

API e Coleções de objetos

Paradigmas de Linguagens de Programação

Heloisa

API (Application Programming Interface)

- Conjunto de implementações (métodos, objetos, classes, interfaces) pré-definidas em uma linguagem de programação que podem ser utilizadas em outros programas.
- Tem por objetivo facilitar a programação, uma vez que detalhes das implementações utilizadas não precisam ser especificados ou conhecidos.

Interface de Programação de Aplicativos (API)

- A API JAVA é uma coleção de classes, interfaces e exceções prontas, agrupadas em pacotes. Alguns pacotes da Api:

Pacote	Descrição
java.applet	Contém classes e interfaces que ativam as applets
java.awt	Permite escrever Interfaces Gráficas de Usuário
java.awt.geom	Fornecer classes 2D para definir e realizar operações em objetos relacionados a geometria bidimensional.
java.awt.image	Dedicado à criação e manipulação de imagens.
java.io	Trata a entrada e saída bruta do programa.
java.lang	Contém os elementos centrais da linguagem: Object, String, Exception, Thread.
java.net	Fornecer classes para desenvolver aplicações em rede
java.util	Contém classes de utilitários como as estruturas de dados.

- As subclasses de classes da API não estão necessariamente no mesmo pacote de seu pai.
- Cada classe, interface e exceção tem sua própria página Web que detalha métodos públicos e variáveis.
- Não é preciso importar pacotes – apenas facilita o código

- Importante:
- considerar os modificadores de métodos, ao usar classes da API
 - estáticos – devem ser chamados sem instanciar a classe
 - protected – não devem ser usados – não estamos escrevendo classes de um pacote API
 - final – classes finais não podem ser extendidas.
- conhecer as exceções que um método ou construtor podem lançar.

O pacote java.lang

(automaticamente importado)

- Classes empacotadoras (“wrappers”):
 - Boolean, Integer, Byte, Character, Integer, Long, Double, Float
- Classe Object: Classe raiz da hierarquia
- Classe String: representa cadeias de caracteres
- Classe System: contém os campos in e out, usados para entrada e saída.
- Classe Math: contém funções matemáticas básicas (logaritmo, exponencial, raiz quadrada,...)

A Classe `java.lang.Object`

- É a classe fundamental de JAVA. Fica no alto da hierarquia, contém vários métodos e pode ser instanciada.
- Todas as classes são subclasses de `Object`, direta ou indiretamente
- Qualquer objeto em Java é um `Object`, podendo ser referenciado como tal
- Podemos escrever procedimentos que recebem um `Object` como argumento ou retornam um `Object`:

`public void` `metodorecebeobject(Object object)`

`public Object` `metodoretornaobject(int i)`

A Classe `java.lang.Object`

- Os métodos básicos são sobrepostos nas subclasses, caso sejam usados.

Método	Descrição	Exceções Lançadas
<code>boolean equals(Object o)</code>	Retorna <code>true</code> se <code>o</code> é equivalente a este objeto.	Nenhuma.
<code>String toString ()</code>	Retorna uma <code>string</code> que representa as informações sobre este objeto.	Nenhuma.
<code>Object clone ()</code>	Cria e retorna uma cópia deste objeto.	<code>CloneNotSupportedException</code>

Método clone()

- Usado para criar uma cópia de um objeto existente:
`objetoclonavel.clone()`
- Para usar o método clone() a classe ou uma de suas superclasses devem implementar a interface Cloneable
- A implementação da classe Object verifica se o objeto em que clone foi chamado implementa a interface Cloneable.
- Se não, lança a exceção **CloneNotSupportedException**.
- Deve ser declarado como:
`protected Object clone() throws CloneNotSupportedException`
- ou:
`public Object clone() throws CloneNotSupportedException`

Exemplo - clone

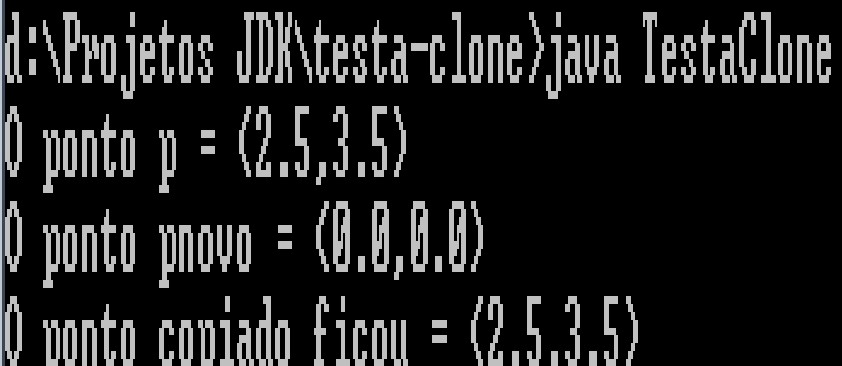
```
public class Ponto2D implements
    Cloneable
{
    private double x, y;

    Ponto2D(double x_ini, double
y_ini)
    {
        x = x_ini;
        y = y_ini;
    }
    ...
// getters e setters
    ...
```

```
public Object clone() {
    try { return super.clone();
    }
    catch (CloneNotSupportedException e) {
        System.out.println("Deu exceção");
        return this;}
    }

    public String toString()    {
        String resultado = "(" + x + "," + y + ")";
        return resultado;
    }
}
```

```
public class TestaClone
{
    public static void main (String[] args)
    {
        Ponto2D p = new Ponto2D(2.5,3.5);
        Object o = p;
        Ponto2D pnovo = new Ponto2D(0.0,0.0);
        System.out.println("O ponto p = " + p);
        System.out.println("O ponto pnovo = " + pnovo);
        pnovo = (Ponto2D) p.clone();
        System.out.println("O ponto copiado ficou = " + pnovo);
    }
}
```



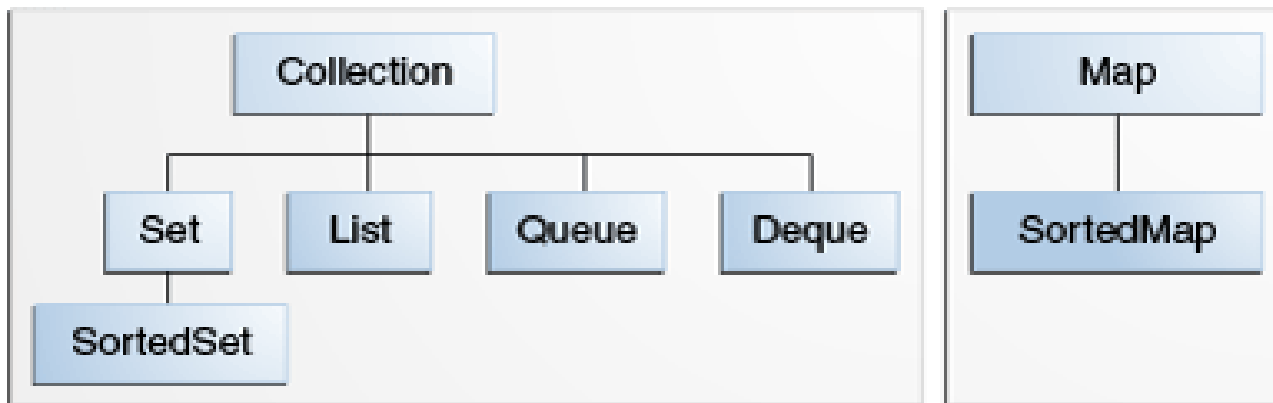
```
d:\Projetos JDK\testa-clone>java TestaClone
O ponto p = (2.5,3.5)
O ponto pnovo = (0.0,0.0)
O ponto copiado ficou = (2.5,3.5)
```

Coleções de objetos

- Coleções (também chamadas containers) são objetos que contém conjuntos de objetos;
- Java contém classes e interfaces que oferecem mecanismos para agrupar e processar objetos em conjunto;
- Muitas dessas classes implementam estruturas de dados complexas, mas que são transparentes para o programador.
- Dados primitivos devem ser empacotados em objetos para serem inseridos em uma coleção.

Framework de coleções em Java

- Interfaces de coleções básicas
- Encapsulam diferentes tipos de coleções (pacote java.util)



Interfaces básicas de coleções

- Set – coleção que não pode conter elementos duplicados;
- List – coleção ordenada de elementos, que podem ser duplicados. Elementos podem ser acessados por índices;
- Queue – coleção de elementos armazenados tipicamente como fila;
- Deque - coleção de elementos armazenados tipicamente como fila ou pilha;
- Map – coleção de objetos que associam chaves a valores. Não pode ter chaves duplicadas e cada chave está associada a apenas um valor (HashTable)

Interface List

- Implementações de uso geral da interface List:
- ArrayList – coleção ordenada de elementos que pode variar de tamanho em tempo de execução;
- LinkedList – coleção de elementos na forma de lista duplamente encadeada. Implementa interfaces List e Deque;
- Vector – coleção de elementos de versões anteriores, retroadaptada para ser implementação da interface List.

As Classes ArrayList e Vector

- São utilizadas como arrays que tem tamanho variável;
- Operações possíveis: adicionar elementos a um vetor, remover elementos, verificar se um elemento está no vetor e processar elementos do vetor;
- É um array de objetos que pode crescer dinamicamente, de maneira a ter espaço para seus elementos;
- Elementos são acessados por suas posições no vetor;
- Elementos não podem ser tipos primitivos. (int deve ser empacotado como Int)

Diferenças entre ArrayList e Vector

- **Sincronização:**
 - **ArrayList não é synchronized**, logo, não deve ser compartilhada entre múltiplas threads;
 - **Vector é synchronized**, logo, é a melhor opção para ser compartilhada entre múltiplas threads;
- **Eficiência:** ArrayList é mais rápido que Vector. Como Vector é synchronized e thread-safe, fica um pouco mais lento.
- **Aumento da Capacidade:** Ao inserir um elemento em uma coleção com espaço esgotado, a capacidade é expandida automaticamente
 - Vector: dobra a capacidade
 - ArrayList: aumenta em 50%

Exemplo usando Vector

```
import java.util.*;
public class VectorDemo {
    Vector names = new Vector(3, 2);
    public static void main (String argv [ ]){
        VectorDemo v = new VectorDemo();
        v.exemplo();
    }
    public void exemplo () {
        names.addElement("Joao");
        names.addElement("Paula");
        names.addElement("Fabio");
        System.out.println("A capacidade e " +
names.capacity());
        System.out.println("O tamanho e " + names.size());
        names.insertElementAt("Maria", 2);
        System.out.println("A capacidade e agora " +
names.capacity());
        System.out.println("O tamanho e agora " +
names.size());
        System.out.println("O último elemento e " +
names.lastElement());
    }
}
```

```
d:\Projetos JDK\vector-teste>javac VectorDemo.java
Note: VectorDemo.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
```

```
d:\Projetos JDK\vector-teste>java VectorDemo
A capacidade e 3
O tamanho e 3
A capacidade e agora 5
O tamanho e agora 4
O último elemento e Fabio
```

Genéricos

- Genéricos são construções de JAVA que oferecem uma forma abstrata de representar tipos e métodos
- A forma mais comum de utilizá-los é com as classes containers
- A declaração da classe tem um parâmetro de tipo:

```
List<Integer>
```

```
    myIntList = new LinkedList<Integer>();
```

```
// objeto da classe LinkedList que só aceita inteiros
```

```
ArrayList<String> myArray = new ArrayList<String> ();
```

```
// objeto da classe ArrayList que só aceita Strings
```

Genéricos - exemplo

A classe genérica Box utiliza um objeto

```
public class Box {  
    private Object object;  
    public void set(Object object) { this.object = object; }  
    public Object get() { return object; }  
}
```

Os métodos aceitam e retornam Objects

```
/** * Versão genérica da classe Box . <T> é o tipo do valor sendo utilizado */
```

```
public class Box<T> {  
    // T significa "Type"  
    private T t;  
    public void set(T t) { this.t = t; }  
    public T get() { return t; }  
}
```

```
Box<Integer> integerBox;
```

```
Box<Integer> integerBox = new Box<Integer>();
```

ArrayList e genéricos

- ArrayList pode ser usado com ou sem genéricos;
- É recomendável usar genéricos, por permitirem verificação de tipos em tempo de compilação;

```
ArrayList<String> stringList = new ArrayList<String>();  
//Generic ArrayList to Store only String objects
```

- Com essa instanciação de stringList, ocorre erro de compilação ao tentar adicionar um elemento que não seja do tipo String;

```
stringList.add("Item");  
//no error because we are storing String
```

```
stringList.add(new Integer(2));  
//compilation error
```

Construtores de ArrayList

Construtores de ArrayList	
ArrayList()	cria um vetor vazio com tamanho default (10 no JDK)
ArrayList(Collection <? Extends E> c)	cria um vetor com os elementos especificados na coleção
ArrayList (int initialCapacity)	cria um vetor vazio com tamanho initialCapacity

Construtores de Vector

Construtores de Vector	
Vector ()	cria um vetor vazio com tamanho default (10 no JDK)
Vector(Collection <? Extends E> c)	cria um vetor com os elementos especificados na coleção
Vector(int initialCapacity)	cria um vetor vazio com tamanho initialCapacity
Vector(int initialCapacity, int capacityIncrement)	cria um vetor vazio com tamanho initialCapacity e incremento de capacityIncrement

Métodos de ArrayList e Vector

<code>boolean add (E element)</code>	adiciona o elemento no fim do vetor
<code>void add(int index, E element)</code>	Adiciona elemento na posição especificada
<code>public Object clone ()</code>	Retorna uma duplicação do vetor.
<code>boolean contains (Object element)</code>	retorna true se o elemento está no vetor
<code>E get (int index)</code>	retorna o elemento da posição especificada.
<code>boolean isEmpty ()</code>	retorna <i>true</i> se o vetor está vazio
<code>public Object [] toArray()</code>	Retorna um array com os elementos do vetor mantendo a ordem

Métodos de ArrayList e Vector

int indexOf (Object element)	retorna a posição do primeiro elemento no vetor que <i>casa</i> com o parâmetro ou -1 se o elemento não está no vetor
int lastIndexOf(Object element)	retorna a posição da última ocorrência do elemento no vetor ou -1 se o vetor não contém o elemento.
boolean remove (Object o)	Remove a primeira ocorrência do elemento, se ele existir.
E remove (int index)	remove e retorna o elemento da posição especificada por index. Desloca os elementos depois desse para a esquerda. Throws: ArrayIndexOutOfBoundsException - se index está fora do intervalo $!(index < 0 \mid \mid index \geq size())$

Métodos de ArrayList e Vector

<code>E void set (int index, E element)</code>	substitui o elemento da posição index pelo novo elemento. Throws: ArrayIndexOutOfBoundsException - se index está fora do intervalo <code>!(index < 0 index >= size())</code>
<code>void ensureCapacity(int minimumCapacity)</code>	aumenta a quantidade de espaço disponível no vetor para aquele especificado por <code>minimumCapacity</code> . Se a capacidade já é maior, nada muda.
<code>int size()</code>	retorna o número de elementos do vetor.
<code>void trimToSize()</code>	reduz a capacidade do vetor para o tamanho corrente. Libera espaço de memória não utilizado, previamente obtido pelo vetor.

Métodos de Vector

<code>void addElement(E obj)</code>	Adiciona elemento no final do vetor, aumentando sua capacidade em uma posição
<code>Object firstElement ()</code>	retorna o primeiro elemento do vetor sem remover. Throws: NoSuchElementException se o vetor não tem elementos
<code>void removeElementAt (int index)</code>	remove o elemento da posição especificada por index. Throws: ArrayIndexOutOfBoundsException - se index está fora do intervalo $!(index < 0 \ \ index \geq size())$
<code>boolean removeElement(Object element)</code>	remove o elemento especificado do vetor. Retorna <i>true</i> se sucesso e <i>false</i> se o elemento não estava no vetor.
<code>void setElementAt (Object object, int index)</code>	substitui o objeto da posição index pelo novo objeto. Throws: ArrayIndexOutOfBoundsException - se index está fora do intervalo $!(index < 0 \ \ index \geq size())$

Exemplo usando ArrayList com genéricos

```
import java.util.ArrayList;

public class ArrayListDemo {
    public static void main(String[] args) {
        // Criar um array vazio com capacidade inicial 5
        ArrayList<Integer> lista = new ArrayList<Integer>(5);
        // inserir elementos na lista com add()
        lista.add(15);
        lista.add(22);
        lista.add(30);
        lista.add(40);
        System.out.println("O tamanho e " + lista.size());
        // inserir elemento 25 na terceira posição
        lista.add(2,25);
    } }
```

Exemplo usando ArrayList

```
System.out.println("O elemento da posicao 2 é" +  
lista.get(2));  
System.out.println("O tamanho agora é " + lista.size());
```

```
// imprimindo todos os elementos
```

```
for (Integer number : lista)  
{  
    System.out.println("Numero = " + number);  
}
```

```
}
```

Numero = 15

Numero = 22

Numero = 25

Numero = 30

Numero = 40

A Classe LinkedList

- Implementação de lista duplamente encadeada das interfaces List e Deque.
- Construtor: LinkedList() - cria uma lista vazia.

void addFirst(E e)	Insere o elemento especificado no início da lista.
void addLast(E e)	Adiciona o elemento especificado ao final da lista
public E element()	Retorna mas não remove o primeiro elemento da lista
E pop ()	retorna e remove o elemento do topo. NoSuchElementException – se a lista estiver vazia
E push (E element)	insere o elemento no topo da pilha. Retorna o elemento inserido.


```
import java.util.*;

public class LinkedListDemo {

    public static void main(String args[]) {
        // create a linked list
        LinkedList list = new LinkedList();
        // add elements to the linked list
        list.add("F");
        list.add("B");
        list.add("D");
        list.add("E");
        list.add("C");
        list.addLast("Z");
        list.addFirst("A");
        list.add(1, "A2");
        System.out.println("Original contents of list: " + list);

        // remove elements from the linked list
        list.remove("F");
        list.remove(2);
        System.out.println("Contents of list after deletion: " + list);
    }
}
```

// remove first and last elements

```
list.removeFirst();  
list.removeLast();  
System.out.println("list after deleting first and last: " + list);
```

// get and set a value

```
Object val = list.get(2);  
list.set(2, (String) val + " Changed");  
System.out.println("list after change: " + list);  
}  
}
```

Original contents of list: [A, A2, F, B, D, E, C, Z]

Contents of list after deletion: [A, A2, D, E, C, Z]

list after deleting first and last: [A2, D, E, C]

list after change: [A2, D, E Changed, C]

A Classe Stack

- Subclasse de Vector, permite implementar um objeto com comportamento de pilha: last-in first-out.
- Construtor: Stack - cria uma pilha vazia.

boolean empty ()	retorna <i>true</i> se a pilha não tem elementos.
E peek ()	retorna o elemento do topo, sem removê-lo. Throws: EmptyStackException Se a pilha está vazia.
E pop ()	retorna e remove o elemento do topo. Throws: EmptyStackException Se a pilha está vazia.
E push (E element)	insere o elemento no topo da pilha. Retorna o elemento inserido.
int search (Object element)	retorna a posição do elemento na pilha ou -1 se o objeto não está na pilha (o objeto do topo está na posição 1)

```

import java.util.Stack;
public class PilhaDemo      {
    Integer elem;
    Stack pilha = new Stack ();
    public static void main (String argv [ ]) {
        PilhaDemo p = new PilhaDemo ( );
        p.exemplo ();
    }
    public void exemplo ( ) {
        for( int i = 0; i < 5 ; i++) {
            elem = new Integer (i);
            pilha.push(elem);
            System.out.println("Empilha " + i ); }
        System.out.println("O elemento do topo da pilha
eh" + pilha.peek( ) ); } }

```

```
import java.util.Stack;
public class PilhaDemo      {
    Integer elem;
    Stack<Integer> pilha = new Stack<> ();
    public static void main (String argv [ ]) {
        PilhaDemo p = new PilhaDemo ( );
        p.exemplo ();
    }
    public void exemplo ( ) {
        for( int i = 0; i < 5 ; i++) {
            // elem = new Integer (i); não precisa empacotar
            pilha.push(i);
            System.out.println("Empilha " + i ); }
        System.out.println("O elemento do topo da pilha
eh" + pilha.peek( ) ); } }
```

A partir da versão 7



ArrayDeque

- Forma alternativa para trabalhar com estrutura de pilha:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

- Não tem limite de capacidade, cresce conforme a necessidade
- Não são “thread-safe”
- Tende a ser mais rápida que Stack quando usada como pilha e mais rápida que LinkedList quando usada como fila.

Classes empacotadoras (wrappers)

- Valores nativos (primitivos) não podem ser inseridos diretamente em uma coleção de elementos (objetos das classes containers), devem primeiro ser encapsulados na respectiva classe empacotadora:
 - boolean – Boolean
 - byte – Byte
 - int – Integer
 - double – Double
 - ...
- Para recuperar elementos de coleções como tipos primitivos:
 - booleanValue()
 - byteValue()
 - intValue()
 - doubleValue()
 -

```
Vector v = new Vector();  
Integer unNumero = new Integer(10);  
Integer outroNumero = new Integer((int) (100*Math.random()));  
....  
v.add(unNumero);  
v.add(outroNumero);  
...  
Integer N1 = v.elementAt(i);  
int n1 = N1.intValue();
```


Ferramentas de Manipulação de Strings

- Classe String - pertence ao pacote java.lang

Métodos:

- `int S.length()` – retorna o comprimento da String S.
- `boolean S1.equals(String S2)` - sobrepõe o método `equals` da classe `Object` e retorna *true* quando as strings são iguais e retorna falso caso contrário.

- `int S1.compareTo (String S2)`
 - retorna -1 se S1 é lexicograficamente menor que S2
 - retorna 0 se S1 = S2
 - retorna 1 se S1 é lexicograficamente maior que S2
- Exemplo:

```
if (S1.compareTo(S2) != 0)  
    System.out.println ("S1 nao e equivalente a String S2");
```

```
if (S1.equals(S2))  
    System.out.println("S1 e equivalente a String S2");
```

Métodos que retornam valores booleanos

Método	Retorna True Quando ...(e false caso contrário)
<code>S1.equalsIgnoreCase(String S2)</code>	A string S1 é igual a S2, independente de maiúsculas ou minúsculas.
<code>S1.startsWith(String S)</code>	A string S1 começa com S.
<code>S1.startsWith(String S,int compr)</code>	A string S1 começa com os primeiros compr caracteres de S.
<code>S1.endsWith(String S)</code>	A string S1 termina com S.
<code>S1.isEmpty()</code>	A string S1 é vazia.
<code>S1.contains(String S)</code>	A string S1 contém a string S

Métodos que retornam valores booleanos

String S = “paradigma”;

String S1 = “ParaDigma”;

String S2 = “para”;

String S3 = “Dig”;

...

S.equalsIgnoreCase(S1) **retorna true**

S.isEmpty() **retorna false**

S.startsWith(S2) **retorna true**

S1.contains(S3) **retorna true**

Substrings

- `char charAt(int index)` – retorna o caracter na posição especificada por `index`. Lança a exceção [IndexOutOfBoundsException](#) se o índice é negativo ou não é menor que o tamanho da string.
- `String S.substring (int inicio)` - retorna uma nova String que começa na posição `inicio`.
- (exceção: [IndexOutOfBoundsException](#))
- Exemplos:
- `"unhappy".substring(2)` returns `"happy"`
- `"Harbison".substring(3)` returns `"bison"`
- `"emptiness".substring(9)` returns `""` (an empty string)

Substrings

- `String S.substring (int inicio, int fim)` - retorna uma nova string que começa na posição inicio e termina na posição fim-1.
- (exceção: [IndexOutOfBoundsException](#))
- Exemplos:
 - `"hamburger".substring(4, 8)` returns "urge"
 - `"smiles".substring(1, 5)` returns "mile"

Substrings

```
String S = "0123456789";  
String S1 = S.substring (0);  
String S2 = S;  
//As strings S, S1 e S2 são todas equivalentes
```

```
String S4=S.substring(4);  
//S4 tem valor "456789"
```

```
String S5 = S.substring(4,7);  
//S5 tem o valor "456"
```

- Os métodos `indexOf` permitem examinar os caracteres individuais de uma `String` e retornam o índice desse character.
- `int S.indexOf(char a)`
Retorna o índice da primeira ocorrência do character `a` na cadeia `S` ou `-1` se o character não aparece na cadeia.
- `int S1.indexOf(char a, int inicio)`
Retorna o índice da primeira ocorrência do character `a` na cadeia `S` a partir da posição `inicio` ou `-1` se o character não aparece na cadeia.

- `int S1.indexOf(String S)`
 - Retorna o índice do início da primeira ocorrência da cadeia S na cadeia S1 ou -1 se a cadeia S não aparece em S1.
- `int S1.indexOf(String S, int inicio)`
 - Retorna o índice do início da primeira ocorrência da cadeia S na cadeia S1 a partir da posição início ou -1 se a cadeia S não parece, em S1.
- Método `lastIndexOf()`
 - cada método `indexOf` tem um `lastIndexOf` correspondente, que começa sua busca no final da String.

Alterando String

- As alterações são feitas em uma cópia, que é retornada. Nenhum dos métodos da classe String altera a String em si. É preciso atribuir a cópia alterada a mesma variável ou a uma variável diferente.

Método para String S1	retorna
S1.concat(String S2)	S1 + S2
S1.replace(char a,char b)	S1 com ocorrências de a substituídas por b
S1.toLowerCase()	S1 sem letras maiúsculas
S1.toUpperCase()	S1 sem letras minúsculas
S1.trim()	S1 sem brancos antes ou depois

Substrings

String S = "0123456789";

String S1 = "abcdec";

String S2 = "Para";

String S3 = "digma";

String S4 = S2.concat(S3); //S4 = "Paradigma"

String S5 = S1.replace('c','z'); // S5 = "abzdez"

S4 = S4.toLowerCase(); //S4 = "paradigma"