Lição 3

Técnicas Avançadas de Programação

Objetivos

Ao final desta lição, o estudante será capaz de:

- Definir e aplicar recursão na programação
- Diferenciar entre pilhas e filas
- Implementar pilhas e filas seqüenciais
- Implementar pilhas e filas encadeadas
- Usar as classes Collection existentes

O que é recursão?

- Pode ser aplicada quando a natureza do problema é repetitiva
- É menos eficiente que a iteração, mas é mais elegante
- É permitido aos métodos chamarem a si próprios
- Argumentos são armazenados temporariamente em uma pilha antes que a chamada do método seja completada

Recursão versus Iteração

- Iteração
  - Uso de estruturas de controle de repetição
  - Finalizado quando a condição do laço retorna falso
  - Mais rápida

- Recursão
  - Chama o método repetidas vezes
  - Finalizado quando uma condição particular é satisfeita
  - Encoraja a boa prática da programação

Recursão versus Iteração

- Ambos podem conduzir a um laço infinito
- Recursão ou interação?

Fatorial: Exemplo

- Passaremos agora para o NetBeans

Tipos de Dados Abstratos (TDA)

- Coleção de elementos de dados que fornece um conjunto de operações que são definidas nos elementos de dados

- Exemplos:
  - Pilhas (stacks)
  - Filhas (queues)
  - Árvores binárias (binarytree)

TDA: Pilhas

- Coleção de dados linearmente ordenados
- A manipulação de elementos é permitida somente no topo da pilha
- Aplicações:
  - Reconhecimento de padrões
  - Conversão entre notações infixa, pós-fixa e pré-fixas

TDA: Pilhas

- Duas operações:
  - Push
  - Pop
- Analogicamente
  - Pilhas de pratos

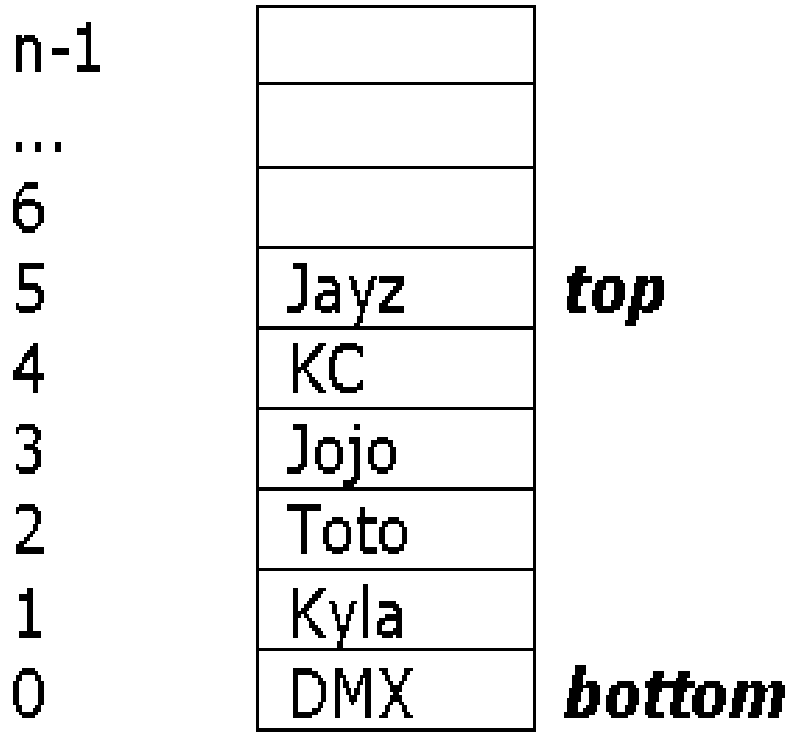
TDA: Pilhas

- Condição para pilha cheia:

  top == n-1.

- Condição para pilha vazia:

  top == -1

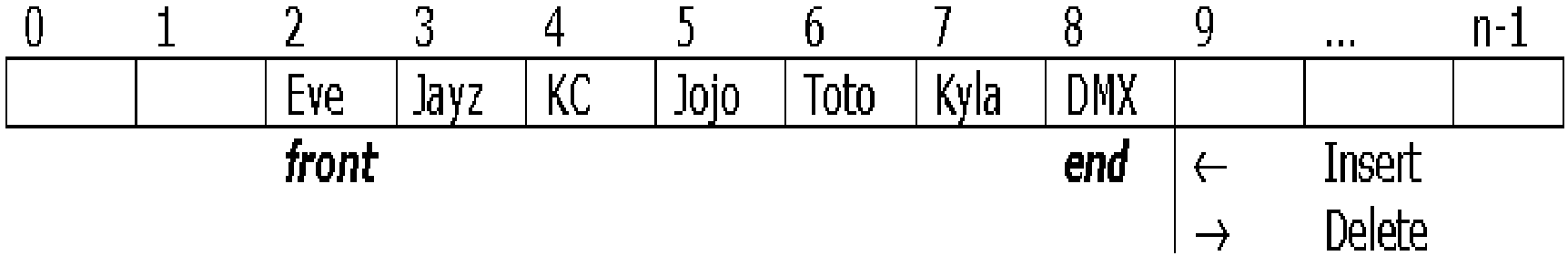


TDA: Filas

- Definição:
  - Enqueue
  - Dequeue

- Analogia:
  - Fila de espera em um banco

TDA: Filas



- Condição para fila cheia:

  end == n-1

- Condição para fila vazia:

  end < front

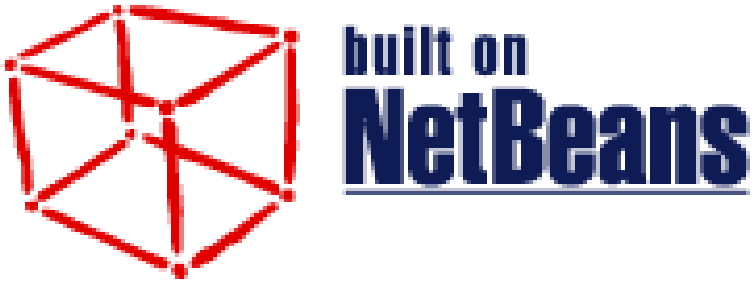
Representação seqüencial e encadeada

- Representação seqüencial
  - Fácil implementação
  - Fixa

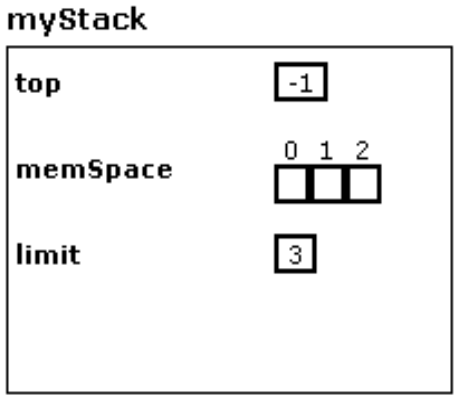
- Representação encadeada
  - Mais complicado de se implementar
  - Flexível

Representação seqüencial e encadeada

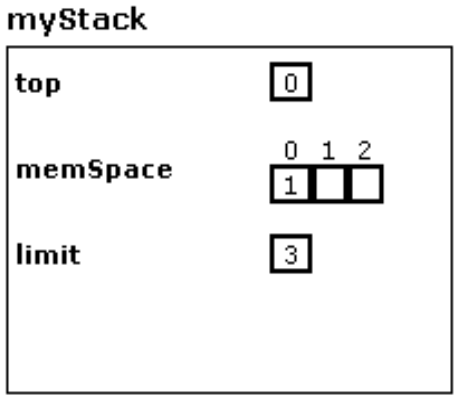
- Passaremos agora para o NetBeans



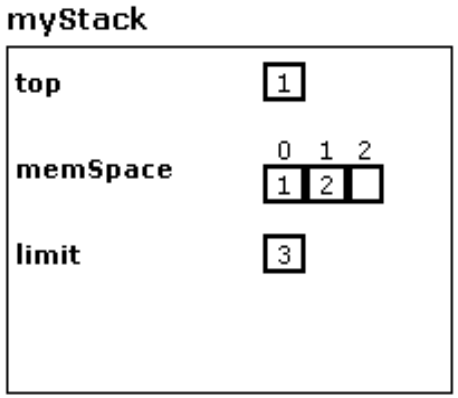
Representação seqüencial e encadeada



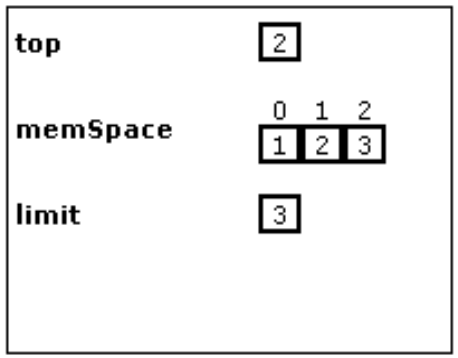
(a) SeqStack myStack = new SeqStack(3);



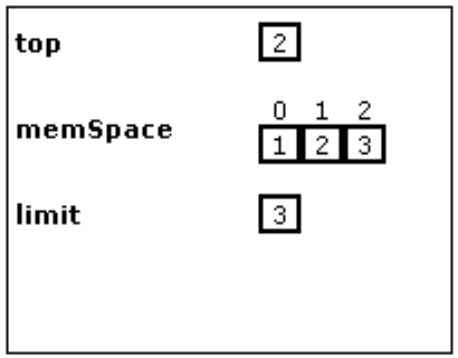
(b) myStack.push(1);



(c) myStack.push(2);

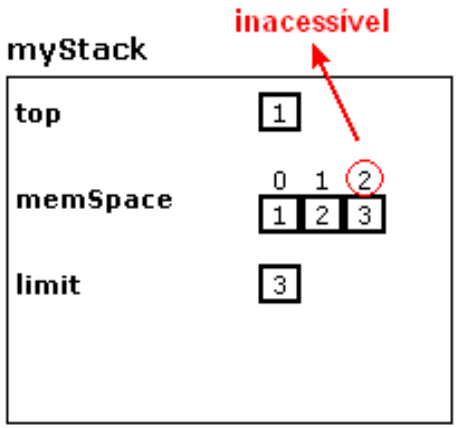


(d) myStack.push(3);



(e) myStack.push(4);

/* nenhuma */
mudança

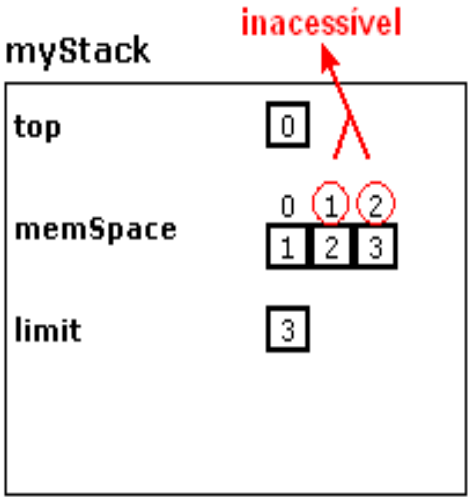


(f) System.out.println(myStack.pop());

prints ↳ 1°

3

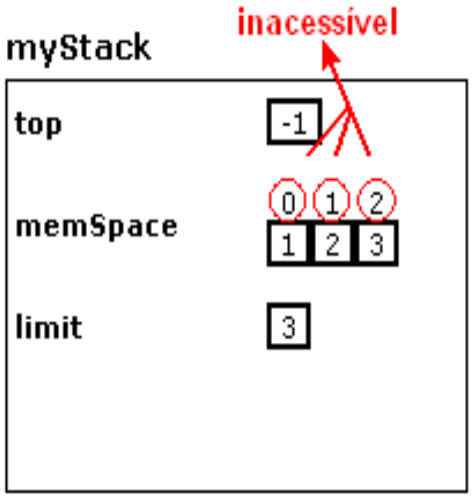
Representação seqüencial e encadeada



(g) System.out.println(myStack.pop());

prints ↳ 2º

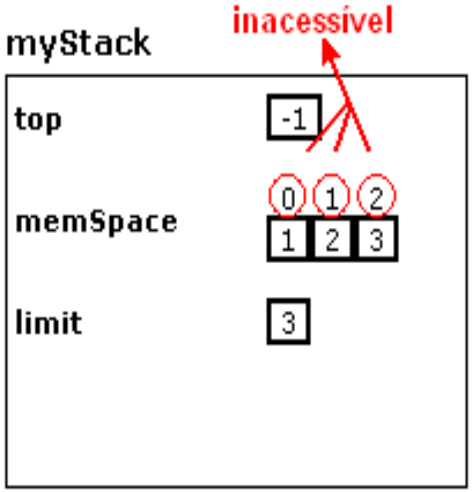
2



(h) System.out.println(myStack.pop());

prints ↳ 3º

1



(i) System.out.println(myStack.pop());

prints ↳ 4º

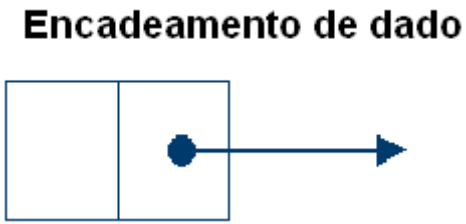
-1

/* nenhuma mudanca */

Lista encadeada

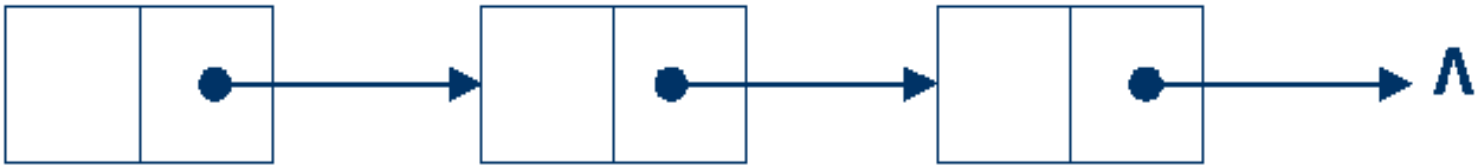
- Definição:
  - Estrutura dinâmica
  - Coleções de nós

- Nó:



Lista encadeada

- Nenhuma lista encadeada está vazia com três nós:

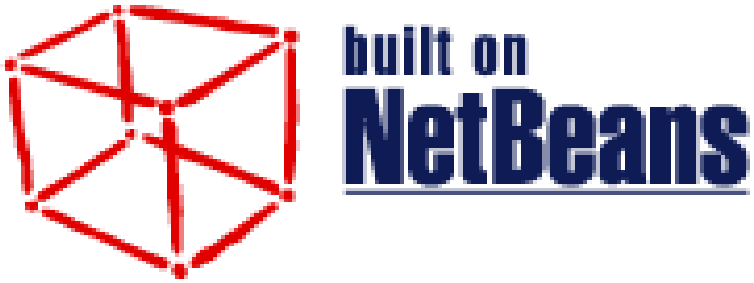


- Implementação da classe nó:

```
class Node {
   int data;
   Node nextNode;
}
```

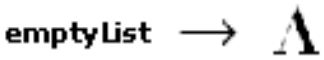
Lista encadeada

- Passaremos agora para o NetBeans



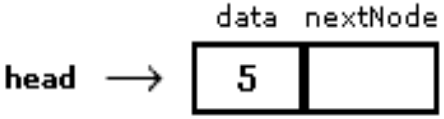
Lista encadeada

(a) Node emptyList = null;

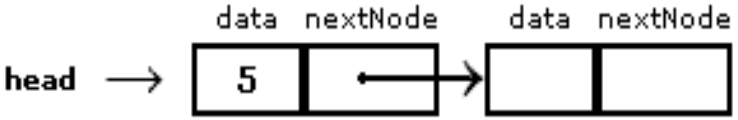


(b) Node head = new Node();

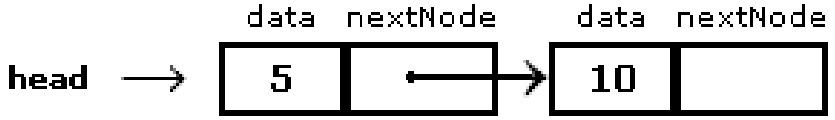
(c) head.data = 5;



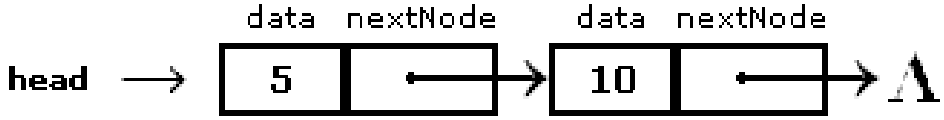
(d) head.nextNode() = new Node();



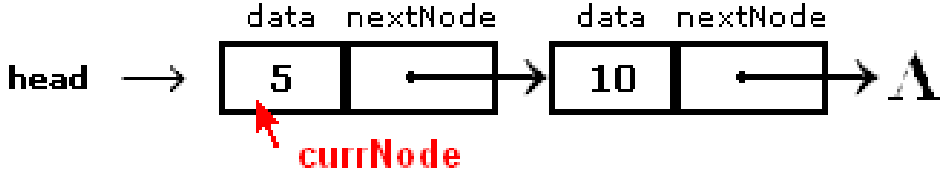
(e) head.nextNode.data = 10;



(f) head.nextNode.nextNode = null;



(g) Node currNode = head;



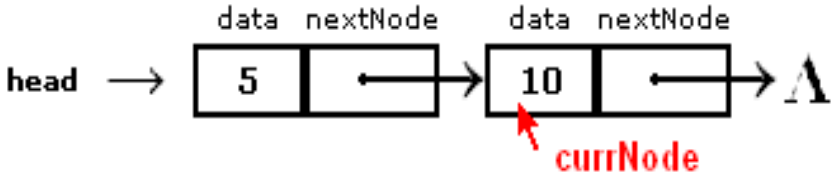
Lista encadeada

(h) while (currNode != null)
condição é verdade

(i) System.out.println(currNode.data);
prints

5

(j) currNode = currNode.nextNode;

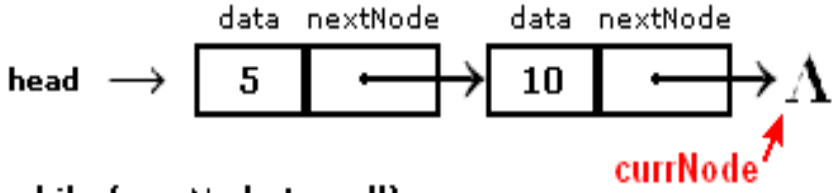


(k) while (currNode != null)
condição é verdade

(l) System.out.println(currNode.data);
prints

10

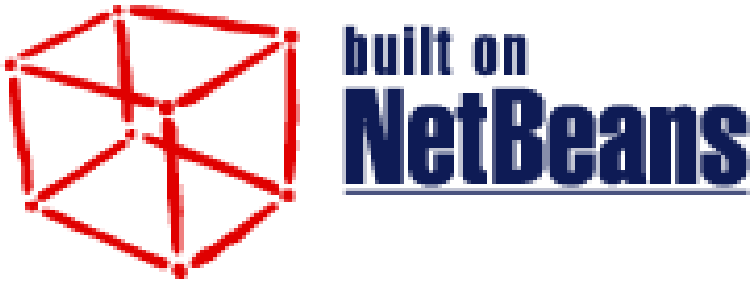
(m) currNode = currNode.nextNode;



(n) while (currNode != null)
/* condição é falsa */

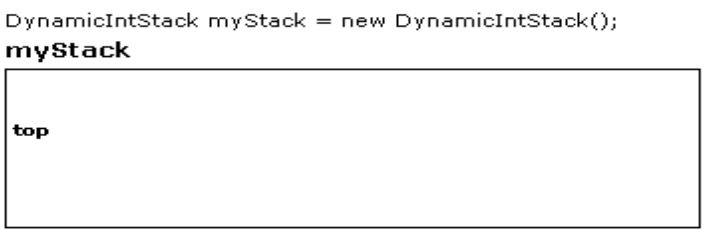
Representação encadeada de uma pilha de inteiros

- Passaremos agora para o NetBeans

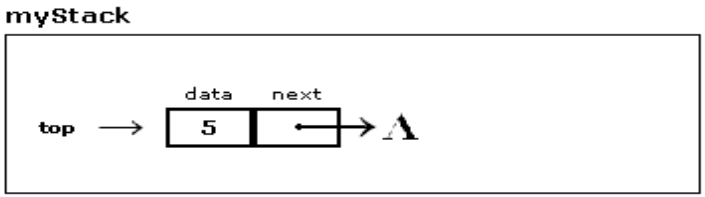


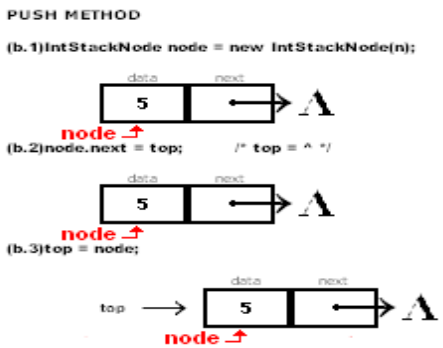
Representação encadeada de uma pilha de inteiros

(a) DynamicIntStack myStack = new DynamicIntStack();

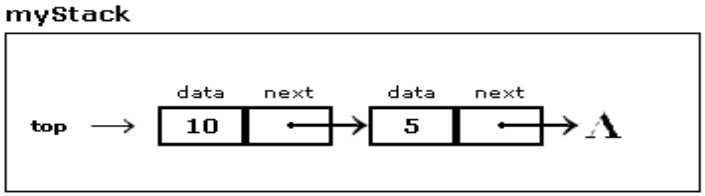


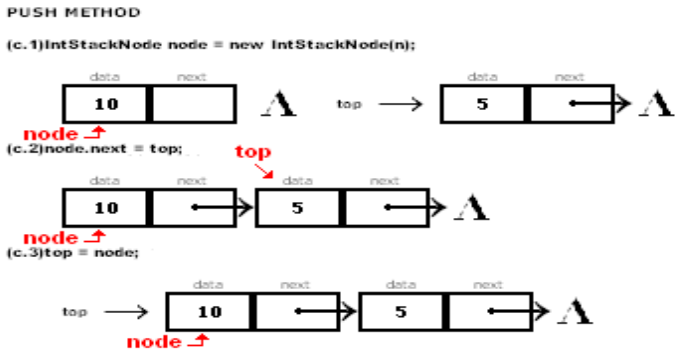
(b) myStack.push(5);





(c) myStack.push(10);



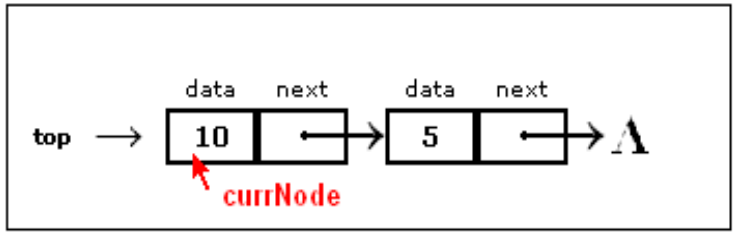


Representação encadeada de uma pilha de inteiros

(d) **imprime os elementos da pilha**

(d.1) IntStackNode currNode = myStack.top;

myStack



(d.2) while (currNode != null)

condição é verdadeira

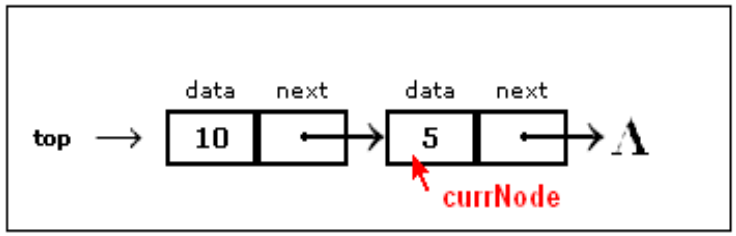
(d.3) System.out.println(currNode.data);

prints

10

(d.4) currNode = currNode.next;

myStack



(d.5) while (currNode != null)

condição é verdadeira

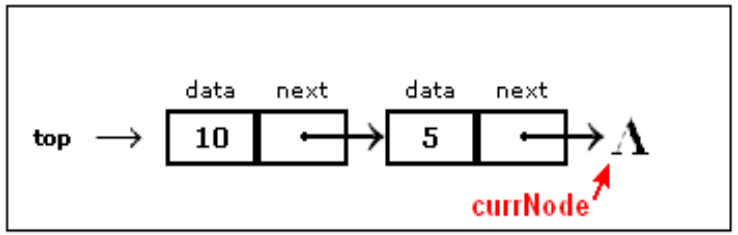
(d.6) System.out.println(currNode.data);

prints

5

(d.7) currNode = currNode.next;

myStack



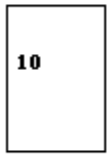
(d.8) while (currNode != null)

/* condição é falsa */

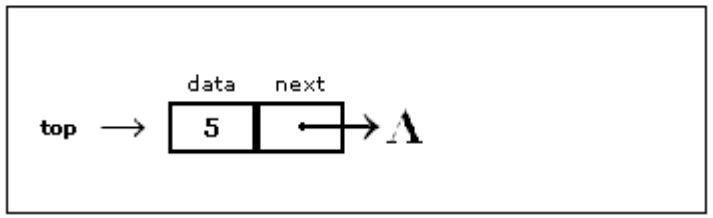
Representação encadeada de uma pilha de inteiros

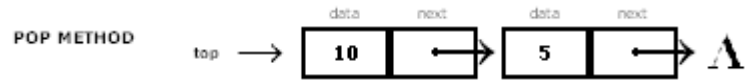
(e) System.out.println(myStack.pop());

prints



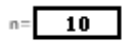
myStack





/* go to else */

(e.1)int n = top.data;



(e.2)top = top.next;

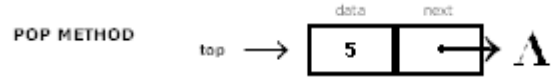
(e.3)return n;

retorna 10 quando é impresso no principal

(f) System.out.println(myStack.pop());

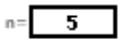
prints

myStack



/* go to else */

(f.1)int n = top.data;



(f.2)top = top.next;

(f.3)return n;

retorna 5 quando é impresso no principal

Java *Collections*

- Java implementa coleções de classes e interfaces

- Exemplos:
  - LinkedList
  - ArrayList
  - HashSet
  - TreeSet

Java *Collections*

- *Coleções de interfaces*
  - Caminho de todas as coleções de interfaces
  - Nenhuma implementação é executada

- Definição de Collection:
  - Grupo de objetos, os quais as vezes são chamados de elementos
  - Pode permitir duplicidade e não exige uma ordem específica

Java *Collections*

- Interface *Set*
  - Coleções não ordenadas que não contém duplicidades
  - Algumas classes implementáveis: *HashSet*, *LinkedHashSet* e *TreeSet*
- Interface *List*
  - Coleções de classes ordenadas onde as duplicidades são permitidas
  - Algumas classes implementadas: *ArrayList*, *LinkedList* e *Vector*

Java *Collections*

- Hierarquia de Java *Collections*

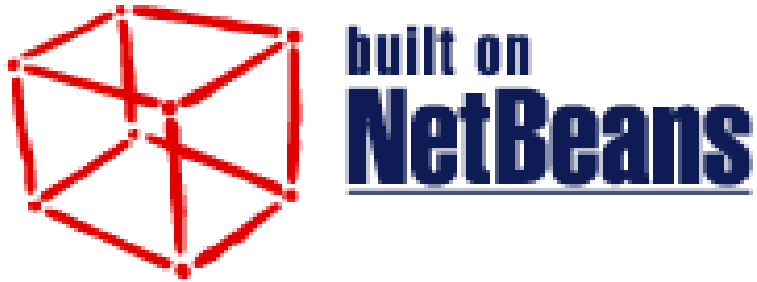
<table>
<tr><td colspan="6"><root interface>
Collection</td></tr>
<tr><td colspan="3"><interface>
Set</td><td colspan="3"><interface>
List</td></tr>
<tr><td colspan="3"><implementing classes></td><td colspan="3"><implementing classes></td></tr>
<tr><td>HashSet</td><td>LinkedHashSet</td><td>TreeSet</td><td>ArrayList</td><td>LinkedList</td><td>Vector</td></tr>
</table>

Métodos de Java *Collection*

```
public boolean add(Object o)
public void clear()
public boolean remove(Object o)
public boolean contains(Object o)
public boolean isEmpty()
public int size()
public Iterator iterator()
```

Java *Collections*: *LinkedList*

- Passaremos agora para o NetBeans



Java *Collections*: *ArrayList*

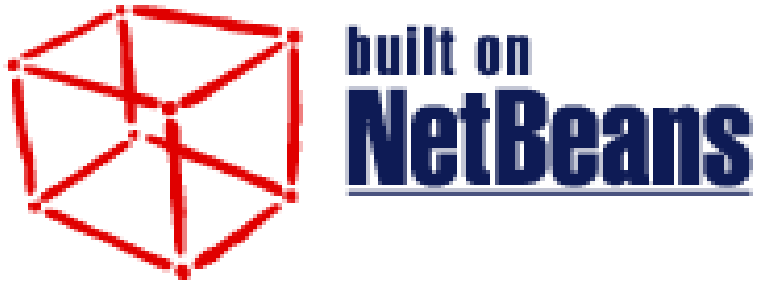
- Um array ordenado e redimensionável que implementa a interface *List*

Java *Collections*: *HashSet*

- Implementação da interface *Set* que usa uma tabela hash
- Tabela Hash
- Benefícios do uso de uma tabela hash

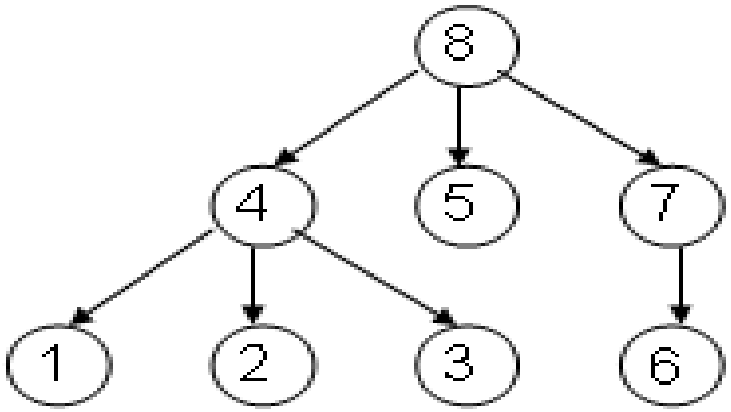
Java *Collections*: *HashSet*

- Passaremos agora para o NetBeans



Java *Collections*: *TreeSet*

- Implementação da interface *Set* que usa uma árvore
- Organizado em ordem ascendente



Java *Collections*: *TreeSet*

- Passaremos agora para o NetBeans

Sumário

- Recursão
  - Definição
  - Recursão versus Iteração
- Tipos de Dados Abstratos
  - Definição, pilhas, filas, representação seqüencial e encadeada
- Java Collections
  - Collection
  - Linked List
  - ArrayList
  - HashSet
  - TreeSet

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**