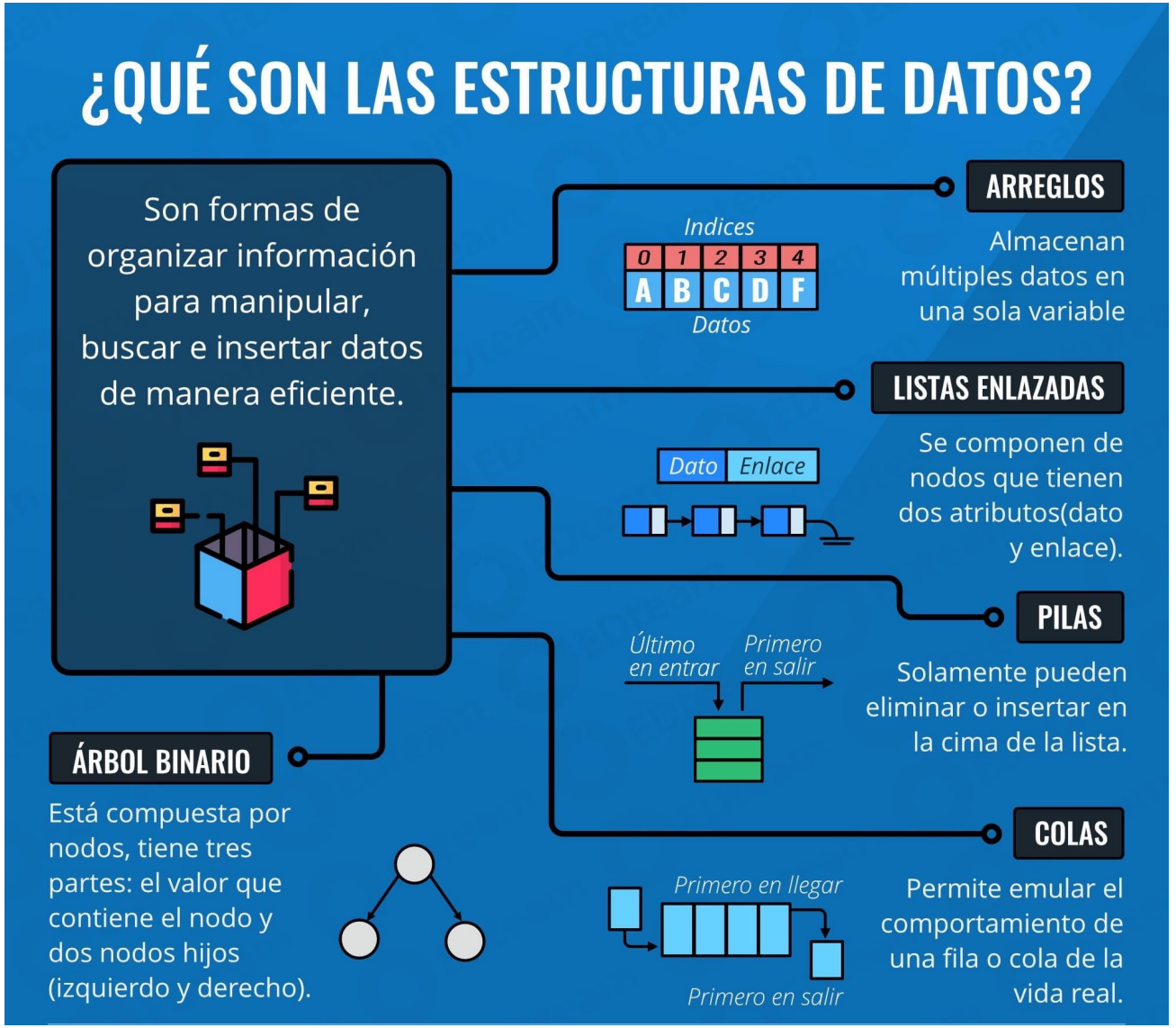
COLECCIONES

Colecciones

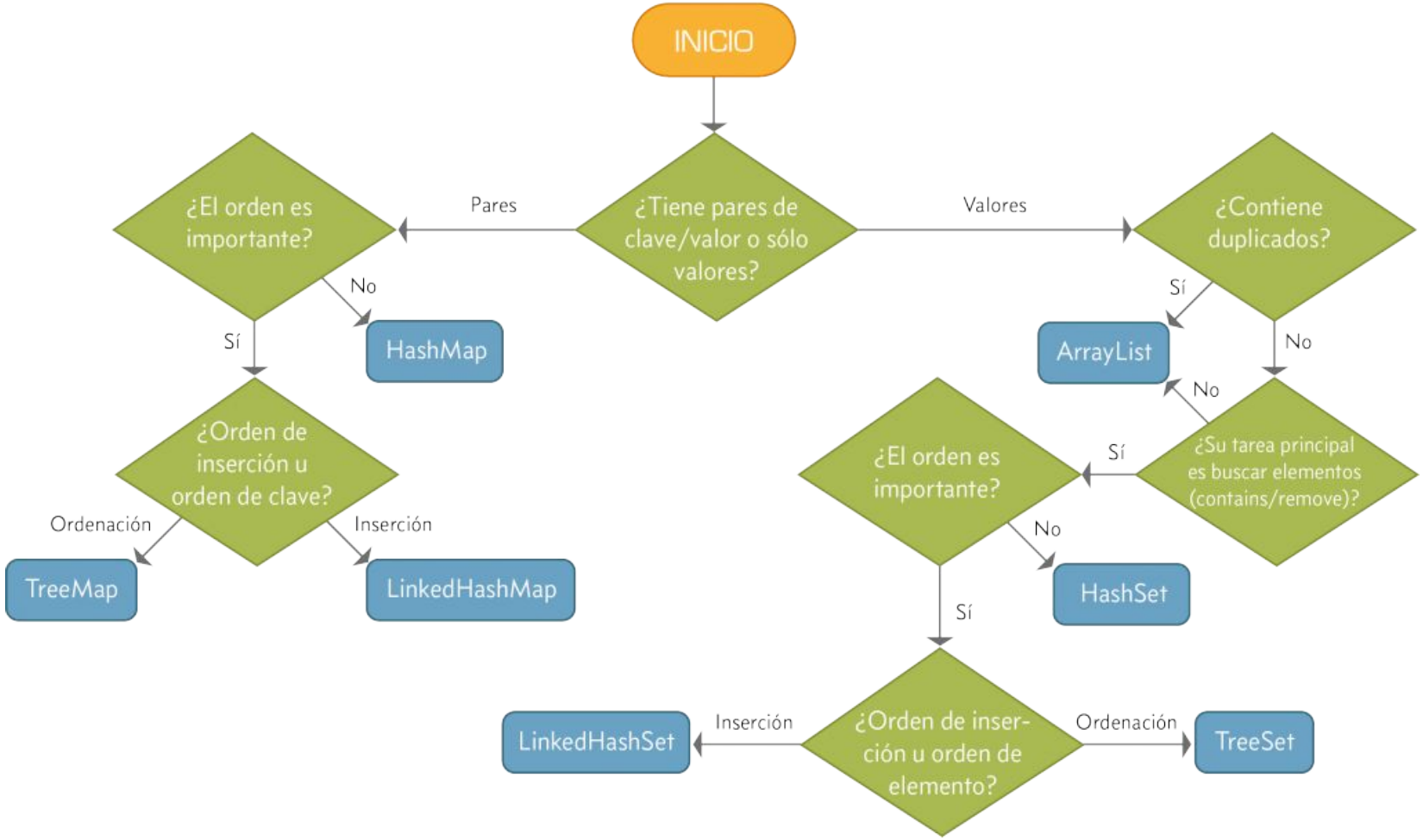


Colecciones

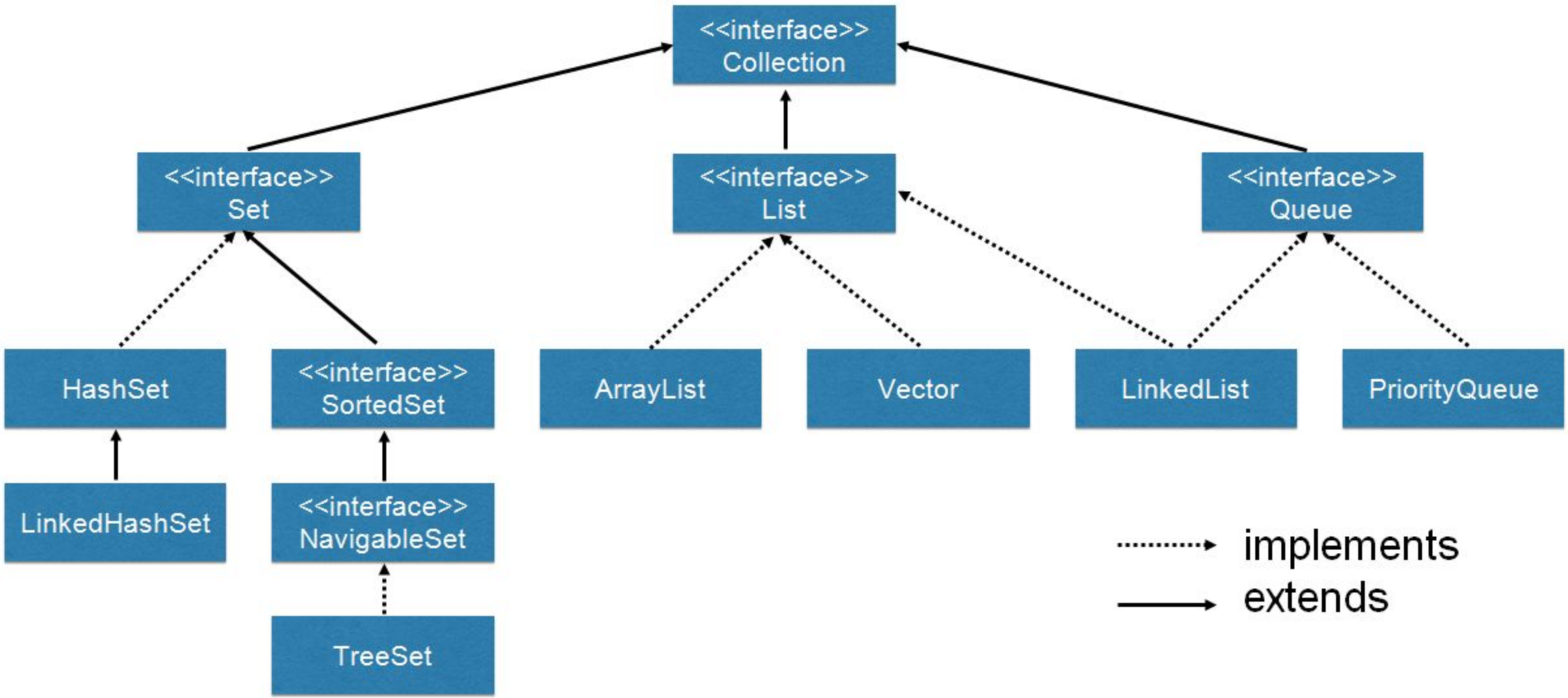
- Concepto:
    - Representa un grupo de objetos (elementos).
    - Es el almacén lógico donde guardar los elementos.
    - En Java se emplea la interfaz genérica Collection.
    - Tipos (interfaces):
        - Set (HashSet, TreeSet, LinkedHashSet).
        - List (ArrayList, LinkedList)
        - Map (HashMap, TreeMap, LinkedHashMap)

Colecciones

- Diagrama de decisión para uso de Colecciones Java:



Collection Interface



Colecciones

- Set (interface):
    - Define una colección que no puede contener elementos duplicados.
    - Implementaciones:
        - **HashSet**: almacena los elementos en una tabla hash. No importa el orden que ocupen los elementos.
        - **TreeSet**: almacena los elementos ordenándolos en función de sus valores. Los elementos almacenados deben implementar la interfaz **Comparable**.
        - **LinkedHashSet**: almacena los elementos en función del orden de inserción.

Colecciones

- List (interface):
    - Define una sucesión de elementos. Admite duplicados.
    - Implementaciones:
        - **ArrayList**: se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de situaciones.
        - **LinkedList**: se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.
        - **Stack**: LIFO

Colecciones

- ArrayList

```
ArrayList<Integer> vector = new ArrayList<Integer>();
System.out.println("Esta vacio?: " + vector.isEmpty());
vector.add(2);
vector.add(5);
vector.add(3);
System.out.println("toString: " + vector);
vector.remove(2);
System.out.println("toString: " + vector);
System.out.println("Esta vacio?: " + vector.isEmpty());
System.out.println("Posición del elemento 5: " +
  vector.indexOf(5));
System.out.println("Tamaño del vector: " + vector.size());
```

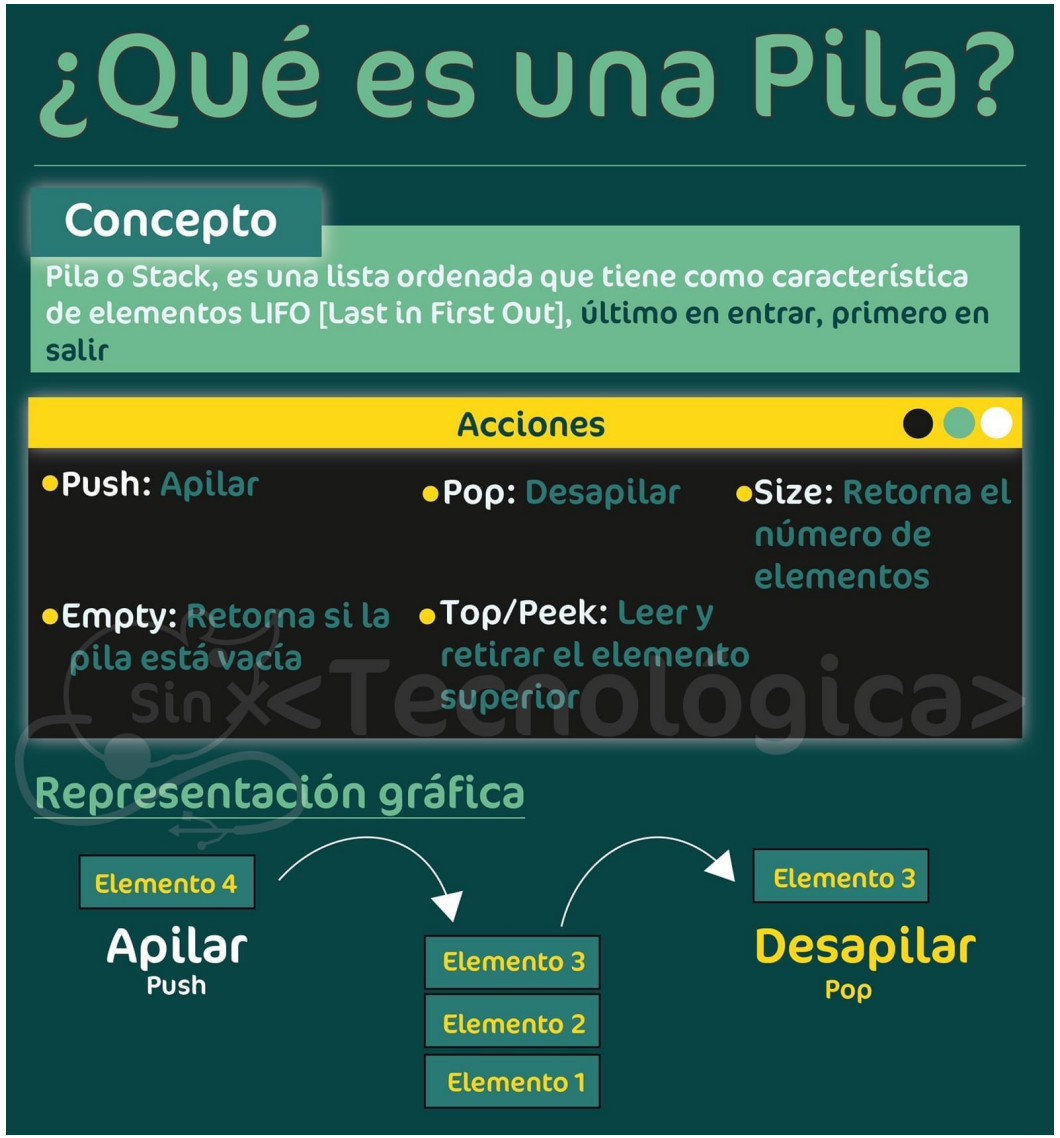
Colecciones

Colecciones

- LinkedList:

```
List<Integer> lista = new LinkedList<Integer>();
System.out.println("Esta vacia?: " + lista.isEmpty());
lista.add(2);
lista.add(1, 5);
lista.add(3);
System.out.println("toString: " + lista);
lista.remove(1);
System.out.println("toString: " + lista);
System.out.println("Esta vacia?: " + lista.isEmpty());
System.out.println("Elemento en pos 1?: " +
  lista.get(1));
System.out.println("Tamaño de la lista: " +
  lista.size());
```

Colecciones



Colecciones

- Stack:

```
Stack<Integer> pila = new Stack<Integer>();
System.out.println("Esta vacia?: " + pila.empty());
pila.push(2);
pila.push(5);
pila.push(3);
System.out.println("toString: " + pila);
pila.pop();
System.out.println("toString: " + pila);
System.out.println("Esta vacio?: " + pila.empty());
System.out.println("Elemento en el tope: " + pila.peek());
```

Colecciones

Pilas (Stacks)

uso de **import** java.util.Stack;

```
import java.util.Stack;

public class StackExample {
  public static void main(String args[]) {
    Stack s = new Stack();
    s.push("Java");
    s.push("Source");
    s.push("and");

    System.out.println("Next: " + s.peek());
    s.push("Support");
    System.out.println(s.pop());
    s.push(".");
    int count = s.search("Java");
    while (count != -1 && count > 1) {
      s.pop();
      count--;
    }
    System.out.println(s.pop());
    System.out.println(s.empty());
  }
}
```

Colecciones

- Map (interface):
  - Asocia claves a valores. No puede contener claves duplicadas y; cada clave, sólo puede tener asociado un valor.
  - Implementaciones:
    - **HashMap**: almacena las claves en una tabla hash. Es la implementación con mejor rendimiento de todas pero no garantiza ningún orden a la hora de realizar iteraciones.
    - **TreeMap**: almacena las claves ordenándolas en función de sus valores. Las claves almacenadas deben implementar la interfaz **Comparable**.
    - **LinkedHashMap**: almacena las claves en función del orden de inserción.
    - **Properties**: útil para almacenar y recuperar archivos de propiedades (opciones de configuración para programas)

Colecciones

- Properties:

```
Properties prop = new Properties();
prop.put(“user”, “ppando”);
prop.get(“user”);
prop.load(new FileInputStream(new File(“/prop.properties")));
```

- HashMap:

```
HashMap<String, Object> map = new HashMap<String, Object>();
map.put("user", "ppando");
map.get("user");
```

Preguntas de entrevista laboral (desarrollador)

ARREGLOS Y STRINGS

- Eliminar elementos duplicados sin usar un buffer.
- Invertir un arreglo.
- Determinar si todos los elementos son únicos.
- Dados 2 strings, decir si son anagramas.
- Reemplazar todas las ocurrencias de un carácter.

MATRICES

- Si un elemento es 0, poner en 0 toda su fila y toda su columna.
- Sumar elementos de sub-matriz.
- Contar elementos negativos de una matriz ordenada, en tiempo lineal.
- Rotar una matriz en 180 grados.

LISTAS ENLAZADAS

- Hallar el nodo n, contando desde el final de la lista.
- En una lista enlazada simple, eliminar el elemento del medio, suponiendo que sólo se tiene acceso a ese elemento y no al nodo inicial de la lista.
- Dados dos números representados por listas donde cada nodo contiene un dígito y los números están almacenados en orden inverso, sumarlos y retornarlos como lista.
- Dada una lista circular, retornar el nodo inicial del bucle.
- Decir si una lista cuyos elementos son caracteres es un palíndromo (recursividad).

PILAS Y COLAS

- Implementar 3 pilas usando un único arreglo.
- Implementar una cola que permita ejecutar *pop(índice)* en una sub-cola.
- Ordenar los elementos de una pila en sentido ascendente.
- Crear una cola usando 2 pilas.
- Implementar una pila, incluyendo una función que retorne el mínimo.

ÁRBOLES Y GRAFOS

- Dado un grafo dirigido, decir si hay algún camino entre dos nodos determinados.
- Verificar si un árbol está balanceado.
- Dado un árbol donde cada nodo contiene un número, imprimir todos los caminos posibles que, en suma, arrojen determinado resultado.
- Crear un árbol de altura mínima a partir de un arreglo.
- Imprimir caminos Hamiltonianos de un grafo.