# MAPAS

1. ¿En qué consiste?
2. Métodos
3. Clases que implementan la interfaz Mapa
    a. En qué consisten.
    b. Ejercicio sobre la clase.
    c. Métodos para trabajar con ellas.
    d. Ejemplo de código.
4. Ejercicio complejo

## ¿En qué consiste?

La interfaz Map **no es un subtipo de la interfaz Collection**, ya que los mapas son estructuras de datos que permiten almacenar pares de **clave-valor**. Es decir, cada elemento en un mapa está compuesto por una **clave única y un valor asociado** a esa clave. Los mapas son útiles cuando necesitas acceder a valores de manera eficiente utilizando una clave específica.

Principales características de los mapas:

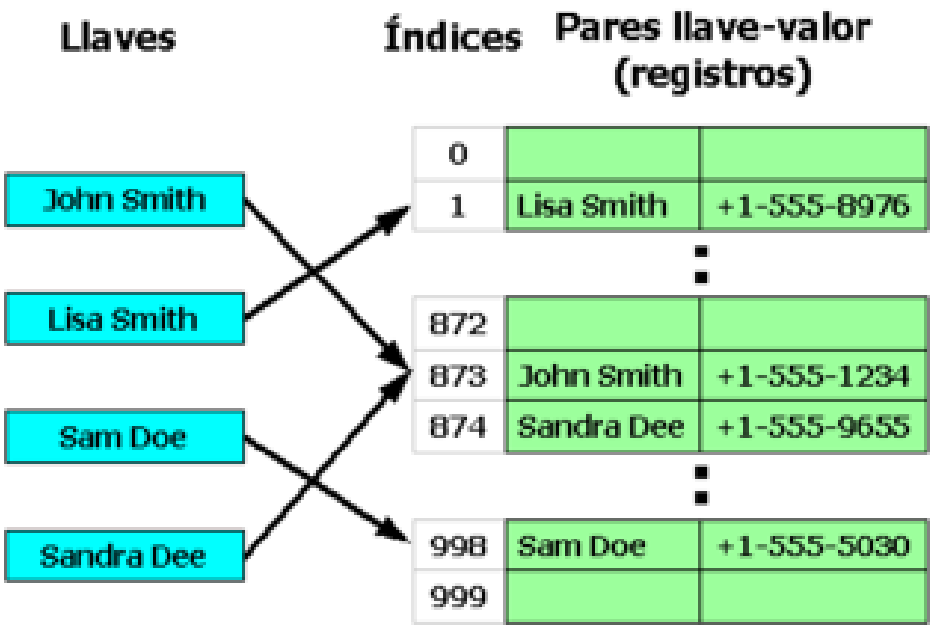
- ➔**Claves únicas:** si intentas agregar un nuevo par con una clave que ya existe, el valor anterior será reemplazado.
- ➔**Valores no únicos:** diferentes claves pueden tener el mismo valor asociado.
- ➔**No ordenados:** por defecto, los mapas no garantizan un orden específico de los elementos (aunque existen implementaciones como TreeMap que sí lo hacen).

# Métodos

| | |
|---|---|
| `Object put(Object key, Object value)` | Añade un valor asociado a una clave. |
| `Object get(Object key)` | Devuelve el valor asociado a una clave |
| `Object remove(Object key)` | Borra el valor asociado a una clave y lo devuelve |
| `boolean containsKey(Object key)` | Indica si hay una entrada asociada a la clave facilitada. |
| `boolean containsValue(Object value)` | Indica si hay claves asociadas al valor facilitado. |
| `int size()` | Indica el número de parejas. |
| `boolean isEmpty()` | Indica si está vacío el mapa. |
| `void clear()` | Borra todos los elementos. |
| `public Set keySet()`<br>`public Collection values();`<br>`public Set entrySet();` | Métodos para obtener claves, valores y parejas como un conjunto. |

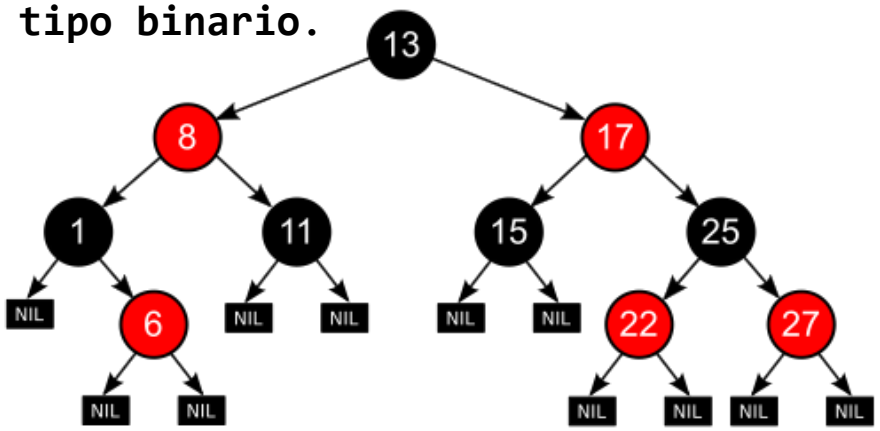
# Clases que implementan la interfaz Mapa

1. HashMap
   a. Almacena pares **clave-valor en una tabla hash.**




   b. No garantiza **ningún orden específico** de los elementos.
   c. **Permite claves y valores null.**
   d. Ofrece un **rendimiento constante.**
   e. **Uso común:** Cuando no necesitas un orden específico y buscas un alto rendimiento.
   f. **Ejercicio**:
      i. Crear un HashMap y añadir mínimo: 3 valores normales, 1 con clave null, 1 con valor null.
      ii. ¿Qué pasó con el orden?
      iii. Añadir otra pareja con clave null, ¿Qué sucede?
      iv. Eliminar una fila de la tabla.
      v. Imprimir las claves y valores en dos filas (una para las claves y otra para los valores).

2. TreeMap
   a. Almacena pares clave-valor en un **árbol rojo-negro de tipo binario.**




   b. Mantiene los **elementos ordenados** por la clave de la pareja.
   c. **No permite claves null** (excepto si se proporciona un comparador que las maneje).
   d. Ofrece un **rendimiento logarítmico**.
   e. **Uso común:** Cuando necesitas que los elementos estén ordenados.
   f. ***Ejercicio:**
      i. Copiar los valores del ejercicio anterior e indicar los correctos.
      ii. Imprime en orden inverso.
      iii. Imprime solo el primer valor.
      iv. Elimina e imprime el último valor.

3. LinkedHashMap
   a. Almacena pares **clave-valor en una tabla hash,** pero **mantiene un orden de inserción o de acceso.**
   b. **Permite claves y valores null.**
   c. Ofrece un **rendimiento similar a HashMap.**
   d. **Uso común:** Cuando necesitas mantener el orden en que se insertaron los elementos o el orden de acceso.
   e. ***Ejercicio:**
      i. ...
      ii. Intentar poner una pareja que no está de primera en primer lugar.

4. Otras
    a. **Hashtable:** similar a HashMap pero thread-safe y no permite valores null.
    b. **ConcurrentHashMap:** versión optimizada de Hashtable.
    c. **WeakHashMap:** si una clave no se utiliza fuera del mapa esta puede ser recolectada por el Garbage Collector.
    d. **EnumMap:** especializada en el uso de enumerados como claves y mantiene el orden natural de las claves.

# Ejercicio complejo

Imagina que trabajas en un almacén que gestiona productos electrónicos. Cada producto tiene un ID único, un nombre y su precio. Requisitos:

1. **Gestión de productos:**
    a. Permitir la búsqueda rápida de un producto por su ID.
2. **Orden de productos:**
    a. Se necesita ordenar los productos por su nombre y, en el caso de que haya nombres repetidos, por su ID.
    b. Se necesita saber el orden de introducción de los productos y cúal fue el último producto almacenado.
3. **Gestión de descuentos:**
    a. Añadir a dos productos un descuento, el descuento deberá estar ligado al nombre del producto (clave=nombre y valor=descuento), se requiere almacenar solamente los productos con descuento.
    b. Aplicar el descuento en el precio del producto sobre una tabla que contenga todos los productos del almacén.

*ACLARACIÓN: La estructura de los mapas no tiene que ser la misma en todos, es decir, que para el apartado 1 puedes hacer mapa de este modo: HashMap <Producto, Double> (Double representaría el precio) y para el apartado 2 otro de tipo: HashMap <String, Producto> (String representaría el ID). El único requisito es que aparezcan los tres atributos de Producto: id, nombre y precio, para eso se tendrá que crear diferentes constructores.