Programación orientada a objetos

Introducción a Set, Queue y Map

{desafío} latam_

Utilizar principios básicos de diseño orientado a objetos para la implementación de una pieza de software acorde al lenguaje Java para resolver un problema de baja complejidad

- Unidad 1: Flujo, ciclos y métodos
- Unidad 2: Arreglos y archivos
- Unidad 3: Programación orientada a objetos
- Unidad 4: Pruebas unitarias y TDD

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Aplicar el uso de Set para resolver problemas cotidianos dentro del mundo de la programación.*
- *Aplicar el uso de Queue para resolver problemas cotidianos dentro del mundo de la programación.*
- *Aplicar el uso de Map para resolver problemas cotidianos dentro del mundo de la programación.*

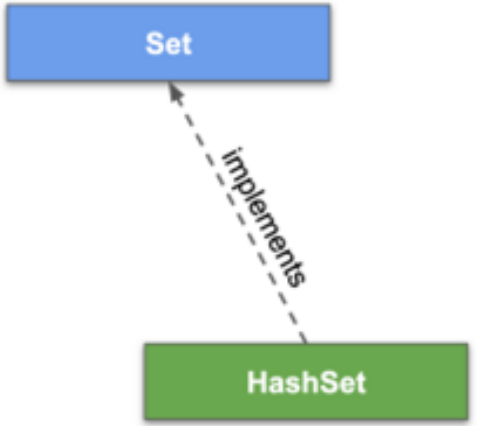
{desafío} latam_

¿Qué otras maneras tenemos para resolver problemas cotidianos?

/* Set, Queue y Map */

Set

- Es una interfaz que se extiende desde Collections.
- Es uno de los tipos de colecciones más desordenado en relación a los objetos, en donde a diferencia de List no se pueden almacenar valores duplicados.
- Agrega un contrato más fuerte sobre el comportamiento de las operaciones equals y hashCode, permitiendo que las instancias de Set se comparen significativamente incluso si sus tipos de implementación difieren.
- Dos instancias de Set son iguales si contienen los mismos elementos.



Set

Implementaciones de propósito general

- HashSet
- TreeSet
- LinkedHashSet

Además, Set tiene varios métodos de acceso posicional como add, remove, clear y size, entre muchos otros, que permiten mejorar el uso de esta interfaz.

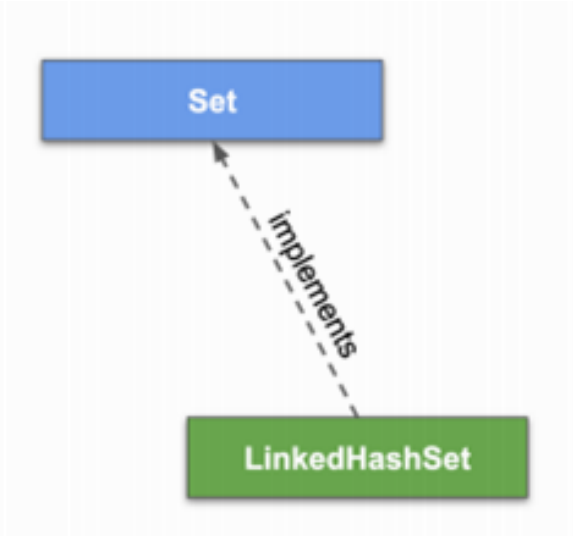
Creando un HashSet

1. Definir un Set del tipo String, cuyo nombre será capitales e instanciamos como una HashSet.
2. Agregar algunos lenguajes de programación para mostrar que el elemento “Java” no se repite.

```
import java.util.Set;
import java.util.HashSet;
	Set<String> languages = new HashSet<>();
	languages.add("Java");
	languages.add("Go");
	languages.add("Erlang");
	languages.add("Java");
	languages.add("Elixir");
	languages.add("Fortran");
	System.out.println(languages);
//[Java, Go, Erlang, Elixir, Fortran]
```

LinkedHashSet

Se implementa como una tabla hash con una lista vinculada que lo ejecuta, ordena sus elementos según el orden en que se insertaron en el conjunto (orden de inserción).



LinkedHashSet

Ejemplo

```
import java.util.LinkedHashSet;
Set<String> programmers = new LinkedHashSet<>();
programmers.add("James Gosling");
programmers.add("Martin Odersky");
programmers.add("Rich Hickey");
programmers.add("Larry Wall");
programmers.add("Graydon Hoare");
System.out.println(programmers);
//[James Gosling, Martin Odersky, Rich Hickey, Larry Wall, Graydon Hoare]
```

TreeSet

Almacena sus elementos en un árbol rojo-negro, es decir, ordena sus elementos en función de sus valores.

Es sustancialmente más lento que HashSet

Ejercicio guiado

Capitales del mundo

Paso 1

Para explicar lo que acabamos de mencionar, realizaremos un ejemplo guiado con las capitales sudamericanas.

```
import java.util.TreeSet;

     Set<String> capitales = new TreeSet<>();
     capitales.add("Buenos Aires");
     capitales.add("Brasilia");
     capitales.add("Asunción");
     capitales.add("Lima");
     System.out.println(capitales);

     //[Asunción, Brasilia, Buenos Aires, Lima]
```

Capitales del mundo

Paso 2

Para ello crearemos otro HashSet llamado capitales2 y le incorporaremos 5 capitales más.

```
Set<String> capitales = new TreeSet<>();
capitales.add("Buenos Aires");
capitales.add("Brasilia");
capitales.add("Asunción");
capitales.add("Lima");
Set<String> capitales2 = new HashSet<>(Arrays.asList("Caracas", "Bogotá",
"Montevideo","Quito", "Brasilia"));
Set<String> capitalesUnidas = new TreeSet<>(capitales);
capitalesUnidas.addAll(capitales2);
System.out.println(capitalesUnidas);
---------------------------
Impresión en pantalla:
[Asunción, Bogotá ,Brasilia, Buenos Aires, Caracas, Lima,  Montevideo, Quito]
```

Capitales del mundo

Paso 3

Si queremos borrar una colección completa de la lista, debemos usar el método removeAll()

```
Set<String> removerCapitales = new HashSet<>(capitales);
removerCapitales.removeAll(capitales2);
System.out.println(removerCapitales);
---------------------------
Impresión en pantalla:

[Asunción,Brasilia, Buenos Aires, Lima]
```

Capitales del mundo

Paso 4

Para encontrar valores en común entre colecciones, se puede usar el método retainAll()

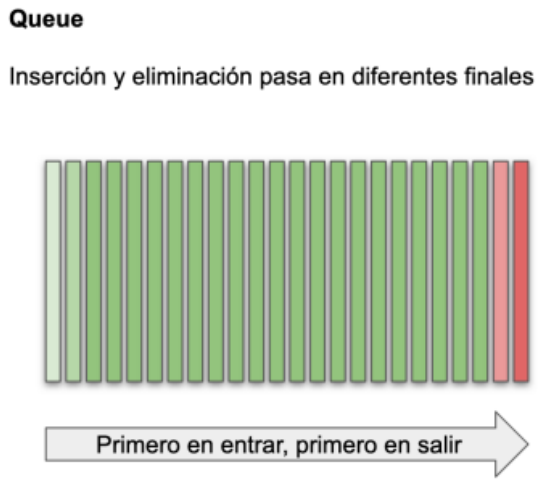
```
Set<String> interseccionCapitales = new HashSet<>(capitales);
interseccionCapitales.retainAll(capitales2);
System.out.println(interseccionCapitales);
---------------------------
Impresión en pantalla:
[Brasilia]
```

Queue

Es una colección ideal para contener elementos antes del procesamiento.

Es una estructura lineal que sigue un orden particular en el que se realizan las operaciones.

Sigue el principio: FIFO (First In First Out) "Primero en entrar, primero en salir”



Queue

Clases más comunes

- PriorityQueue
- LinkedList
- PriorityBlockingQueue

Ejercicio guiado

Métodos de acceso posicional

Paso 1

Para crear una LinkedList<>() del tipo Queue hay que importar su implementación desde “util.java.LinkedList” en la parte superior de la clase y luego instanciarla.

```
import java.util.LinkedList;

     Queue continentes = new LinkedList<>();
```

Métodos de acceso posicional

Paso 2

Para agregar elementos a encolar, podemos utilizar el método add()

```
Queue continentes = new LinkedList<>();
continentes.add("África");
continentes.add("América");
continentes.add("Europa");
continentes.add("Oceanía");
continentes.add("Asia");
continentes.add("Antártica");
---------------------------
Impresión en pantalla:
[África, América, Europa, Oceanía, Asia, Antártica]
```

Métodos de acceso posicional

Paso 3

Para poner un elemento específico del encolamiento continentes, podemos usar el método remove()

```
System.out.println(continentes.remove("Antártica"));
System.out.println(continentes);
---------------------------
Impresión en pantalla:
[África, América, Europa, Oceanía, Asia]
```

Métodos de acceso posicional

Paso 4

Para eliminar un encabezado, es decir, el primero de la lista se elimina, podemos usar el método poll().

```
System.out.println(continentes.poll());
System.out.println(continentes);
---------------------------
Impresión en pantalla:
[América, Europa, Oceanía, Asia]
```

Métodos de acceso posicional

Paso 5

Para obtener el encabezado de la cola sin eliminarlo podemos usar el método peek()

```
System.out.println("peek : " + continentes.peek());
System.out.println(continentes);
---------------------------
Impresión en pantalla:
peek: [América]
```

Métodos de acceso posicional

Paso 6

Para encontrar un elemento, al igual que peek se puede hacer sin eliminar el objeto con el método element()

```
System.out.println("element: "+continentes.element());
System.out.println(continentes);
---------------------------
Impresión en pantalla:
element: [América]
```

Métodos de acceso posicional

Paso 7

Para encontrar el tamaño de un encolado, podemos utilizar el método size()

```
System.out.println(continentes.size());
---------------------------
Impresión en pantalla:
[4]
```

Map

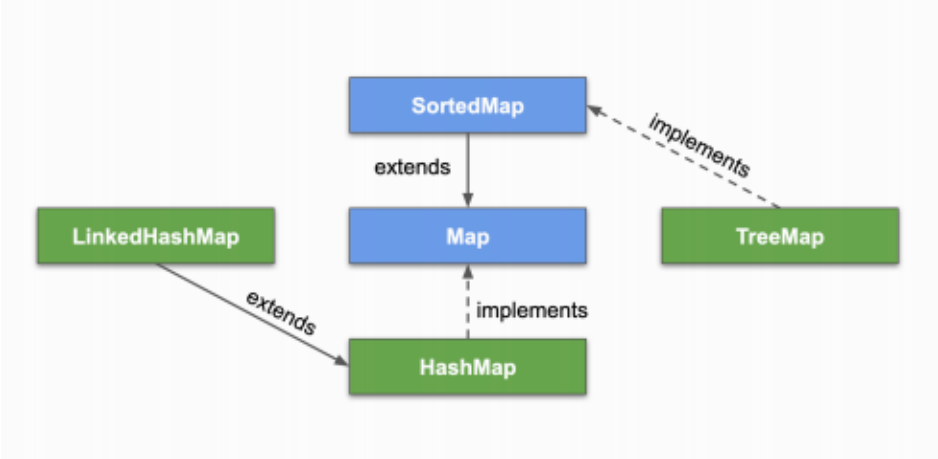
- Representa una asignación entre una clave y un valor.
- Se comporta un poco diferente del resto de los tipos de colecciones.
- No puede contener claves duplicadas y cada clave puede mapearse a más de un valor.

Jerarquía Map

La plataforma Java contiene tres implementaciones de mapas de uso general:

- HashMap
- TreeMap
- LinkedHashMap.

Su comportamiento y rendimiento son precisamente análogos a HashSet, TreeSet y LinkedHashSet.



¿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.

Ejercicio guiado

Métodos en la interfaz de Map

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

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

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

Métodos en la interfaz de Map

Paso 2

Para incorporar elementos a este TreeMap, podemos utilizar el método put()

```
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]
```

Métodos en la interfaz de Map

Paso 3

Para eliminar un objeto, en este caso un planeta, podemos usar el método remove()

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

Métodos en la interfaz de Map

Paso 4

Para obtener un elemento desde Map, podemos utilizar el método get()

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

Métodos en la interfaz de Map

Paso 5

Para ver si una clave determinada aún está presente en la colección, podemos usar el método containsKey()

```
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
```

Métodos en la interfaz de Map

Paso 6

Para retornar una o más claves del Map, podemos usar el método keySet()

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

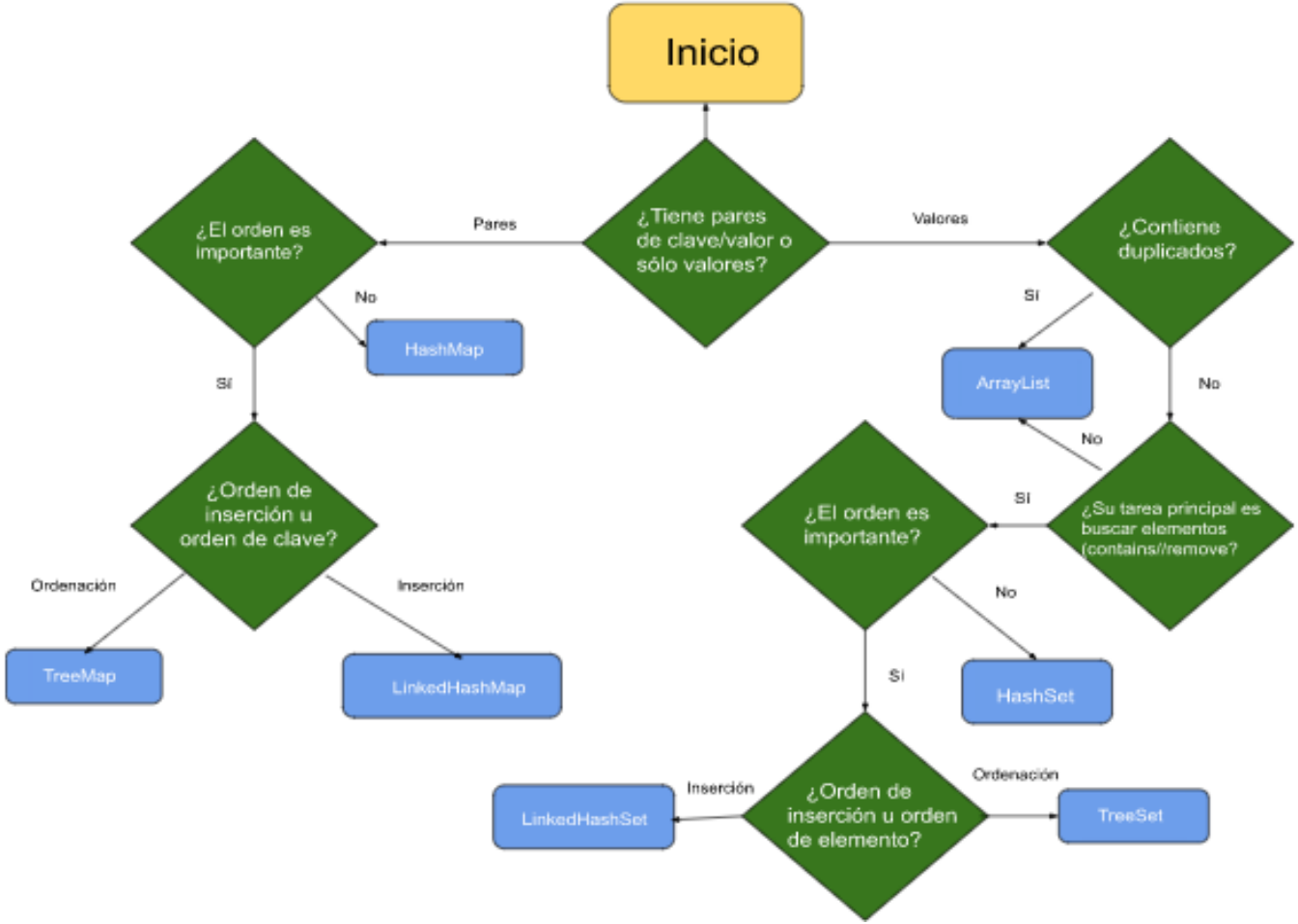
Métodos en la interfaz de Map

Paso 7

Para obtener aquellos valores de planetas que tengan una distancia menor a 16 años luz, podemos usar el método entrySet()

```
for (Map.Entry<String, Integer> entry : planetas.entrySet()) {
     String key = entry.getKey();
     Integer value = entry.getValue();
     System.out.println(key);
     System.out.println(value);
}
```

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


Inicio
¿Tiene pares de clave/valor o sólo valores?
Pares
Valores
¿El orden es importante?
No
HashMap
Sí
¿Orden de inserción u orden de clave?
Ordenación
Inserción
TreeMap
LinkedHashMap
¿Contiene duplicados?
Sí
ArrayList
No
¿Su tarea principal es buscar elementos (contains//remove?
Sí
No
¿El orden es importante?
Sí
HashSet
No
¿Orden de inserción u orden de elemento?
Inserción
Ordenación
LinkedHashSet
TreeSet

¿Cuáles métodos pertenecen a la interface Set?

{desafío} latam_

¿Cuál es la funcionalidad que ocupa Queue?

¿Cuáles son los elementos claves que componen la interfaz Map?

Próxima sesión...

- *Desafío guiado*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales