FACAMP
FACULDADES DE CAMPINAS

Laboratório de Programação

Aula 6

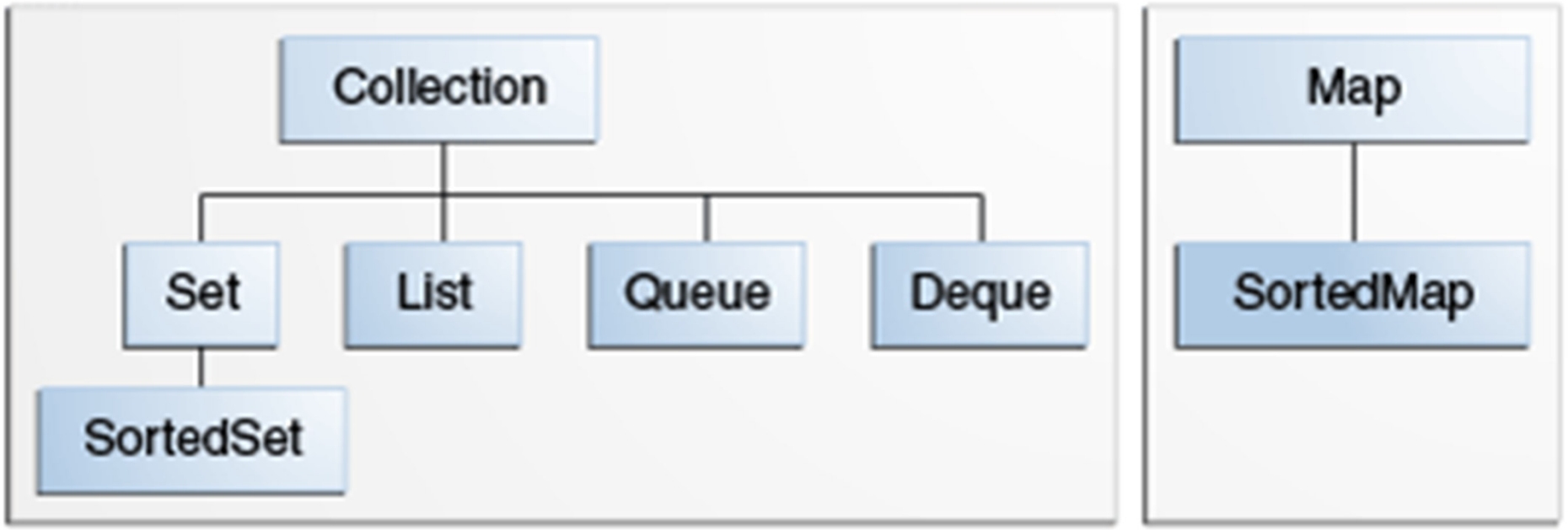
Java Collections, tokens de String e I/O em arquivos texto

2º semestre de 2019
Prof José Martins Junior

Estruturas dinâmicas em Java

- Java não disponibiliza operações com ponteiros
  - Em outras linguagens de programação, como C e Pascal, tal artifício torna-se necessário quando da implementação de estruturas de dados dinâmicas
- Java inclui diversas classes em sua API para solução da maioria dos problemas de armazenamento, organização e recuperação de dados em memória
  - Exemplos (pacote java.util):
    - BitSet, HashMap, HashSet, Hashtable, TreeMap, TreeSet, Vector

Hierarquia de interfaces



Interfaces de alto nível

- Para operação de estruturas de dados dinâmicas, existem coleções (Collection) e mapas (Map)
- Collection e Map são interfaces com subinterfaces que descrevem os tipos principais de estruturas
  - Collection: grupo de elementos/objetos
    - Set – um conjunto de elementos não repetidos
      - SortedSet - conjunto ordenado
    - List – lista sequencial
    - Queue – filas (FIFO)
    - Deque – filas duplas, operam FIFO e LIFO
  - Map: armazena pares chave/valor (as chaves não podem repetir)
    - SortedMap – são ordenados

Principais classes implementadas

- Set
  - AbstractSet, EnumSet, **HashSet**, LinkedHashSet
- SortedSet
  - **TreeSet**
- List
  - AbstractList, **ArrayList**, LinkedList, Stack, Vector
- Queue
  - AbstractQueue, PriorityQueue
- Deque
  - ArrayDeque
- Map
  - AbstractMap, EnumMap, **HashMap**, Hashtable, LinkedHashMap
- SortedMap
  - **TreeMap**

Hashtable e HashMap

- Hash tables são estruturas cujo índice (endereço) de cada elemento (chave) na memória é calculado por uma função de espalhamento (hashing)
  - Normalmente utilizam-se duplas compostas de uma chave e um valor relacionado
  - A busca de cada chave é realizada em tempo constante pelo cálculo do índice através da função de espalhamento
  - Em outras linguagens, como C e Pascal, deve-se estimar uma quantidade de memória suficientemente grande para armazenamento do conjunto, evitando colisões
- Na API Java
  - A classe Hashtable tem é sincronizada e atualmente é obsoleta
  - A classe **HashMap** a substitui com a mesma funcionalidade

TreeMap

- O problema com a HashMap é que a listagem dos elementos não é ordenada
- A TreeMap provê uma estrutura para armazenamento ordenado (pelas chaves) de duplas (chave X valor) relacionadas
- A ordenação natural das chaves é realizada pela interface SortedMap que a classe implementa

ArrayList

- Implementa um array redimensionável sob a interface List
- Permite a adição de qualquer tipo de objeto, inclusive null
- Os elementos podem ser acessados em suas posições, ou sequencialmente, com o uso de um objeto Iterator
- A operação de inserção é realizada em tempo constante – O(n)
  - As demais operações, têm complexidade linear, como uma lista ligada
- A instância de um ArrayList tem uma capacidade inicial que é redimensionada automaticamente

HashSet

- Implementa um conjunto hash sob a interface Set
- Permite a adição de qualquer tipo de objeto, inclusive null
- Os elementos são inseridos de acordo com uma função própria de espalhamento, o que não garante o ordenamento
- Sendo assim, os elementos podem ser procurados (se estão ou não presentes) em tempo constante O(n), ou também listados, com um Iterator

TreeSet

- Implementação um conjunto ordenado, baseado em TreeMap
- Os elementos são colocados em ordem crescente de acordo com o tipo dos dados inseridos
- A implementação garante complexidade log(n) (pior caso) nas operações básicas (inserção, remoção e busca)

Tokens de Strings

- Classe StringTokenizer
  - Permite dividir em pedaços (tokens) um texto (String) delimitado
  - Principais métodos

```
public boolean hasMoreTokens()
public String nextToken()
```

- Exemplo

```
String frase = "Este é uma frase delimitada por espaços";
StringTokenizer strTok = new StringTokenizer(frase, " ");
while(strTok.hasMoreTokens()) {
  System.out.println(strTok.nextToken());
}
```

Leitura de arquivos texto

- Classe FileReader
  - Permite ler o conteúdo de um arquivo texto, cuja localização (diretório + nome) deve ser fornecida como parâmetro
- Exemplo

```
try {
  FileReader fr = new FileReader("C:\\dados\\teste.txt");
  BufferedReader br = new BufferedReader(fr);
  String line = br.readLine();
  while (line != null) {
    System.out.println(line);
    line = br.readLine();
  }
} catch (IOException e) {
  System.out.println("Erro: " + e.getMessage());
}
```

Escrita em arquivos texto

- Classe FileWriter
  - Permite escrever conteúdos em um arquivo texto, cuja localização (diretório + nome) deve ser fornecida como parâmetro
- Exemplo

```
FileWriter fw = null;
try {
  FileReader fr = new FileReader("C:\\dados\\teste.txt");
  fw = new FileWriter("C:\\dados\\teste2.txt");
  BufferedReader br = new BufferedReader(fr);
  String line = br.readLine();
  while (line != null) {
    fw.write(line + "\n");
    line = br.readLine();
  }
} catch (IOException e) { System.out.println(e.getMessage());
} finally {
  try {
    fw.close();
  } catch (IOException e1) { }
}
```

Bibliografia

DEITEL, P.; DEITEL, H. Java TM: como programar. 8ª edição. São Paulo: Pearson Prentice Hall, 2012. 1144p.

GOSLING, J.; ARNOLD, K.; HOLMES, D. A Linguagem de Programação Java. 4ª edição, Porto Alegre: Bookman, 2007.