COLEÇÕES

ACH 2003 — COMPUTAÇÃO ORIENTADA A OBJETOS

Daniel Cordeiro

16 de março de 2016

Escola de Artes, Ciências e Humanidades | EACH | USP

AVISO

- Não haverá aula nos dias 23 e 25 de março *semana santa* e dias 30 de março e 1º de abril

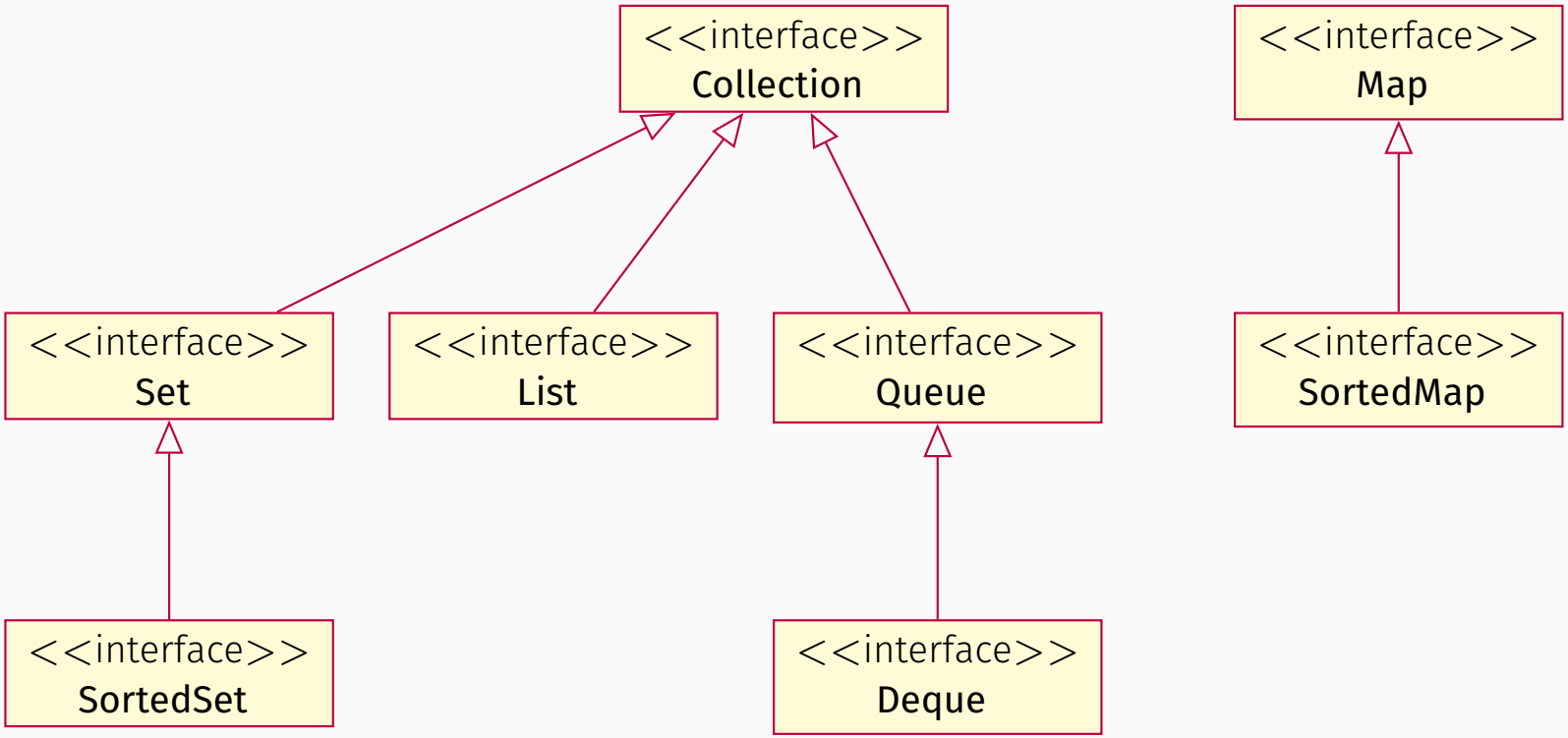
ARCABOUÇO DE COLEÇÕES

Um arcabouço de coleções é uma arquitetura unificada para representação e manipulação de coleções contendo:

Interfaces: tipos abstratos de dados que representam coleções

Implementações: implementações (reutilizáveis) concretas as interfaces de coleções

Algoritmos: métodos que realizam alguma computação útil, tais como busca e ordenação. Esses algoritmos são polimórficos, ou seja, o mesmo método pode ser usado por diferentes implementações das interfaces de coleções.



Principais interfaces de coleções

- `Map` não é exatamente uma `Collection`
- Todas são genéricas: `public interface Collection<E>`

COLLECTION

A interface garante a implementação de operações básicas, tais como:

```
boolean      add(E e)
boolean      addAll(Collection<? extends E> c)
void         clear()
boolean      contains(Object o)
boolean      containsAll(Collection<?> c)
boolean      isEmpty()
Iterator<E>  iterator()
boolean      remove(Object o)
boolean      removeAll(Collection<?> c)
boolean      retainAll(Collection<?> c)
int          size()
<T> T[]      toArray(T[] a)
```

LIST

Uma `List` é uma coleção ordenada (sequência) de elementos que podem ter duplicatas. A interface `List` inclui operações para:

- **Acesso posicional:** manipula elementos baseado nas suas posições numéricas na lista — `get`, `set`, `add`, `addAll` e `remove`
- **Busca:** busca por um objeto particular na lista e devolve sua posição — `indexOf` e `lastIndexOf`
- **Iteração:** estende a semântica do `Iterator` para a natureza sequencial das listas — `listIterator`
- **Visão intervalar:** o método `sublist` realizar operações em intervalos arbitrários da lista

IMPLEMENTAÇÕES DE LIST

A plataforma Java oferece duas implementações de `List`:

ArrayList que geralmente possui melhor desempenho

LinkedList que oferece melhor desempenho em casos específicos

ACESSO POSICIONAL E BUSCAS

- o acesso a posições é realizado geralmente com os métodos `get`, `set`, `add` e `remove`
- `set` e `remove` devolvem o valor sendo sobrescrito ou removido
- o método `addAll` insere todos os elementos de uma coleção, a partir da posição especificada. A ordem dos elementos é definida pelo iterador da coleção

EXEMPLO DE USO

```java
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
```

EXEMPLO DE USO

```java
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
```

Note novamente que esse método é *polimórfico*. Ele funciona com qualquer implementação de listas!

EXEMPLO DE USO

```java
public static <E> void swap(List<E> a, int i, int j) {
    E tmp = a.get(i);
    a.set(i, a.get(j));
    a.set(j, tmp);
}
```

Note novamente que esse método é *polimórfico*. Ele funciona com qualquer implementação de listas!

```java
public static void shuffle(List<?> list, Random rnd) {
    for (int i = list.size(); i > 1; i--)
        swap(list, i - 1, rnd.nextInt(i));
}
```

(algoritmo da classe `Collections`, o que ele faz?)

- O `Iterator` devolvido pelo método `iterator()` de `List` percorre os elementos da lista em ordem
- `ListIterator`, que herda de `Iterator`, fornece os métodos `hasPrevious()` e `previous()` que funcionam analogamente aos métodos `hasNext()` e `next()` de `Iterator`

Uso de `ListIterator`:

```
for (ListIterator<Type> it =
                list.listIterator(list.size());
     it.hasPrevious(); ) {
    Type t = it.previous();
    ...
}
```

(note o argumento que `listIterator()` recebe)

- O cursor do iterador está sempre entre dois elementos — entre aquele que seria devolvido por `previous()` e o que seria devolvido por `next()`
- Os $n + 1$ índices válidos do cursor correspondem às $n + 1$ "lacunas" entre os elementos
- Chamadas a `next()` e `previous()` podem ser intercaladas, mas cuidado que a primeira chamada a `previous()` devolverá o mesmo elemento que a última chamada a `next()`

OPERAÇÕES EM INTERVALOS

- o método `subList(int fromIndex, int toIndex)` devolve uma visão de um pedaço da lista, com os elementos com índices no intervalo [fromIndex; toIndex) (aberto no final)
- essa visão também é uma `List`
- qualquer operação que espera uma `List` pode ser usada com o intervalo devolvido pelo método `subList()`

```
public static <E> List<E> darAsCartas(List<E> baralho, int n) {
    int tamDoBaralho = baralho.size();
    List<E> visãoDasCartas = baralho.subList(tamDoBaralho - n, tamDoBaralho);
    List<E> cartas = new ArrayList<E>(visãoDasCartas);
    visãoDasCartas.clear();
    return cartas;
}
```

ALGORITMOS PARA LISTAS

A classe `Collections` implementa alguns algoritmos específicos para uso com listas. Exemplos:

sort ordena a lista com um algoritmo de ordenação por intercalação (*merge*) — rápido e a ordenação é *estável*

shuffle devolve uma permutação aleatória dos elementos da lista

reverse inverte a ordem dos elementos da lista

rotate rotaciona os elementos de uma determinada distância

swap troca dois elementos da lista

fill sobrescreve todos os elementos da lista com um dado valor

binarySearch realiza uma busca binária em uma lista ordenada

QUEUE

- Uma **Queue** é uma coleção que guarda elementos antes deles serem processados
- Normalmente ordena os elementos em FIFO, mas há exceções importantes como as *filas de prioridades*
- Além das operações de **Collection**, oferece opções de inserção, remoção e inspeção

```
public interface Queue<E> extends Collection<E> {
    E element();
    boolean offer(E e);
    E peek();
    E poll();
    E remove();
}
```

QUEUE

- Os métodos de `Queue` são oferecidos de duas formas:
  1. uma lança uma exceção se a operação falha
  2. outra devolve um valor especial se a operação falha (`null` ou `fail`)

Operação	Lança exceção	Usa valor especial
Inserção	`add(e)`	`offer(e)`
Remoção	`remove()`	`poll()`
Exame	`element()`	`peek()`

QUEUE

- `Queue`s podem definir diferentes ordenações pros elementos
- Toda implementação da interface deve definir essa ordem
- Independentemente da ordem escolhida, a cabeça da fila é o elemento que seria removido por uma chamada a `remove()` ou `poll()`
- `remove()` e `poll()` diferem quando a fila estiver fazia. Nesse caso, o primeiro lança um `NoSuchElementException`, enquanto o segundo devolve `null`
- `element()` e `peek()` devolvem, mas não removem, o elemento na cabeça da fila. Agem, respectivamente, como o `remove()` e `poll()` no caso de fila vazia
- Não insira `null` na fila, pois o valor é utilizado pelos métodos acima com um significado especial

EXEMPLO

```
import java.util.*;

public class Countdown {
    public static void main(String[] args)
                                throws InterruptedException {
        int time = Integer.parseInt(args[0]);
        Queue<Integer> queue = new LinkedList<Integer>();

        for (int i = time; i >= 0; i--)
            queue.add(i);

        while (!queue.isEmpty()) {
            System.out.println(queue.remove());
            Thread.sleep(1000);
        }
    }
}
```

EXEMPLO II

```
static <E> List<E> heapSort(Collection<E> c) {
    Queue<E> queue = new PriorityQueue<E>(c);
    List<E> result = new ArrayList<E>();

    while (!queue.isEmpty())
        result.add(queue.remove());

    return result;
}
```

DEQUE

- Um **Deque** (*double-ended-queue*) é uma coleção linear que permite inserções e remoções de elementos no início e no fim da fila
- Implementa pilha e fila ao mesmo tempo
- É implementado pelas classes `ArrayDeque` e `LinkedList`
- A interface de **Deque** permite a implementação de filas do tipo FIFO e LIFO

	Primeiro elemento (cabeça)		Último elemento (cauda)	
	Lança exceção	*Valor especial*	*Lança exceção*	*Valor especial*
Inserção	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remoção	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Exame	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

BIBLIOGRAFIA

- The Java™ Tutorials – Collections: `https://docs.oracle.com/javase/tutorial/collections/`