


Laboratório de Engenharia de Software

INF1636 – PROGRAMAÇÃO ORIENTADA A OBJETOS

Departamento de Informática – PUC-Rio

Ivan Mathias Filho
ivan@inf.puc-rio.br


Programa – Capítulo 15




Laboratório de Engenharia de Software

- Coleções
- A Interface List
- A Classe LinkedList
- Exercício
- A Interface Set
- A Classe HashSet
- A Classe TreeSet
- A Interface Map
- A Classe HashMap

© LES/PUC-Rio

Laboratório de Engenharia de Software	Programa – Capítulo 15	
	<ul style="list-style-type: none">• Coleções• A Interface <code>List</code>• A Classe <code>LinkedList</code>• Exercício• A Interface <code>Set</code>• A Classe <code>HashSet</code>• A Classe <code>TreeSet</code>• A Interface <code>Map</code>• A Classe <code>HashMap</code>	
© LES/PUC-Rio		

Laboratório de Engenharia de Software	Coleções – introdução	
	<ul style="list-style-type: none">• A linguagem Java possui uma biblioteca que implementa várias estruturas de dados padrão, tais como lista encadeada, árvore binária, pilha e etc.;• Tais estruturas agrupam múltiplos elementos em um único objeto, com o objetivo de armazenar, recuperar e manipular dados agregados;• As coleções nas versões Java pré-1.2 incluíam <code>Vector</code>, <code>Hashtable</code> e <code>Array</code>, embora ainda não formassem um framework de coleções.	
© LES/PUC-Rio		

Framework de Coleções



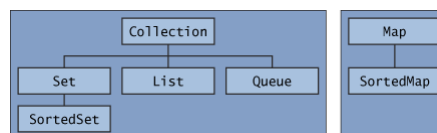
- Um framework de coleções é uma arquitetura unificada para representar e manipular coleções. Ele possui os seguintes elementos:
 - **interfaces:** tipos abstratos de dados que representam coleções. Permitem que as coleções sejam manipuladas independentemente de sua representação;
 - **implementações:** implementações concretas das interfaces de coleção. São estruturas de dados reusáveis.

© LES/PUC-Rio

Interfaces (1)



- Formam uma hierarquia de diferentes tipos de coleções;
- Permitem a manipulação de dados independentemente de sua representação.



- Set é um tipo especial de Collection, SortedSet é um tipo de Set e etc. Note que a hierarquia consiste de duas árvores separadas: Map e Collection .

© LES/PUC-Rio

Interfaces (2)



- Todas as interfaces de coleção são genéricas:
 - `public interface Collection<E>...`
 - `<E>` significa que a interface é genérica.
- Quando uma instância de `Collection` é declarada, deve-se especificar o tipo de objeto que a coleção irá armazenar:
 - `List<String> list = new ArrayList<String>();`
- Isto permite verificar em tempo de compilação se o tipo de objeto a ser armazenado na coleção é compatível com o tipo da coleção.

© LES/PUC-Rio

Interfaces (3)




- **Collection:**
 - Raiz da hierarquia de coleções;
 - Usada quando for necessária genericidade máxima;
 - Java não provê implementações diretas para esta interface.
- **Set:**
 - Coleção que não pode conter elementos duplicados;
 - Modela a abstração matemática de conjuntos.
- **SortedSet:**
 - Conjunto (`Set`) com elementos ordenados ascendentemente.

© LES/PUC-Rio

Laboratório de Engenharia de Software

Interfaces (4)




- **List:**
 - Representa uma coleção não ordenada de elementos;
 - Pode conter elementos duplicados;
 - Pode-se controlar a posição em que um elemento é inserido;
 - Pode-se acessar um elemento usando-se um índice;
 - Corresponde à noção de array.
- **Queue:**
 - Representa uma coleção de elementos com prioridades associadas;
 - Provê operações de inserção, extração e inspeção.
 - Tipicamente organiza os elementos no esquema **FIFO**.

© LES/PUC-Rio

Laboratório de Engenharia de Software

Interfaces (5)



- **Map:**
 - Mapeia chaves e valores;
 - Não pode conter chaves duplicadas;
 - Corresponde ao uso de Hashtables.
- **SortedMap:**
 - Um Map com chaves em ordem ascendente.

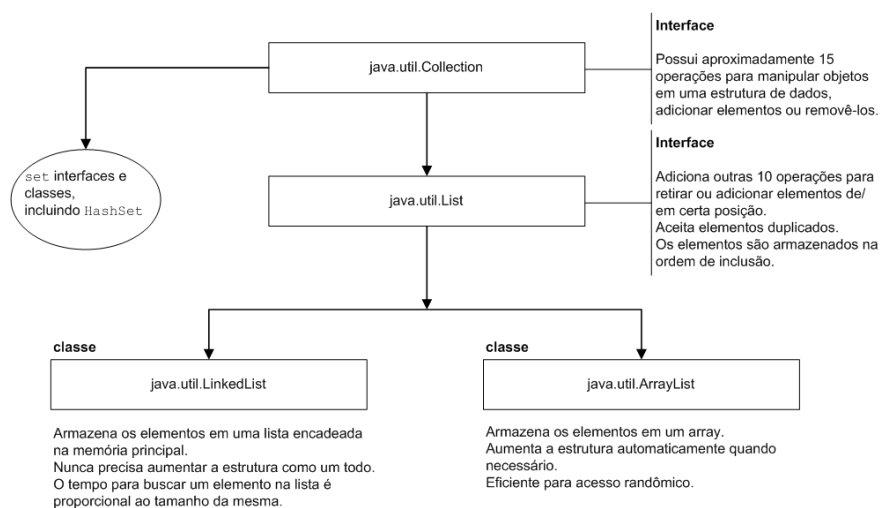
© LES/PUC-Rio

Programa – Capítulo 15



- Coleções
- **A Interface List**
- A Classe LinkedList
- Exercício
- A Interface Set
- A Classe HashSet
- A Classe TreeSet
- A Interface Map
- A Classe HashMap

A Interface List (1)



A Interface List (2)



- A interface List adiciona cerca de 10 operações às operações elementares encontradas em Collection:

```
public interface List<E> extends Collection<E> {
    public boolean addAll(int index, Collection<? extends E> c);
    public void add(int index, E element);
    public E get(int index) ;
    public E set(int index, E element);
    public E remove(int index);
    public int indexOf(Object o);
    public int lastIndexOf(Object o);

    public ListIterator<E> listIterator();
    public ListIterator<E> listIterator(int index) ;
    public List<E> subList(int from, int to);
}
```

© LES/PUC-Rio

A Interface List (3)



- A operação subList() retorna uma nova lista, cujos elementos referenciam os elementos da lista original, incluindo o from e excluindo o to;
- Foram incluídas duas operações para a obtenção de iteradores:
 - listIterator() – obtém um iterador começando com o primeiro elemento da coleção;
 - listIterator(int index) – obtém um iterador começando no elemento definido pelo parâmetro index.

© LES/PUC-Rio

A Interface ListIterator (1)



- A interface `ListIterator` fornece as seguintes operações:

```
public interface ListIterator<E> extends Iterator<E> {
    public boolean hasNext();
    public E next();
    public boolean hasPrevious();
    public E previous();

    public int nextIndex();
    public int previousIndex();

    public void add(E e);
    public void remove();
    public void set(E e);
}
```

© LES/PUC-Rio

A Interface ListIterator (2)



- Um `ListIterator` permite o percurso de uma lista de objetos em ambos os sentidos;
- Para posicionar um iterador no final da lista deve-se passar `list.size()` como parâmetro para a operação `listIterator(int index)`;
- Ao contrário de um `Iterator`, um `ListIterator` não possui um elemento corrente;
- O cursor fica entre os elementos `c.previous()` e `c.next()`.

© LES/PUC-Rio

A Interface `ListIterator` (3)



- As operações `remove()` e `set()` se aplicam aos elementos retornados pelo `next()` ou `previous()` mais recente:
 - A operação `set(E e)` substitui tal elemento pelo objeto passado como argumento (`e`);
 - A operação `remove()` remove da lista o objeto em questão.

Programa – Capítulo 15



- Coleções
- A Interface `List`
- **A Classe `LinkedList`**
- Exercício
- A Interface `Set`
- A Classe `HashSet`
- A Classe `TreeSet`
- A Interface `Map`
- A Classe `HashMap`

A Classe LinkedList (1)



- Classe concreta que implementa a interface `List`;
- Implementa uma lista encadeada que pode ser percorrida em ambos os sentidos;
- Tem bom desempenho nos percursos sequenciais;
- Não é uma boa opção quando os acessos randômicos são preponderantes.

© LES/PUC-Rio

A Classe LinkedList - Exemplo



```
import java.util.*;

public class EX15_01 {
    public static void main(String[] args) {
        List<String> cs=new LinkedList<String>();
        ListIterator<String> li;

        cs.add("Brasil");
        cs.add("Argentina");
        cs.add("Paraguai");
        cs.add("Uruguai");


        li=cs.listIterator(cs.size());

        while(li.hasPrevious())
            System.out.println(li.previous());
    }
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

Programa – Capítulo 15




- A Interface `List`
- A Classe `LinkedList`
- **Exercício**
- A Interface `Set`
- A Classe `HashSet`
- A Classe `TreeSet`
- A Interface `Map`
- A Classe `HashMap`

© LES/PUC-Rio

Laboratório de Engenharia de Software

A Classe `LinkedList` - Exercício



- Implemente as classes `Ponto` e `Polígono` usando uma `LinkedList`;
- A classe `Polígono` deve conter um método `double` `perimetro()`. Implemente-o.

```
public class Ponto {  
    private double x,y;  
  
    public Ponto(double a,double b) {  
        x=a;  
        y=b;  
    }  
  
    public double dist(Ponto p) {  
        //complete o método  
    }  
}
```

© LES/PUC-Rio

Exercício – Resposta (1)



Laboratório de Engenharia de Software

```

public class Ponto {
    private double x,y;

    public Ponto(double a,double b) {
        x=a;
        y=b;
    }

    public double dist(Ponto p) {
        double a,b;

        a=this.x-p.x;
        a*=a;
        b=this.y-p.y;
        b*=b;
        return Math.sqrt(a+b);
    }
}

```

© LES/PUC-Rio

Exercício – Resposta (2)



Laboratório de Engenharia de Software

```

import java.util.*;

public class Poligono {
    private LinkedList<Ponto> v=new LinkedList<Ponto>();

    public void insVertice(Ponto p) {
        v.add(p);
    }

    public double perimetro() {
        double acum=0.0;
        ListIterator<Ponto> li=v.listIterator();
        Ponto p0=null,p1=null,p2=null;
        p1=li.next();
        p0=p1;
        while(li.hasNext()) {
            p2=li.next();
            acum+=p1.dist(p2);
            p1=p2;
        }
        acum+=p0.dist(p2);
        return acum;
    }
}

```

© LES/PUC-Rio

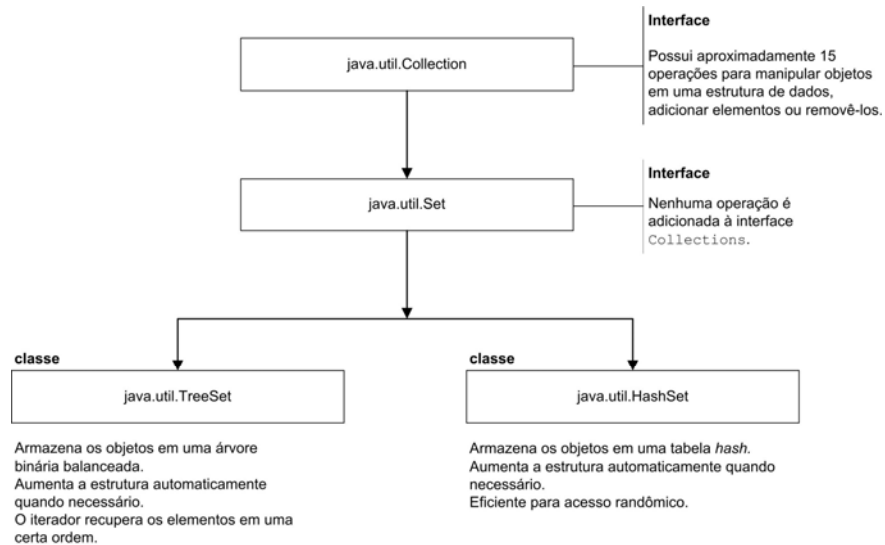
Programa – Capítulo 15



- Coleções
- A Interface `List`
- A Classe `LinkedList`
- Exercício
- **A Interface `set`**
- A Classe `HashSet`
- A Classe `TreeSet`
- A Interface `Map`
- A Classe `HashMap`

© LES/PUC-Rio

A Interface `set` (1)



© LES/PUC-Rio

A Interface Set (2)



- A interface Set não adiciona nenhuma operação à interface Collection. De forma simplificada, elas são as seguintes:

```
public interface Set<E> extends Collection<E> {
    public int size();
    public boolean isEmpty();
    public boolean contains(Object o);
    public boolean add(E e);
    public boolean remove(Object o);
    public Iterator<E> iterator();

    public boolean addAll(Collection<? extends E> c);
    public boolean removeAll(Collection<?> c);
    public boolean retainAll(Collection<?> c);
    public boolean containsAll(Collection<?> c);
    public void clear();
}
```

© LES/PUC-Rio

Programa – Capítulo 15



- Coleções
- A Interface List
- A Classe LinkedList
- Exercício
- A Interface Set
- **A Classe HashSet**
- A Classe TreeSet
- A Interface Map
- A Classe HashMap

© LES/PUC-Rio

A Classe HashSet (1)



- Classe concreta que implementa a interface Set;
- Os objetos inseridos na coleção são armazenados em uma tabela *hash*;
- O programador deve apenas inserir os objetos; o HashSet se encarrega de colocá-los no local adequado;
- Caso um objeto já esteja armazenado na tabela, ele não será adicionado uma segunda vez.

A Classe HashSet – Exemplo (1)



```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Set<Ponto> ss=new HashSet<Ponto>();

        ss.add(new Ponto(0,0));
        ss.add(new Ponto(0,0));
        ss.add(new Ponto(0,0));
        ss.add(new Ponto(0,0));

        for(Ponto e:ss)
            System.out.println(e.getCoord());
    }
}
```

A Classe HashSet - Exemplo (2)



- O programa anterior exibirá na console o seguinte:

```
0.0 0.0
0.0 0.0
0.0 0.0
0.0 0.0
```

- Isso ocorre porque, embora os pontos tenham as mesmas coordenadas, eles são objetos distintos;
- Isto é, são objetos com o mesmos valores, embora não sejam o mesmo objeto.

© LES/PUC-Rio

A Classe HashSet - Exemplo (3)



```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Set<Ponto> ss=new HashSet<Ponto>();
        Ponto p=new Ponto(0,0);


        ss.add(p);
        ss.add(p);
        ss.add(p);
        ss.add(p);

        for(Ponto e:ss)
            System.out.println(e.getCoord());
    }
}
```

© LES/PUC-Rio

Laboratório de Engenharia de Software

A Classe `HashSet` - Exemplo (4)




- O programa anterior exibe na console o seguinte :

```
0.0 0.0
```
- Isso irá ocorrer porque tentamos inserir o mesmo objeto mais de uma vez;
- Como um `HashSet` é um conjunto, as duplicatas não são adicionadas à coleção.

© LES/PUC-Rio

Laboratório de Engenharia de Software

Programa – Capítulo 15



- Coleções
- A Interface `List`
- A Classe `LinkedList`
- Exercício
- A Interface `Set`
- A Classe `HashSet`
- **A Classe `TreeSet`**
- A Interface `Map`
- A Classe `HashMap`

© LES/PUC-Rio

A Classe TreeSet (1)



- Classe concreta que implementa a interface Set;
- Os elementos são inseridos de modo a manter o conjunto ordenado;
- Um TreeSet funciona de modo parecido com um HashSet, exceto pelo esforço extra para manter a estrutura ordenada;
- Deve-se pagar o custo adicional apenas se a ordenação for essencial, caso contrário o uso de um HashSet será mais vantajoso.

© LES/PUC-Rio

A Classe TreeSet (2)



- A classe TreeSet possui alguns métodos adicionais, relacionados à manutenção dos elementos em ordem:

```
public class TreeSet<E> extends AbstractSet<E>
    implements SortedSet<E>, Cloneable, java.io.Serializable {
    public TreeSet();
    public TreeSet(Comparator c);
    public TreeSet(Collection c);
    public TreeSet(SortedSet<E> s);
    public SortedSet<E> subSet(E from, E to);
    public SortedSet<E> headSet(E to);
    public SortedSet<E> tailSet(E from);
    public Comparator<? super E> comparator();
    public E first();
    public E last();
}
```

© LES/PUC-Rio

A Interface Comparator



- A interface `Comparator` é usada para comparar dois objetos;
- Ela possui as seguintes operações:

```
public interface Comparator<E> {
    public int compare(E o1, E o2);
    public boolean equals(Object o);
}
```

- Uma implementação da operação `compare` deve retornar o seguinte:
 - Um valor negativo, caso `o1` seja “menor” que `o2`;
 - Um valor positivo, caso `o1` seja “maior” que `o2`;
 - Zero, se `o1` for “igual” a `o2`;

© LES/PUC-Rio

A Classe TreeSet - Exemplo (1)



```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Set<Ponto> ss=new TreeSet<Ponto>(new CompPonto());

        ss.add(new Ponto(5,0));
        ss.add(new Ponto(2,2));
        ss.add(new Ponto(4,6));
        ss.add(new Ponto(0,0));

        for(Ponto e: ss)
            System.out.println(e.getCoord());
    }
}
```

© LES/PUC-Rio

A Classe TreeSet - Exemplo (2)



- Devemos agora fornecer uma implementação para a interface Comparator:

```
import java.util.Comparator;

public class CompPonto implements Comparator<Ponto> {
    public int compare(Ponto p1,Ponto p2) {
        if((p1.x+p1.y)>(p2.x+p2.y))
            return 1;
        else
            if((p1.x+p1.y)<(p2.x+p2.y))
                return -1;
            else
                return 0;
    }
}
```

© LES/PUC-Rio

A Classe TreeSet - Exemplo (3)



- O programa anterior exibirá na console o seguinte:
- ```
0.0 0.0
2.0 2.0
5.0 0.0
4.0 6.0
```
- Isso se deve ao método usado na comparação de dois pontos.

© LES/PUC-Rio

Programa – Capítulo 15




Laboratório de Engenharia de Software

- Coleções
- A Interface `List`
- A Classe `LinkedList`
- Exercício
- A Interface `Set`
- A Classe `HashSet`
- A Classe `TreeSet`
- **A Interface `Map`**
- A Classe `HashMap`

© LES/PUC-Rio

A Interface `Map` (1)



Laboratório de Engenharia de Software

- Um `Map` é uma estrutura de dados que conecta **chaves** e **valores**;
- Cada **chave** pode ser associada a somente um **valor**;
- Exemplos de pares (**chave, valor**):
  - (numero telefone, cliente)
  - (matricula, aluno)
  - (cpf, contribuinte)
- O primeiro elemento do par é a **chave** para a recuperação do segundo;
- Um `Map` é como uma tabela de pares (**chave, valor**).

© LES/PUC-Rio

## A Interface Map (2)



- Declaração da interface: `public interface java.util.Map<K,V>`
- Principais operações:
  - `V put(K key, V value)` – associa o valor(value) à chave(key). Se o Map já possui um valor associado à chave, ele será substituído pelo novo valor(value);
  - `V get(Object key)` – retorna o valor associado à chave (key) ou o valor null, caso não haja nenhum valor associado à chave.

© LES/PUC-Rio


## A Interface Map (3)



- Principais operações (cont):
  - `Set<K> keySet()` – retorna as chaves armazenadas no Map como um Set.
  - `Collection<V> values()` – retorna os valores armazenados no Map como um Collection.
  - `Set<Map.Entry<K,V>> entrySet()` - retorna um Set de pares (chave, valor) contendo objetos representados pela interface interna Map.Entry.

© LES/PUC-Rio

| Programa – Capítulo 15                |                                                                                                                                                                                                                                                                                                                                                                                |  |
|---------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------|
| Laboratório de Engenharia de Software | <ul style="list-style-type: none"><li>• Coleções</li><li>• A Interface <code>List</code></li><li>• A Classe <code>LinkedList</code></li><li>• Exercício</li><li>• A Interface <code>Set</code></li><li>• A Classe <code>HashSet</code></li><li>• A Classe <code>TreeSet</code></li><li>• A Interface <code>Map</code></li><li>• <b>A Classe <code>HashMap</code></b></li></ul> |                                                                                     |
|                                       | © LES/PUC-Rio                                                                                                                                                                                                                                                                                                                                                                  |                                                                                     |

| A Classe <code>HashMap</code> – Exemplo (1) |                                                                                                                                                                                                          |  |
|---------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Laboratório de Engenharia de Software       | <pre>public class Aluno {<br/>    String nome;<br/><br/>    public Aluno(String n) {<br/>        nome=n;<br/>    }<br/><br/>    public String getNome() {<br/>        return nome;<br/>    }<br/>}</pre> |                                                                                       |
|                                             | © LES/PUC-Rio                                                                                                                                                                                            |                                                                                       |

## A Classe HashMap – Exemplo (2)



```
import java.util.*;

public class Ex {
 public static void main(String[] args) {
 Map<Integer,Aluno> lc=new HashMap<Integer,Aluno>();
 int matric;
 String nome;
 Scanner con=new Scanner(System.in);

 // Cadastramento dos alunos

 System.out.println("Forneca uma matricula");
 matric=con.nextInt();
 while(matric!=0) {
 con.nextLine();
 System.out.println("Forneca o nome");
 nome=con.nextLine();
 lc.put(matric,new Aluno(nome));
 System.out.println("Forneca uma matricula");
 matric=con.nextInt();
 }
 }
}
```

© LES/PUC-Rio

## A Classe HashMap – Exemplo (3)



```
// Exibição da lista de chamada
System.out.println("LISTA DE CHAMADA");
Set<Map.Entry<Integer,Aluno>> s;
s=lc.entrySet();

for(Iterator <Map.Entry<Integer,Aluno>> i=s.iterator();
 ;i.hasNext();) {

 String resp;
 Map.Entry<Integer,Aluno> me;
 me=(Map.Entry<Integer,Aluno>) i.next();
 Integer mat=me.getKey();
 Aluno a=me.getValue();
 resp=mat.toString()+" "+a.getNome();
 System.out.println(resp);
}
```

© LES/PUC-Rio



## A Classe HashMap – Exemplo (4)



Laboratório de Engenharia de Software

```
// Consulta à lista de chamada
System.out.println("CONSULTA A LISTA DE CHAMADA");
System.out.println("Forneca uma matricula");
matric=con.nextInt();
while(matric!=0) {
 Aluno a=lc.get(matric);
 if(a==null)
 System.out.println("Matricula inexistente");
 else
 System.out.println(a.getNome());
 System.out.println("Forneca uma matricula");
 matric=con.nextInt();
}
}
```

© LES/PUC-Rio

## Outros Elementos do Framework de Coleções (1)



Laboratório de Engenharia de Software

- **public class Stack<E>:**
  - representa uma pilha (LIFO - último a entrar, primeiro a sair) de objetos.
  - Estende a classe Vector com cinco operações que permitem que um vector possa ser tratado como uma pilha.
- **public class Vector<E>:**
  - Adaptada para implementar a interface List, tornando-se um membro de Java Collections Framework.
  - Implementa um array cujo tamanho pode aumentar ou diminuir para acomodar a adição e a remoção de itens dinamicamente.
  - Por ser um array, seus elementos podem ser acessados por meio de um índice inteiro.
  - A implementação de Vector é sincronizada. Dessa forma, se uma implementação thread-safe não for necessária, recomenda-se usar ArrayList.

© LES/PUC-Rio

## Outros Elementos do Framework de Coleções (2)



- **public interface Queue<E>:**
  - coleção desenhada para guardar elementos antes de serem processados.
  - Além das operações básicas de coleções, as filas fornecem operações adicionais de inserção, remoção e inspeção.
  - Cada um desses métodos existe em duas formas:
    - **Levanta uma exceção caso a operação falhe:** add(), remove() e element()
    - **Retorna um valor especial (null ou false, dependendo da operação):** offer(), poll() e peek()