API

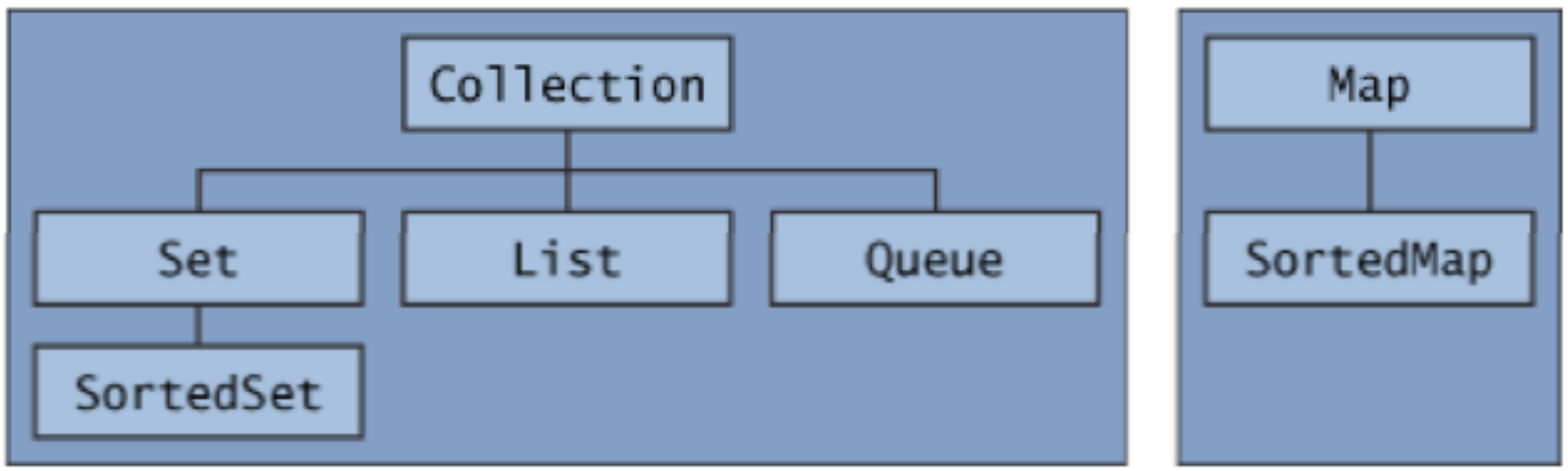
Agenda

- Coleções
- File I/O

Coleções

O Java Collections Framework é a composição é uma arquitetura unificada para representar e manipular coleções e contém:

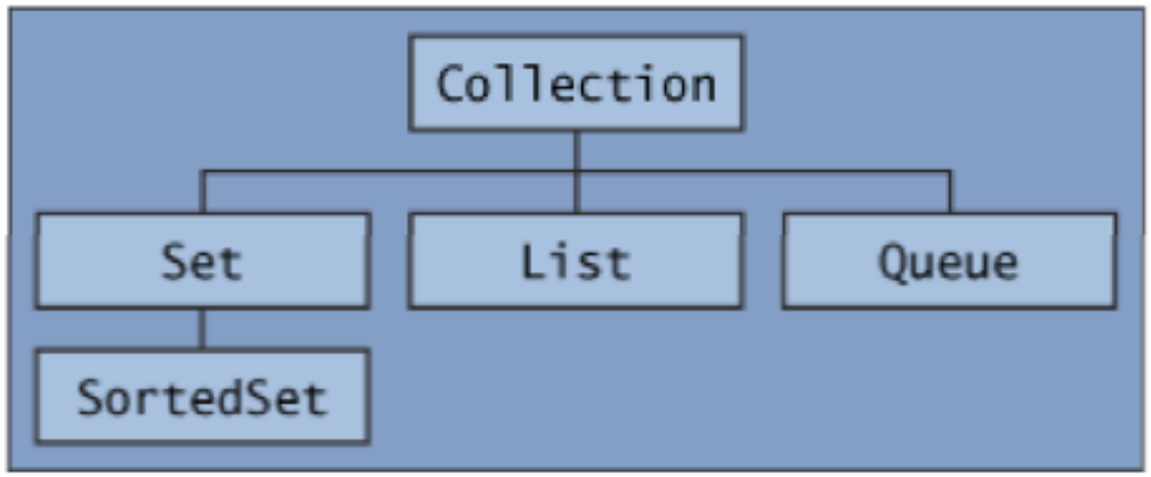
- Interfaces
- Implementações
- Algoritmos



Coleções

Uma Collection representa um grupo de objetos chamados de elementos.

A interface Collection é o último denominador comum para todas as implementações de coleções e é utilizada como argumento quando é necessário o máximo de generalização.

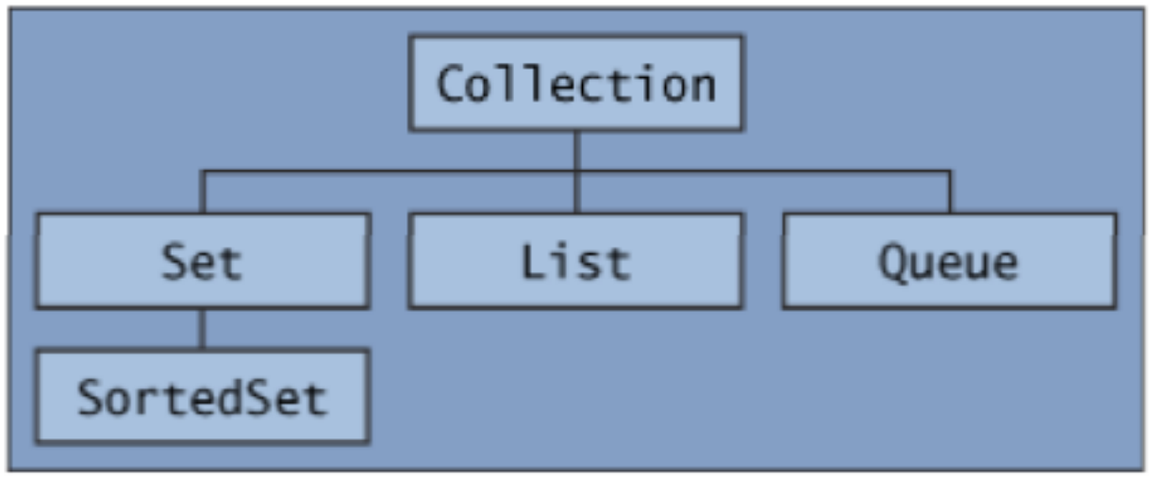


Coleções

Alguns tipo de coleções permitem duplicidade de elementos, e outros não.

Alguns são ordenados e outros não.

As mais específicas sub-interfaces são Set, List e Queue.



Collection

A Interface Collection tem vários métodos declarados e desta forma são comuns a todas as suas implementações, seja List Set ou Queue.

Lista de alguns métodos de Collections

Métodos	Descrição
size()	Quantos elementos estão contidos na coleção
isEmpty()	true se a coleção estiver vazia
contains()	Verifica se um objeto está na coleção
add()	Adiciona um elemento à coleção
remove()	Remove um elemento à coleção
clear()	Remove todos os elementos de uma coleção
toArray()	Retorna um array dos elementos contidos na coleção
stream()	Retorna um objeto que suporta operações de agregação sequencial ou paralelo com os elementos da coleção

Set

O Set é uma coleção que não pode conter duplicidade de elementos.

Esta interface modela a abstração matemática de um set, tais como um conjunto de cartas em um jogo de truco, as disciplinas de um curso superior.

```java
// Declara um set de Strings
Set<String> lista = new TreeSet<>();

// Inclui objetos no set
lista.add("abc");
lista.add("def");
lista.add("fgh");

// Ordena o set
// A implementação TreeSet ordena automaticamente na
// inserção e não permite duplicidade de chave

// Apresenta o set com foreach
for (String txt : lista)
    System.out.println(txt);

// Testa a existência do objeto no set
if(lista.contains("def"))
    // remove o objeto do set
    lista.remove("def");

// Apresenta o set com Stream
lista.stream().forEach(txt -> System.out.println(txt));
```

List

O List é uma coleção ordenada (as vezes chamada de seqüência).

List pode conter elementos duplicados.

Ao adicionarmos um elemento em List é associado um índice a este elemento (sua posição).

As implementações mais utilizadas de List são ArrayList e Vector.

```java
// Declara uma lista de Strings
List<String> lista = new ArrayList<>();

// Inclui objetos na lista
lista.add("abc");
lista.add("def");
lista.add("fgh");

// Ordena a lista
Collections.sort(lista);

// Apresenta a lista com foreach
for (String txt : lista)
    System.out.println(txt);

// Testa a existência do objeto na lista
if(lista.contains("def"))
    // remove o objeto da lista
    lista.remove("def");

// Apresenta a lista com Stream
lista.stream().forEach(txt -> System.out.println(txt));
```

Queue

O Queue é uma coleção que armazena elementos em uma FIFO (primeiro que entra é o último que sai).

Numa fila FIFO, todos os novos elementos são adicionados ao final da fila.

Métodos declarados em Queue

Métodos	Descrição
remove() e poll()	Remove e retorna o elemento no topo da fila
element() e peek()	Retorna o elemento no topo da fila sem removê-lo
offer()	Usado somente em Queues com limites, adiciona elementos

```java
// Declara uma Fila de Strings
Queue<String> lista = new PriorityQueue<>();

// Inclui objetos na Fila
lista.add("abc");
lista.add("def");
lista.add("fgh");

// Ordena a lista
// A implementação de PriorityQueue ordena
// automaticamente na inserção de objetos

// Apresenta a Fila com foreach
for (String txt : lista)
    System.out.println(txt);

// Testa a existência do objeto na Fila
if(lista.contains("def"))
    // remove o objeto da Fila
    lista.remove("def");

// Apresenta a Fila com Stream
lista.stream().forEach(txt -> System.out.println(txt));

// Retira os objetos da Fila
for (int i = lista.size(); i > 0; i--)
    System.out.println(lista.poll());
```

Map

O Map é um objeto que associa valores a chaves.

Um Map não pode conter chaves duplicadas.

Cada chave só pode estar associada a um valor.

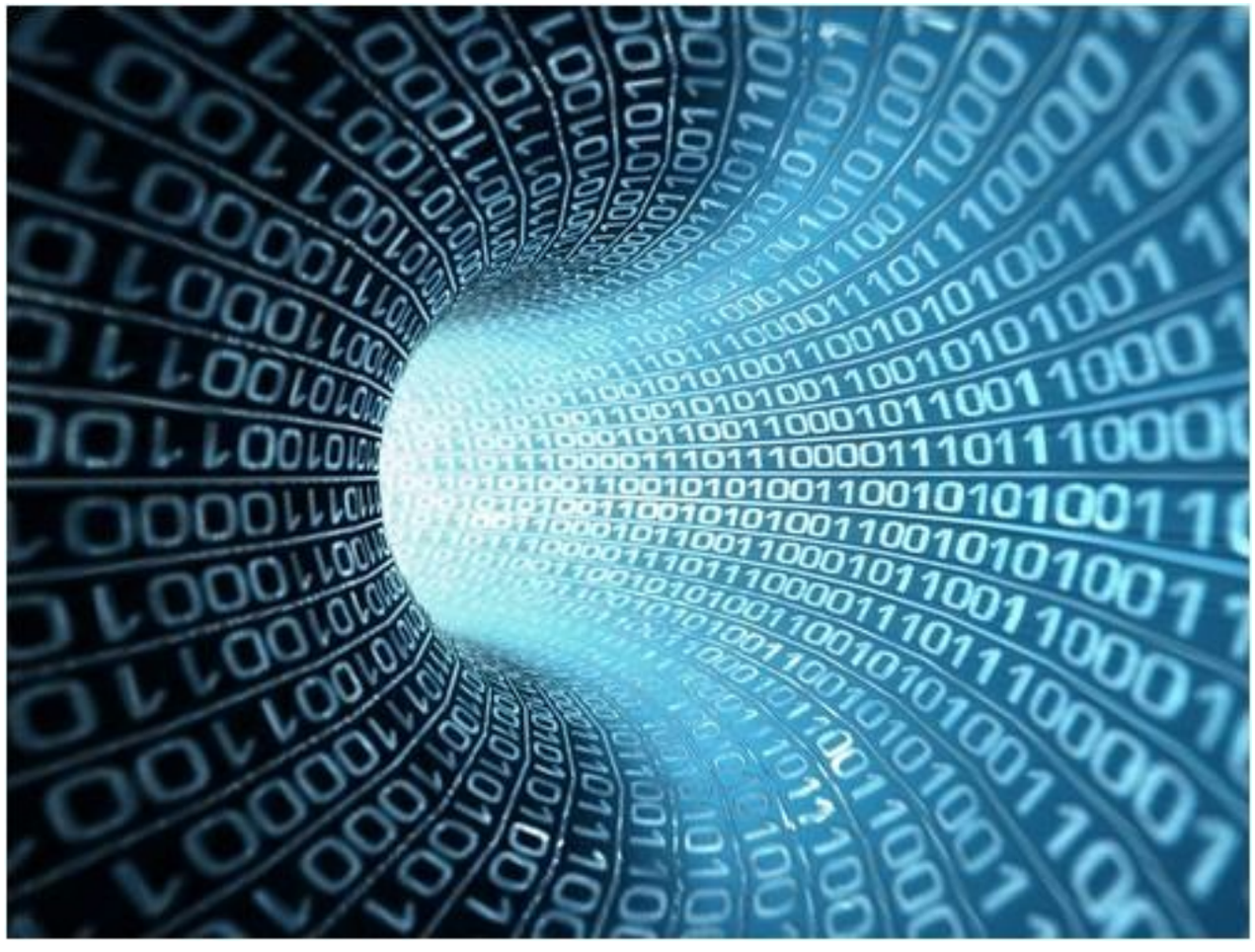
Métodos declarados em Map

Métodos	Descrição
put()	Adiciona chaves e valores ao Map
get()	Retorna o elemento associado a chave informada
containsKey()	Retorna true se a chave existe no Map
containsValue()	Retorna true se o valor existe no Map
values()	Retorna uma Collection dos elementos contidos no Map

```
// Declara um Mapa de Strings
Map<String, String> lista = new TreeMap<>();

// Inclui objetos no Mapa
lista.put("chave1", "abc");
lista.put("chave2", "def");
lista.put("chave3", "fgh");

// Ordena a lista
// A implementação de TreeMap ordena automaticamente
// na inserção de objetos pela chave e não permite
// duplicidade da chave e subistitui o valor caso
// isto aconteça

// Apresenta os valores do Mapa com foreach
for (String txt : lista.values())
    System.out.println(txt);

// Testa a existência do objeto no Mapa pela chave
if(lista.containsKey("chave2"))
    // remove o objeto do Mapa pela chave
    lista.remove("chave2");

// Apresenta os valores do Mapa com Stream
lista.values().stream().forEach(txt -> System.out.println(txt));
```

File I/O

Antes de falar a respeito de como efetuar gravações de arquivos em Java é necessário definir **Streams**.



Streams são uma abstração de baixo nível para a comunicação de dados em Java.

Um **Stream** representa um ponto num canal de comunicação.

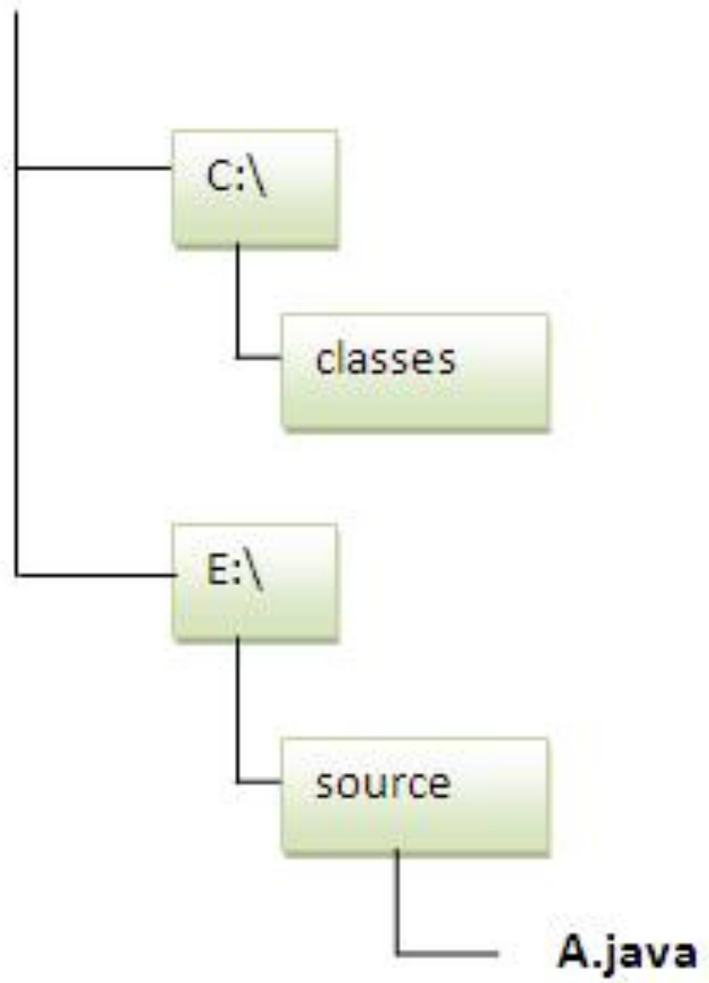
File I/O

Os **Streams** são uma fila do tipo ***FIFO***.

Isto significa que os primeiros bytes escritos num **OutputStream** serão os primeiros a serem lidos de um **InputStream**.

File I/O

A classe **File** representa o nome de um arquivo independente da arquitetura do Sistema Operacional.



File I/O

A class **File** oferece vários métodos para determinar informações sobre arquivos e diretórios, bem como permitindo a modificação de seus atributos.

Método	Descrição
canRead()	Retorna true se o arquivo pode ser lido
canWrite()	Retorna true se o arquivo pode ser gravado
delete()	Retorna true se obtiver sucesso na deleção do arquivo
exists()	Retorna true se o arquivo existir
getName()	Retorna o nome do arquivo sem o diretório
getPath()	Retorna o nome do arquivo com a parte do diretório
getDirectory()	Retorna o nome do diretório onde o arquivo reside
isFile()	Retorna true se for um arquivo
isDirectory()	Retorna true se for um diretório
length()	Retorna o tamanho do arquivo
list()	Retorna um array de Strings contendo os nomes dos arquivos e sub-diretórios contidos num diretório
mkdir()	Retorna true se conseguir criar o diretório
renameTo()	Retorna true se conseguir renomear o arquivo

File I/O

As classes **FileOutputStream**, **FileWriter** e **PrintWriter** são utilizadas para a gravação de dados em arquivos.

```
try {
        // Declara um objeto do tipo File
        File fl = new File("c:/User/Fulano/Desktop/meuDoc.txt");

        // Abre o arquivo para gravação
        FileOutputStream fo = new FileOutputStream(fl);

        // Grava um texto no arquivo
        String txt = "Texto de exemplo 1\n";
        // Transforma a String em array de bytes
        fo.write(txt.getBytes());

        // Fecha o arquivo
        fo.close();

        // Abre o arquivo para acrecentar mais textos
        FileWriter fw = new FileWriter(fl, true);

        // Grava um texto
        fw.write("Texto de Exemplo 2\n");

        // Cria um PrintWriter a partir do FileWriter
        PrintWriter pw = new PrintWriter(fw);

        // Grava um novo texto
        pw.println("Texto de Exemplo 3");

        // Fecha o arquivo
        fw.close();

} catch(FileNotFoundException ex) {
        ex.printStackTrace();
} catch(IOException ex) {
        ex.printStackTrace();
}
```

File I/O

As classes **FileInputStream**, **FileReader** e **BufferReader** são utilizadas para a leitura de dados em arquivos.

```
try {
        // Declara um objeto do tipo File
        File fl = new File("c:/User/Fulano/Desktop/minhaImagem.dat");

        // Abre o arquivo para leitura
        FileInputStream fi = new FileInputStream(fl);

        // Reserva a área de leitura
        byte[] buffer = new byte[1024];
        // Lê 1024 bytes do arquivo
        fi.read(buffer);
        // Fecha o arquivo
        fi.close();

        // Abre o arquivo para leitura
        FileReader fr = new FileReader("C:/meusDados/lista.txt");

        // Reserva a área de leitura
        char[] texto = new char[1024];
        // Lê uma sequencia de caracteres do arquivo
        fr.read(texto);

        // Cria um PrintWriter a partir do FileWriter
        BufferedReader br = new BufferedReader(fr);

        // Lê todas as linhas restante do arquivo
        String linha = br.readLine();

        while(linha != null) {
                System.out.println(linha);
                linha = br.readLine();
        }

        // Fecha o arquivo
        fr.close();
} catch(FileNotFoundException ex) {
        ex.printStackTrace();
} catch(IOException ex) {
        ex.printStackTrace();
}
```

File I/O

As classe **ObjectOutputStream** é utilizada para a gravação de objetos em arquivos.

Um objeto para que possa ser *serializado* (gravado) deve implementar a interface **Serializable**.

```
try {
        // Cria um objeto para ser gravado
        List<String> lista = new ArrayList<String>();

        // Adiciona objetos na lista
        lista.add("Texto");
        lista.add("java");
        lista.add("Novo");

        // Cria um arquivo para gravação
        FileOutputStream fo = new FileOutputStream("meusObjetos.dat");

        // Cria um ObjectOutputStream a partir do FileOutputStream
        ObjectOutputStream objOut = new ObjectOutputStream(fo);

        // Grava o objeto
        objOut.writeObject(lista);

        // fecha o arquivo
        objOut.close();
} catch(FileNotFoundException ex) {
        ex.printStackTrace();
} catch(IOException ex) {
        ex.printStackTrace();
}
```

File I/O

As classe **ObjectInputStream** é utilizada para a leitura de objetos em arquivos.

```
try {

        // Abre um arquivo para leirura
        FileInputStream fi = new FileInputStream("meusObjetos.dat");

        // Cria um ObjectInputStream a partir do FileInputStream
        ObjectInputStream objIn = new ObjectInputStream(fi);

        // Lê o objeto
        List<String> lista = (List<String>)objIn.readObject();

        // fecha o arquivo
        objIn.close();

        // Exibe os objetos lidos
        for(String txt : lista) {
                System.out.println(txt);
        }

} catch(ClassNotFoundException ex) {
        ex.printStackTrace();
} catch(FileNotFoundException ex) {
        ex.printStackTrace();
} catch(IOException ex) {
        ex.printStackTrace();
}
```

Referências

- Programando em Java2 - Teoria & Aplicações
  Rui Rossi dos Santos - Axcel Books - 2004

- Core Java2 - Volume I - Fundamentos
  Cay S. Horstmann & Gary Cornell - The Sun Microsystems Press - Série Java - 2003

- Java Programming
  Nick Clements, Patrice Daux & Gary Williams - Oracle Corporation - 2000