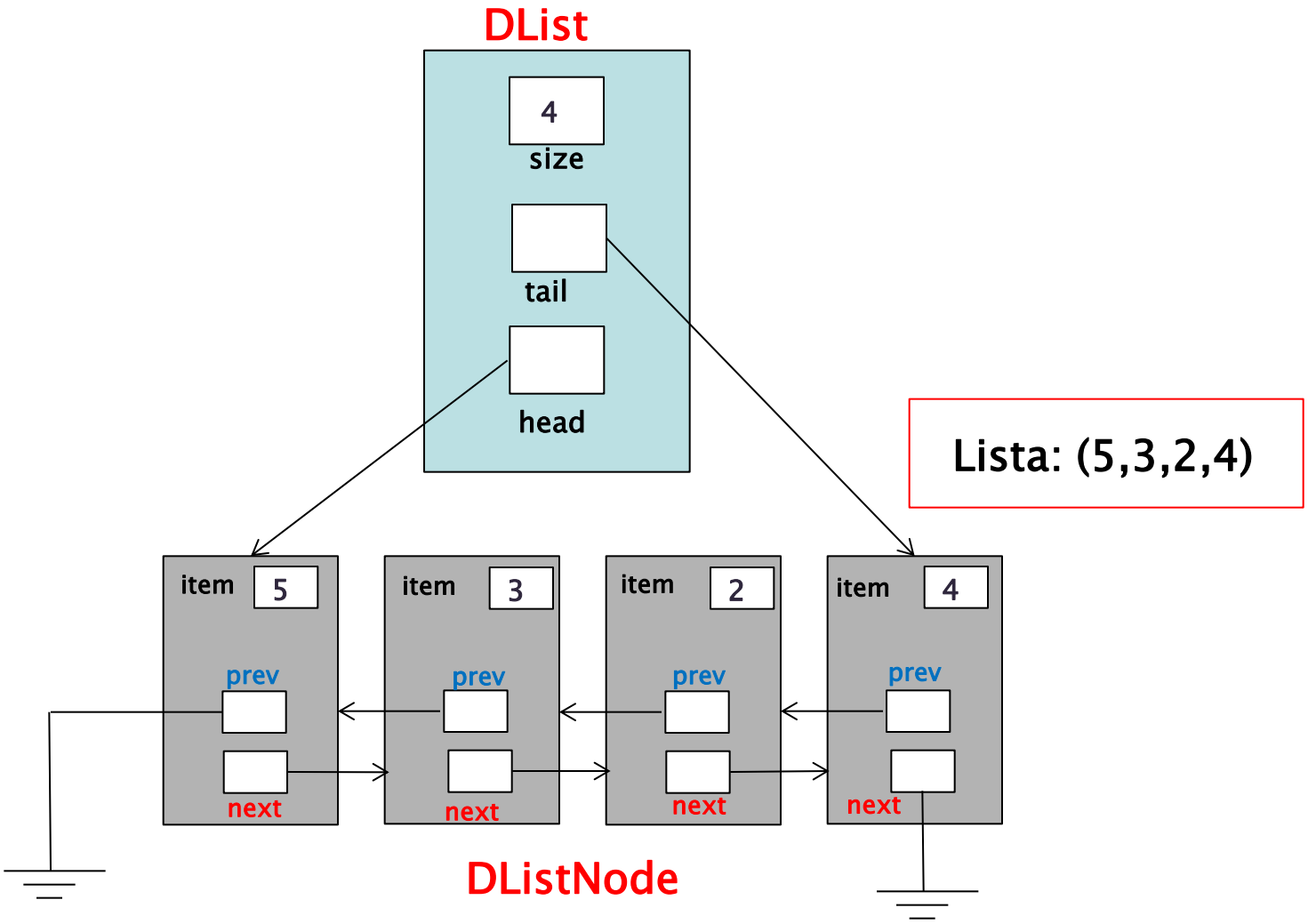
Listas duplamente ligadas

```
package uscs;

public class DList {

        public    int size;
        public    DListNode head;
        public    DListNode tail;

}
```

```
package uscs;

public class DListNode {

        public int item;
        public DListNode next;
        public DListNode prev;
}
```



Exemplo: Lista Duplamente ligada



Lista Circular

Lista: (5,3,2,4)