SCC0204 - Programação Orientada a Objetos

Java Collections

Prof. Jose Fernando Rodrigues **Junior**
http://www.icmc.usp.br/~junio
junio@icmc.usp.br

INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
- USP -

Introdução

- **Set (conjunto):** composição elementos sem repetição e sem ordem definida. Exemplo:
  - A = {banana, maçã, abacate}

- **Multiset (multi-conjunto):** composição de elementos **possivelmente com repetição** e sem ordem definida. Exemplo:
  - A = {banana, maçã, abacate, banana}

  Também denominado: list (lista), grupo (bunch), bag (saco, sacola), heap (monte), sample (amostra), suite (jogo de) e collection (coleção)

- **Sequência:** conjunto de elementos cuja **ordem importa**. Exemplo: {banana, maçã, abacate} ≠ {maçã, banana, abacate}

Introdução

- **Set (conjunto):** composição elementos sem repetição e sem ordem definida. Exemplo:
  - A = {banana, maçã, abacate}

- **Multi**mentos **poss**efinida. Exem
  - A =

Em computação, pela própria definição baseada na máquina de Turing, conjuntos e multiconjuntos possuem alguma ordem – pois estão em memória. Essa ordem pode ser considerada (definindo-se uma sequência) ou não.

Também denominado: list (lista), grupo (bunch), bag (saco, sacola), heap (monte), sample (amostra), suite (jogo de) e collection (coleção)

- **Sequência:** conjunto de elementos cuja **ordem importa**. Exemplo: {banana, maçã, abacate} ≠ {maçã, banana, abacate}

Introdução

- **Conjuntos, multiconjuntos e sequências** são alguns dos **principais conceitos** de toda a matemática, ciência da computação e mesmo cognição humana

- Tratam-se de **conceitos abstratos** (em contraste a conceitos concretos, como um carro por exemplo)

- Em muitos contextos, por praticidade, o termo **Collection** se refere **de maneira geral** a conjuntos, multiconjuntos e sequências (ordered collections)

Collections básicas

A interface Collection

- Collections em Java começam pela interface de mesmo nome – e são genéricas!

```
public interface Collection<E> extends Iterable<E> {
    // Basic operations
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);         //optional
    boolean remove(Object element); //optional
    Iterator<E> iterator();

    // Bulk operations
    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);  //optional
    boolean removeAll(Collection<?> c);         //optional
    boolean retainAll(Collection<?> c);         //optional
    void clear();                               //optional

    // Array operations
    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

As operações da interface Collection são as operações comuns a conjuntos, multi-conjuntos e sequências.

Iteradores

- A idéia de coleções que se baseiam em uma interface raíz e que são genéricas traz **dois problemas**:

  1) a maneira **como se percorre** os elementos da coleção; por exemplo, percorrer um ArrayList (por índice) é diferente de se percorrer um LinkedList (por encadeamento)

  2) não se sabe com antescedência qual o tipo armazenado na coleção, portanto, **ponteiros e aritmética de ponteiros não é possível**

- Solução: **iteradores**
- Iteradores **abstraem como e o quê** está sendo percorrido em uma coleção, permitindo a escrita de código mais geral (que possui por baixo código mais específico)

Iteradores

- A idéia de coleções que se baseiam em uma interface raíz e que são genéricas traz **dois problemas**:

  1) a maneira **como se percorre** os elementos da coleção; por
  dif
  enc

  2) n
  arm
  arit

- Soluç
- Iterad

em uma coleção, permitindo a escrita de código mais geral (que possui por baixo código mais específico)

A solução é ter a interface Iterable<E> como superinterface da interface Collection <E>:

interface Collection <E> extends Iterable<E>

O que requer os seguintes métodos:

- boolean hasNext();
- E next();
- void remove();

Iteradores

A interface Iterator <E>:

- boolean hasNext()

  Returns true if the iteration has more elements.

- E next()

  Returns the next element in the iteration.

- void remove()

  Removes from the underlying collection the last element returned by the iterator (optional operation).

Esses **três métodos devem ser implementados para se definir o iterador de uma coleção genérica**. Mas já estão disponíveis em diversas das coleções do Java.

Arrays e listas dinâmicas

- **Arrays** (arranjos) e **linked lists** (listas encadeadas) são as estruturas mais usadas para problemas de collections – elas **permitem que se tenha conjuntos, multiconjuntos e sequências em memória**
- **Exemplo NetBeans → ListTest**
- Quando usar ArrayList e quando usar LinkedList?
  - **ArrayList**: quando se têm **muito acesso aleatório** (acessos não sequenciais), o que é eficiente por meio de índice de array abstraído pelo iterador, e pouca edição (add e remove)
  - **LinkedList**: quando se tem **muita edição** (add e remove) e **acesso sequencial** dos elementos da lista

Arrays e listas dinâmicas

- Alguns collections muito conhecidos são as estruturas de dados **Pilha** e **Fila**
- Qual é a relação entre arrays, linked lists, pilhas e filas?
  - Arrays e linked lists são apenas conjuntos de dados (ou coleções)
  - Pilhas e filas são conjuntos (ou coleções) com restrições de acesso
  - Portanto:
    - Um array pode ser uma pilha ou uma fila
    - Um linked list pode ser uma pilha ou uma fila
    - Tanto arrays como linked lists são listas (ou conjuntos, ou coleções)
    - Um array não pode ser um linked list, e vice versa
    - Uma pilha não pode ser uma fila, e vice versa
    - Todos são coleções

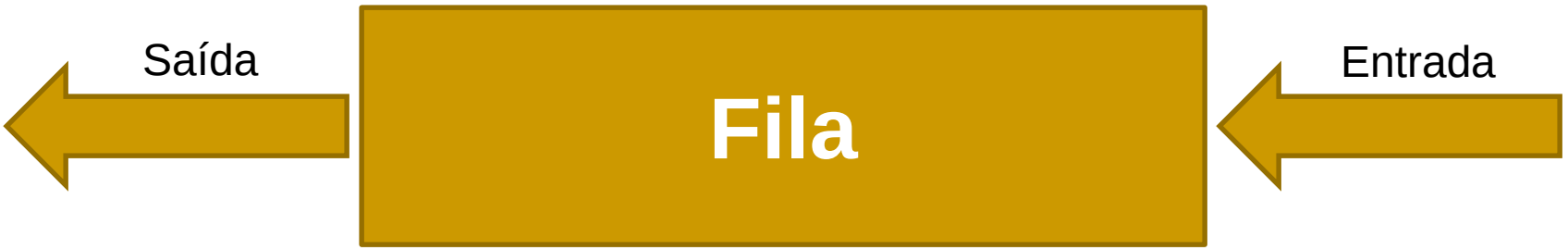
Arrays e listas dinâmicas

- Alguns collections muito conhecidos são as estruturas de dados **Pilha** e **Fila**
- Qual é a relação entre arrays, linked lists, pilhas e filas?
  - Arrays e linked lists são apenas conjuntos de dados (ou coleções)
  - Pilhas e filas são conjuntos (ou coleções) com restrições de acesso
  - Portanto:
    - Um array pode ser uma pilha ou uma fila
    - U
    - T es)
    - Um array não pode ser um linked list, e vice versa
    - Uma pilha não pode ser uma fila, e vice versa
    - Todos são coleções

Estes e outros fatos se refletem na definição e no uso das coleções do Java.

Queue (Fila)

- Fila: trata-se de uma coleção na qual **a entrada só ocorre de um lado e a saída só ocorre do lado oposto** – FIFO, portanto

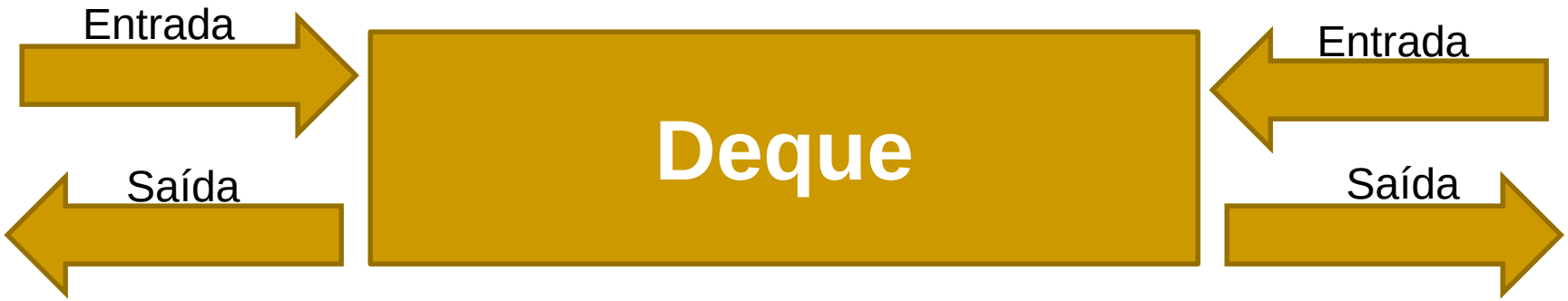


Stack (Pilha)

- Pilha: uma coleção na qual **a entrada e a saída só ocorrem do mesmo lado** – LIFO, portanto

ArrayDeque

- Array que aceita operações de inserção e remoção em ambas as extremidades



- Portanto, um ArrayDeque pode ser:
  - uma **fila** sobre um array: métodos **add** e **remove**
  - uma **pilha** sobre um array: métodos **push** e **pop**

- **Exemplo NetBeans → QueueStackArrayTest**

LiskedList

- Da mesma maneira, um **linked list** aceita naturalmente operações de adição e remoção em ambas as extremidades

- Portanto, uma LinkedList pode ser:
  - uma **fila** sobre uma lista encadeada: métodos **add** e **remove**
  - uma **pilha** sobre sobre uma lista encadeada: métodos **push** e **pop**

- **Exemplo NetBeans → QueueStackLinkedListTest**

Sorting

- Como já mencionado em outra aula, a ordenação de coleções (listas) depende da **definição de uma relação de ordem total** entre seus elementos

- Isso é possível usando-se a interface **Comparable**:

```
public interface Comparable<T> {
    public int compareTo(T o);
}
```

- Uma vez implementada, pode-se ordenar uma dada coleção chamando-se o método estático

```
public static void sort(List<T> list)
```

Sorting

- Os tipos primitivos do Java implementam a interface Comparable em suas respectivas classes de boxing.

Classes Implementing Comparable

Class	Natural Ordering
`Byte`	Signed numerical
`Character`	Unsigned numerical
`Long`	Signed numerical
`Integer`	Signed numerical
`Short`	Signed numerical
`Double`	Signed numerical
`Float`	Signed numerical
`BigInteger`	Signed numerical
`BigDecimal`	Signed numerical
`Boolean`	`Boolean.FALSE < Boolean.TRUE`
`File`	System-dependent lexicographic on path name
`String`	Lexicographic
`Date`	Chronological

Sorting

- Suponha o seguinte problema: **você está usando objetos de uma classe da qual você não tem acesso ao código fonte, e esta classe não implementa a interface Comparable ou implementa diferentemente do que você deseja**.
- Se você quiser ordenar uma lista com os objetos desta classe, como fazer?
  - Para este problema, o Java oferece uma alternativa, a interface Comparator

```
public interface Comparator<T> {
    int compare(T o1, T o2);
}
```

- Torna-se possível a definição de comparadores de objetos ao invés de objetos que são comparáveis entre si.

public static void sort(List<T> list, Comparator<T> c)

- **Exemplo NetBeans → ComparatorTest e ListTest**

Até aqui

Listas (“conjuntos” COM repetição) e SEM restrições (acesso e remoção em posições aleatórias)

- Array: ArrayList<E>
- Lista encadeada: LinkedList<E>

Listas (“conjuntos” COM repetição) e COM restrições (acesso e remoção apenas nas extremidades)

- Array: ArrayDeque
  - Pilha: push/pop
  - Fila: add/remove
- Lista encadeada: LinkedList<E>
  - Pilha: push/pop
  - Fila: add/remove
- class Stack<E>: pilha
- interface Queue<E>: fila

Sets (conjuntos SEM repetição)

Próximos slides

Sets

Set

- A interface Set descende de Collection

  public interface Set<E> extends Collection<E>

- Trata-se de um conjunto, **matematicamente falando**, sem repetição e sem ordem

- Interface base para coleções que **NÃO permitem repetição** de elementos
  - Não pode haver dois elementos e1 e e2 tais que **e1.equals(e2)**
  - Não pode haver dois elementos com valor **null**

Implementações de Set

- Existem três implementações, todas para o mesmo fim, mas com desempenho e requisitos diferentes:
  - **HashSet**: usa um hash para oferecer tempo constante, **idealmente O(1)**, para operações (add, remove, contains e size); requer que equals e hashCode sejam sobreescritos – não mantém nenhuma ordem, melhor desempenho
  - **LinkedHashSet**: semelhante ao HashSet, **usa também uma lista encadeada** para manter a ordem dos elementos de acordo com a ordem de entrada
  - **TreeSet**: requer que a interface Comparable seja implementada – **ordena pelo valor dos elementos** mantendo-os em uma árvore binária de busca (não usa hash) – pior desempenho, mas provê ordenação simplificada
- Como o HashSet evita repetições?

Caso dois elementos possuam o mesmo código hash, prossegue para a comparação com o método equals. Portanto, deve-se sobreescrever tanto o método equals quanto o hashCode

- **Exemplo NetBeans → SetTest**

Até aqui

Listas (“conjuntos” COM repetição) e SEM restrições (acesso e remoção em posições aleatórias)

- Array: ArrayList<E>
- Lista encadeada: LinkedList<E>

Listas (“conjuntos” COM repetição) e COM restrições (acesso e remoção apenas nas extremidades)

- Array: ArrayDeque
  - Pilha: push/pop
  - Fila: add/remove
- Lista encadeada: LinkedList<E>
  - Pilha: push/pop
  - Fila: add/remove
- class Stack<E>: pilha
- interface Queue<E>: fila

Sets (conjuntos SEM repetição)

- HashSet
- LinkedHashSet
- TreeSet

Maps

Map

- Map é um conceito amplamente usado em ciência da computação, refere-se ao mapeamento entre chaves e valores

- Map é parte do arcabouço de coleções, mas que não descende de Collection, mas sim da interface Map

```
public interface Map<K,V>
```

- **Modela o conceito matemático de função, isto é:**
  - as chaves não possuem repetição
  - uma dada chave mapeia exatamente um valor

Map

- A interface

```
public interface Map<K,V> {

   // Basic operations
   V put(K key, V value);
   V get(Object key);
   V remove(Object key);
   boolean containsKey(Object key);
   boolean containsValue(Object value);
     ...

   // Collection Views
   public Set<K> keySet();              //as chaves sem repetição, determinam um conjunto
   public Collection<V> values();       //já os valores, determinam apenas uma coleção
     ...
}
```

Até aqui

Listas (“conjuntos” COM repetição) e SEM restrições (acesso e remoção em posições aleatórias)

- Array: ArrayList<E>
- Lista encadeada: LinkedList<E>

Listas (“conjuntos” COM repetição) e COM restrições (acesso e remoção apenas nas extremidades)

- Array: ArrayDeque
  - Pilha: push/pop
  - Fila: add/remove
- Lista encadeada: LinkedList<E>
  - Pilha: push/pop
  - Fila: add/remove
- class Stack<E>: pilha
- interface Queue<E>: fila

Sets (conjuntos SEM repetição)

- HashSet
- LinkedHashSet
- TreeSet

Mapas (definem o conceito matemático de função*)

- HashMap
- LinkedHashMap
- TreeMap

*com a diferença de que o contra-domínio (valores) não é necessariamente um conjunto

Até aqui

Listas (“conjuntos” COM repetição) e SEM restrições (acesso e remoção em posições aleatórias)

- Array: ArrayList<E>
- Lista encadeada: LinkedList<E>

Listas (“cor
e COM restr
apenas

Sets (con

Mapas (definem o conceito matemático de função*)

- HashMap
- LinkedHashMap
- TreeMap

Os **sets** podem ser considerados casos especiais de **maps** nos quais o valor mapeado por uma chave é o próprio valor da chave.

*com a diferença de que o contra-domínio (valores) não é necessariamente um conjunto