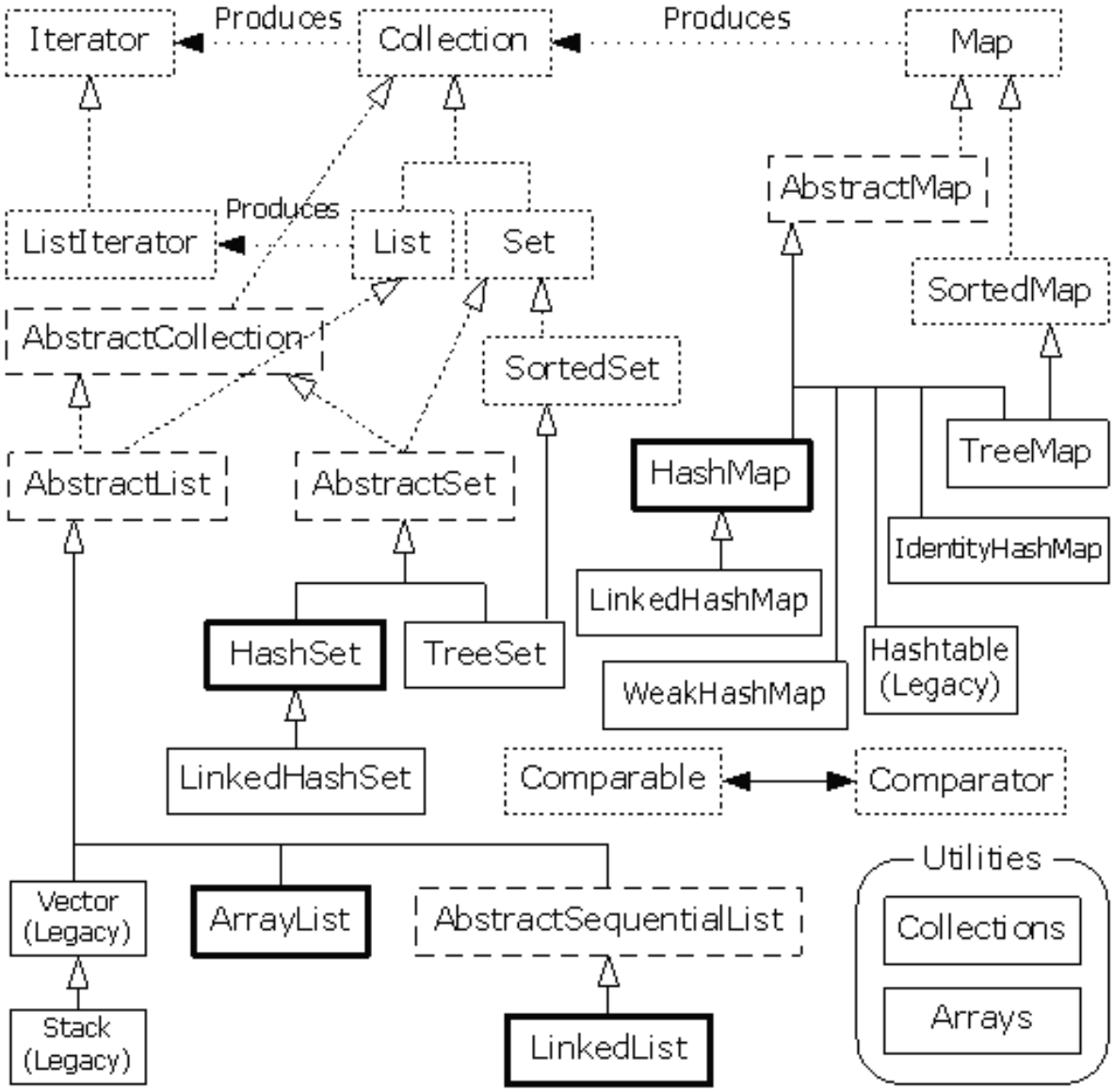
Colecciones

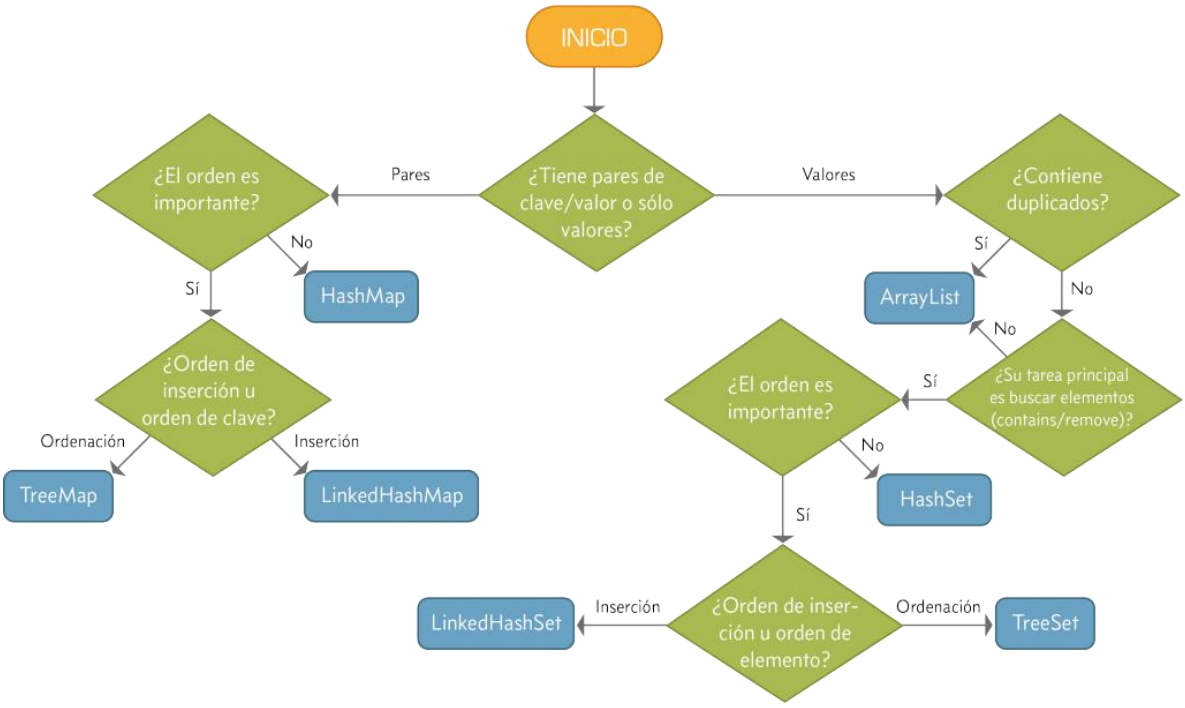
Las colecciones permiten crear conjuntos de elementos. Es una generalización del concepto de vector.

- 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)

Tipos



Como decidir que tipo usar



- 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.

- 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

Ejemplo 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());
```

Elemplo 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());
```

Ejemplo 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());
```

Map

- 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)

Properetiés

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

Hash Map

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