Introducción a Map

¡Comencemos!

¿Qué aprenderás?

- Aplicar el uso de Map para resolver problemas cotidianos dentro del mundo de la programación.
- Crear distintas implementaciones de Map para ocupar métodos como agregar, eliminar y recorrer datos.

Introducción

La interfaz Map representa una asignación entre una clave y un valor. La interfaz de map no es un subtipo de Collection. Por lo tanto, se comporta un poco diferente del resto de los tipos de colecciones. Un map no puede contener claves duplicadas y cada clave puede mapearse a más de un valor.

Algunas implementaciones permiten una clave nula y valor nulo (como HashMap y LinkedHashMap), pero otras no (como TreeMap). El orden de un mapa depende de la implementación, por ejemplo, TreeMap y LinkedHashMap tienen un orden predecible, mientras que HashMap no. Hay dos interfaces para implementar Map en Java: Map y SortedMap, y tres clases: HashMap, TreeMap y LinkedHashMap.

¡Vamos con todo!

Jerarquía en Map

La plataforma Java contiene tres implementaciones de mapas de uso general: HashMap, TreeMap y LinkedHashMap. Su comportamiento y rendimiento son precisamente análogos a HashSet, TreeSet y LinkedHashSet.

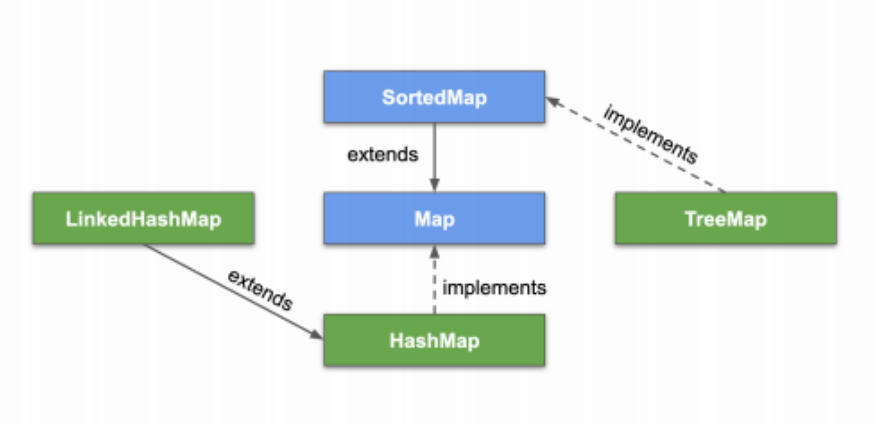

Imagen 1. Jerarquía de Map.
Fuente: Desafío Latam

¿Por qué y cuándo usar Maps?

Los mapas son perfectos para usar con la asociación de valores clave. Por ejemplo, cuando uno busca una palabra en el diccionario, nos guiamos por la letra del abecedario para buscar la palabra específica. Map hace algo similar y utiliza claves para realizar la búsqueda cuando desean recuperar y actualizar elementos. Algunos ejemplos son:

- Un mapa de códigos de error y sus descripciones.
- Un mapa de códigos postales y ciudades.
- Un mapa de clases y estudiantes. Cada clase está asociada con una lista de estudiantes.

Resumiendo

La gran variedad de implementaciones que hemos visto previamente, nos sirve para manejar distintas estructuras de datos e ir definiendo su funcionalidad en base a las características que ofrecen. A Continuación mostraremos un resumen de cuándo ocupar cada uno de estos métodos.

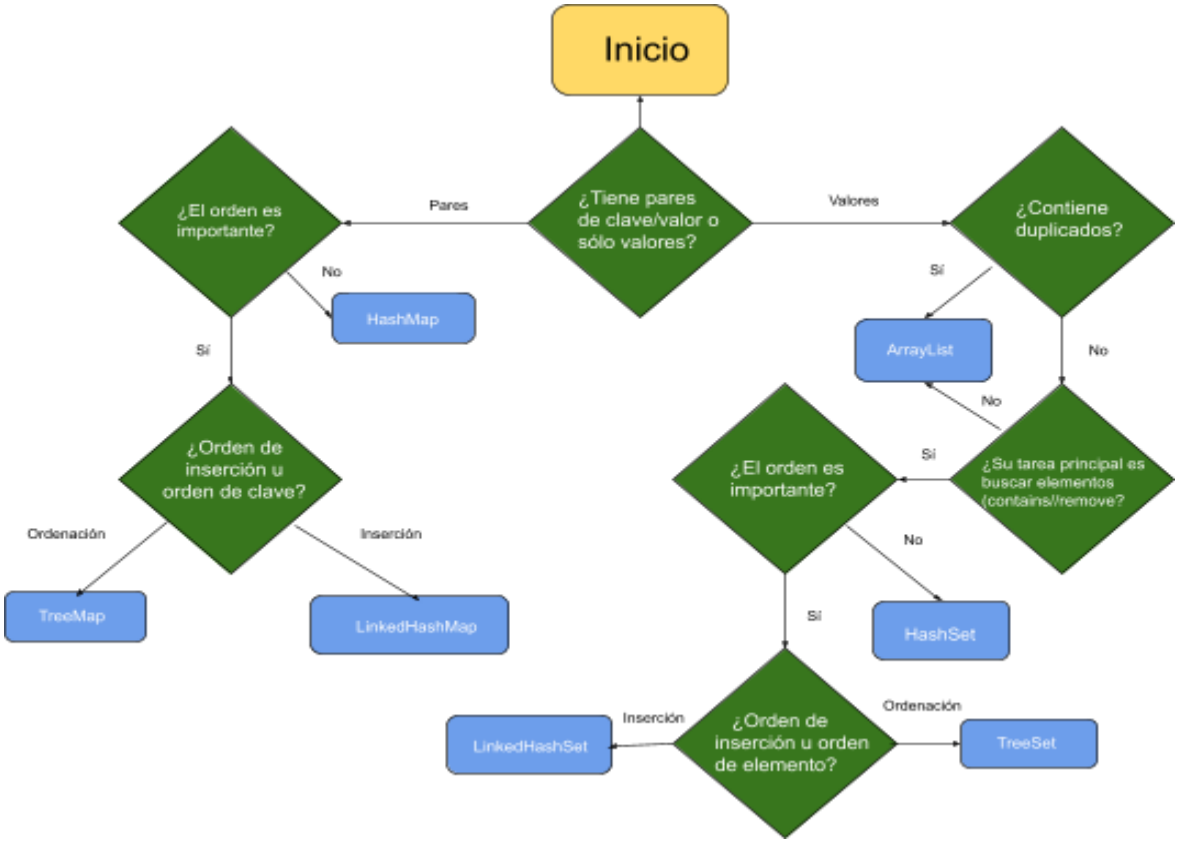

Imagen 2. Resumen de funcionalidades.
Fuente: Desafío Latam

Ejercicio guiado: Métodos en la interfaz de Map

Seguiremos usando referencia geoespacial, pero esta vez con enfoque a los planetas.

- **Paso 1**: Para crear una TreeMap<>() del tipo Map, hay que importar su implementación desde "util.java.TreeMap" en la parte superior de la clase y luego instanciarla, como se muestra en la siguiente imagen:

```
import java.util.TreeMap;
import java.util.Map;

            Map<String, Integer> planetas = new TreeMap<>();
```

- **Paso 2**: Para incorporar elementos a este TreeMap, podemos utilizar el método put(). Este método se utiliza para insertar nuevos valores, con una clave y asignándole un valor. Por ejemplo, la clave sería el nombre del planeta y los valores a entregar serían los años luz que se encuentran de la tierra.

```
        Map<String, Integer> planetas = new TreeMap<>();
        planetas.put("Mercurio", 10);
        planetas.put("Venus", 20);
        planetas.put("Marte", 15);
        planetas.put("Jupiter", 50);
        System.out.println(planetas);
        ---------------------------
        Impresión en pantalla:
        [Jupiter = 50, Marte = 15, Mercurio = 10, Venus = 20]
```

- **Paso 3**: Para eliminar un objeto, este caso un planeta, podemos usar el método remove(). Este método se utiliza para eliminar la entrada de una clave específica.

```
        planetas.remove("Venus");
        System.out.println(planetas);
        ---------------------------
        Impresión en pantalla:
        [Jupiter = 50, Marte = 15, Mercurio = 10]
```

- **Paso 4**: Para obtener un elemento desde Map, podemos utilizar el método get(). Este método se utiliza para devolver el valor de una clave específica. En este caso, nos devolverá 50 que son los años luz desde la tierra a Júpiter.

```
System.out.println(planetas.get("Jupiter"));
---------------------------
Impresión en pantalla:
[50]
```

- **Paso 5**: Para ver si una clave determinada aún está presente en la colección, podemos usar el método containsKey(). Este método se utiliza para buscar una clave específica, en este caso buscaremos el planeta Tierra y Júpiter.

```
System.out.print(planetas.containsKey(“Tierra”));
System.out.print(planetas.containsKey(“Jupiter”));
---------------------------
Impresión en pantalla:
[false] //Significa que no está en la colección
[true]  //La podemos encontrar en la colección
```

- **Paso 6**: Para retornar una o más claves del Map, podemos usar el método keySet(). Este método se utiliza para devolver la vista tipo Set que contiene todas las claves. En este caso por ejemplo ocuparemos un forEach() para recorrer toda la colección.

```
planetas.keySet().forEach(System.out.print);
---------------------------
Impresión en pantalla:
[Júpiter, Marte, Mercurio]
```

- **Paso 7**: Para obtener aquellos valores de planetas que tengan una distancia menor a 16 años luz, podemos usar el método entrySet(). Este método se utiliza para devolver la vista tipo Set que contiene todas las claves y valores, en este caso por ejemplo ocuparemos un forEach() para recorrer toda la colección e imprimir aquellos elementos que cumplan con el requisito de filtrado.

```
            planetas.entrySet().stream().filter(aniosLuz->
aniosLuz.getValue()<16).forEach(System.out::print);
           --------------------------
           Impresión en pantalla:
           [Marte = 15, Mercurio = 10]
```