Estructuras de datos

Estructuras de datos

Estructuras de datos

En programación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente.

Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

https://es.wikipedia.org/wiki/Estructura_de_datos

Tipos genéricos

Tipos genéricos

Un tipo genérico es una clase o interfaz genérico parametrizado mediante tipos de datos.

https://docs.oracle.com/javase/tutorial/java/generics/types.html


HELP!

Clase no genérica

```
public class Contenedor {

    private int dato;

    public Contenedor(int dato) {
        this.dato = dato;
    }

    public int getDato() {
        return dato;
    }

    public void setDato(int dato) {
        this.dato = dato;
    }
}
```

Clase no genérica

```
public static void main(String[] args) {

    Contenedor c = new Contenedor(7);

    System.out.println("El valor almacenado es: " + c.getDato());
}
```

Clase no genérica

```
public class Contenedor {

    private double dato;

    public Contenedor(double dato) {
        this.dato = dato;
    }

    public double getDato() {
        return dato;
    }

    public void setDato(double dato) {
        this.dato = dato;
    }
}
```

Clase genérica

```
public class Contenedor<T> {

    private T dato;

    public Contenedor(T dato) {
        this.dato = dato;
    }

    public T getDato() {
        return dato;
    }

    public void setDato(T dato) {
        this.dato = dato;
    }
}
```

Clase genérica

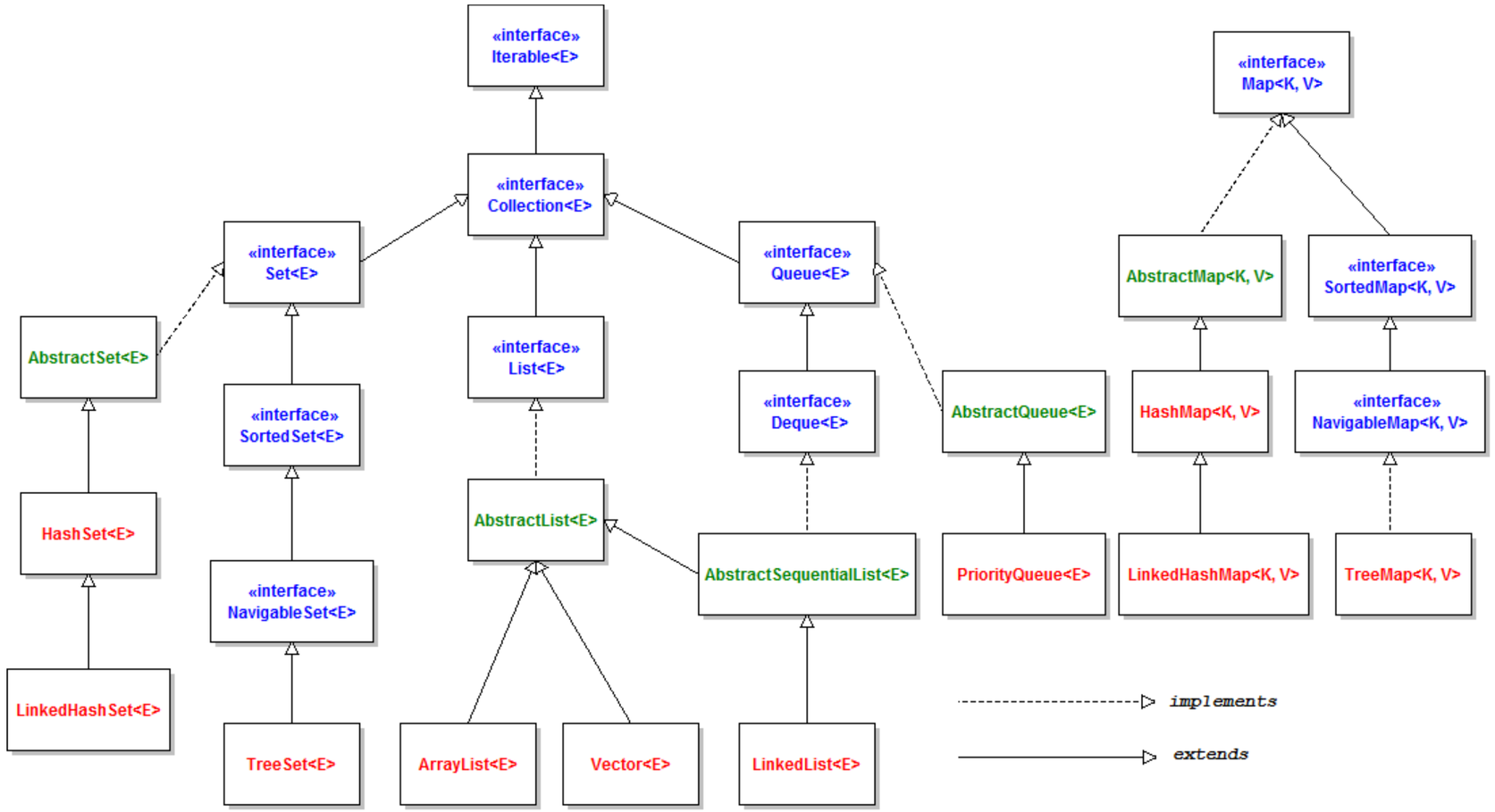
```
public static void main(String[] args) {

    Contenedor<String> c = new Contenedor<>("ABCDE");

    System.out.println("El valor almacenado es: " + c.getDato());
}
```

Convención de nombrado

Letra	Significado
E	Element
K	Key
N	Number
T	Type
V	Value

Colecciones

Java Collection Framework



http://www.codejava.net/java-core/collections/overview-of-java-collections-framework-api-uml-diagram

Java Collection Framework

Estructura	Interfaz	Implementación

Lista	`List<E>`	`ArrayList<E>`
		`LinkedList<E>`
		`Vector<E>`

Conjunto	`Set<E>`	`HashSet<E>` `TreeSet<E>`

Diccionario	`Map<K,V>`	`HashMap<K,V>`
		`HashTable<K,V>`
		`LinkedHashMap<K,V>`
		`TreeMap<K,V>`

Listas

Interfaz List<E>

Lista

- Colección de elementos en una secuencia particular que mantienen un orden y permite duplicados

Java Collection Framework

Estructura	Interfaz	Implementación

Lista	`List<E>`	`ArrayList<E>`
		`LinkedList<E>`
		`Vector<E>`

Conjunto	`Set<E>`	`HashSet<E>` `TreeSet<E>`

Diccionario	`Map<K,V>`	`HashMap<K,V>`
		`HashTable<K,V>`
		`LinkedHashMap<K,V>`
		`TreeMap<K,V>`

Clase ArrayList<E>

- El acceso a un elemento en particular es eficiente
- Para eliminar un elemento, se ha de mover toda la lista para eliminar ese "hueco"

Clase LinkedList<E>

- Los elementos están conectados con el anterior y el posterior
- Es eficiente mover/eliminar elementos de la lista, simplemente moviendo/eliminando sus referencias hacia otros elementos
- La desventaja es que para usar el elemento N de la lista, debemos recorrer N elementos de la lista hasta encontrarlo

Clase Vector<E>

- Es igual que `ArrayList<E>`, pero sincronizado
- Si se utiliza el mismo vector en dos hilos al mismo tiempo, se garantiza que el acceso concurrente se hace de forma ordenada
- Esta característica hace que sea mucho más lento que `ArrayList<E>` pero sea seguro utilizarlo en programas multihilo

Declaración de una lista

```
List<String> nombres = new ArrayList<>();

List<String> masNombres = new LinkedList<>();
```

(Algunos) métodos de List<E>

Método	Operación
`add(Object o)`	Añade un objeto al final de la lista
`add(int indice, Object o)`	Añade un objeto a la lista en la posición indicada
`get(int indice)`	Devuelve el objeto de la lista de la posición indicada
`remove(int indice)`	Elimina el objeto de la lista pasado por parámetro
`clear()`	Elimina todos los elementos de la lista
`indexOf(Object o)`	Devuelve la posición de la primera vez que un elemento coincida con el objeto pasado por parámetro Si el elemento no se encuentra devuelve -1
`lastIndexOf(Object o)`	Devuelve la posición de la última vez que un elemento coincida con el objeto pasado por parámetro Si el elemento no se encuentra devuelve -1
`size()`	Devuelve el número de elementos de la lista
`contains(Object o)`	Devuelve verdadero si en la lista aparece el objeto pasado por parámetro, para lo cual utiliza intrínsecamente el método `equals()`

`https://docs.oracle.com/javase/8/docs/api/java/util/List.html`

Operaciones sobre una lista

```
nombres.add("Jorge");
nombres.add("María");
nombres.add("Olga");

System.out.println(nombres.get(1)); // María

nombres.remove(0);

System.out.println(nombres.size()); // 2
```

¿ArrayList<E> o LinkedList<E>?

- Si predomina el acceso aleatorio, `ArrayList<E>`
- Si predomina el insertar y eliminar elementos, `LinkedList<E>`
- Si necesitamos ambas características, habrá que hacer pruebas de rendimiento y elegir la combinación adecuada o incluso duplicar las estructuras dependiendo del programa

Recorrido de una lista

Recorrido de una lista

- Tratamos todos los elementos de la lista, de uno en uno

Recorrido de una lista

```
public class Ubicacion {

    private double latitud;
    private double longitud;

    public double getLatitud() {
        return latitud;
    }

    public void setLatitud(double latitud) {
        this.latitud = latitud;
    }

    public double getLongitud() {
        return longitud;
    }

    public void setLongitud(double longitud) {
        this.longitud = longitud;
    }
}
```

Recorrido de una lista

```
List<Ubicacion> ubicaciones = new ArrayList<>();

Ubicacion temp = new Ubicacion();
temp.setLatitud(42.867564);
temp.setLongitud(-2.677465);

ubicaciones.add(temp);

...
```

for

```
for (int i = 0; i < ubicaciones.size(); i++) {
    System.out.println("Lat: " + ubicaciones.get(i).getLatitud());
    System.out.println("Lon: " + ubicaciones.get(i).getLongitud());
}
```

for(each)

```
for (Ubicacion ub : ubicaciones) {
    System.out.println("Lat: " + ub.getLatitud());
    System.out.println("Lon: " + ub.getLongitud());
}
```

Iterador

```
Iterator<Ubicacion> it = ubicaciones.iterator();

while (it.hasNext()) {
    Ubicacion ub = it.next();

    System.out.println("Lat: " + ub.getLatitud());
    System.out.println("Lon: " + ub.getLongitud());
}
```

.forEach() y expresión λ

Java 8+

```
ubicaciones.forEach(ub -> {
    System.out.println("Lat: " + ub.getLatitud());
    System.out.println("Lon: " + ub.getLongitud());
});
```

Igualdad entre objetos

Igualdad entre objetos

Método	Operación
`==`	Compara dos referencias entre sí, para ver si se trata del mismo objeto
`equals()`	Retorna un valor booleano que indica `true` cuando el objeto activo es igual al objeto que recibe como parámetro
`hashcode()`	Retorna un entero que representa el valor *hash* que identifica a un objeto de forma única

Igualdad entre objetos

- La implementación por defecto de `equals()` y `hashcode()` retorna el resultado en función de las direcciones de memoria de los objetos

- Si queremos comparar los contenidos de los objetos (sus propiedades) habrá que sobrecargar los métodos adaptándolos al caso concreto

- Si usamos == o no sobrecargamos `equals()`, sólo estaremos comparando referencias, no el contenido de los objetos

Comparación de objetos

Interfaz Comparable

```
public int compareTo(Object o)
```

- Para que dos objetos se puedan comparar, nuestra clase debe implementar el interfaz `Comparable`

Interfaz Comparable

Valor de retorno de `compareTo()`	Significado
Negativo	`thisObject < anotherObject`
0	`thisObject == anotherObject`
Positivo	`thisObject > anotherObject`

Conjuntos

Interfaz Set<E>

Conjunto

- Colección de elementos sin orden particular y que no permite duplicados
- Si se intenta añadir un elemento duplicado, se descarta
- Necesita saber cuando dos elementos son iguales, por lo que hay que redefinir `equals()` y `hashCode()`

Java Collection Framework

Estructura	Interfaz	Implementación

Lista	`List<E>`	`ArrayList<E>`
		`LinkedList<E>`
		`Vector<E>`

Conjunto	`Set<E>`	`HashSet<E>` `TreeSet<E>`

Diccionario	`Map<K,V>`	`HashMap<K,V>`
		`HashTable<K,V>`
		`LinkedHashMap<K,V>`
		`TreeMap<K,V>`

Clase HashSet<E>

- Implementa la interface `Set<E>`
- Hace distinción de identidad, sólo pueden existir elementos diferentes
- No se respeta el orden en el que se insertan los elementos
- El tamaño del conjunto se adapta dinámicamente a lo que se necesite

Clase TreeSet<E>

- Implementa la interface `Set<E>`
- Hace distinción de identidad, sólo pueden existir elementos diferentes
- Mantiene los elementos ordenados utilizando el orden natural
- El tamaño del conjunto se adapta dinámicamente a lo que se necesite

Declaración de un conjunto

```
Set<Integer> numeros = new HashSet<>();

Set<Integer> masNumeros = new TreeSet<>();
```

(Algunos) métodos de Set<E>

Método	Operación
`add(Object o)`	Añade el objeto pasado por parámetro al conjunto, siempre que éste no exista ya, y devuelve un booleano
`clear()`	Elimina a todos los elementos del conjunto
`contains(Object o)`	Devuelve `true` si el conjunto contiene el objeto pasado por parámetro Para ello, se compara de forma interna con el método `equals()` o con el método `hashCode()`
`isEmpty()`	Devuelve `true` si el conjunto está vacío
`iterator()`	Devuelve un iterador que permite recorrer el conjunto
`remove(Object o)`	Elimina el objeto pasado por parámetro, si existe, y devuelve un booleano
`size()`	Devuelve un entero que es el número de elementos del conjunto

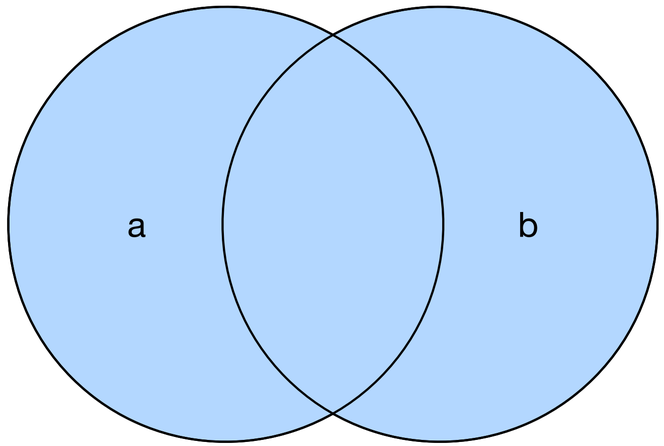
`https://docs.oracle.com/javase/8/docs/api/java/util/Set.html`

Operaciones sobre conjuntos

```
numeros.add(3);
numeros.add(2);
numeros.add(7);
numeros.add(7);
numeros.add(5);

for (int n : numeros) {
    System.out.format("%d ",n);  // 2 3 5 7
}
```

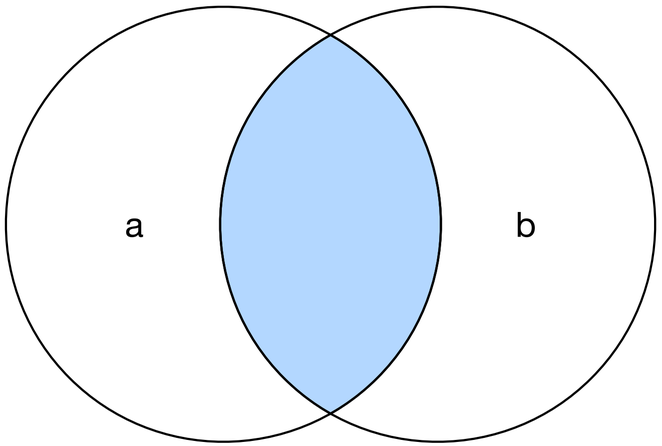
Operaciones entre conjuntos

Unión



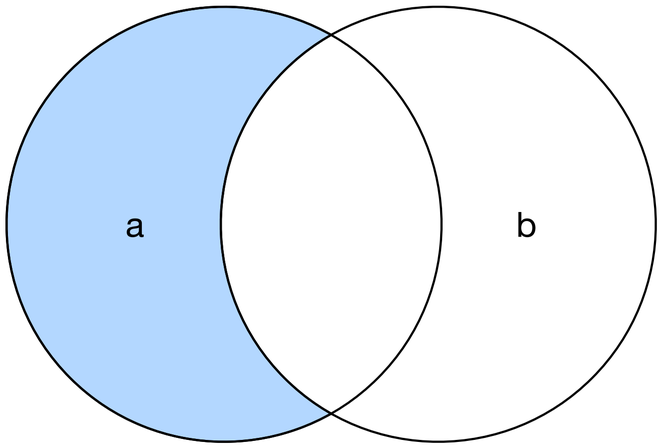

```
a.addAll(b);
```

Intersección




```
a.retainAll(b);
```

Diferencia




```
a.removeAll(b);
```

Operaciones entre conjuntos

```
masNumeros.add(5);
masNumeros.add(1);
masNumeros.add(2);

masNumeros.retainAll(numeros);

for (int n : masNumeros) {
    System.out.format("%d ",n);  // 2 5
}
```

Diccionarios

Interfaz Map<K,V>

Diccionario

- Es una estructura de datos agrupados en parejas clave/valor (key/value)
- También se les llama arrays asociativos
- La clave debe ser única y se emplea para acceder al valor

Java Collection Framework

Estructura	Interfaz	Implementación

Lista	`List<E>`	`ArrayList<E>`
		`LinkedList<E>`
		`Vector<E>`

Conjunto	`Set<E>`	`HashSet<E>` `TreeSet<E>`

Diccionario	`Map<K,V>`	`HashMap<K,V>`
		`HashTable<K,V>`
		`LinkedHashMap<K,V>`
		`TreeMap<K,V>`

Clase HashMap<K,V>

- Mapea claves con valores, pero no permite claves duplicadas
- Tanto la clave como el valor pueden ser cualquier tipo de objeto, y ambos pueden tener el valor `null`
- Esta clase no garantiza el orden de los elementos ni que no puedan cambiar de orden

Clase HashTable<K,V>

- Almacena pares del tipo clave/valor en una tabla de dispersión
- Se proporciona un objeto que sirve como clave y otro como valor (vinculando el valor a la clave)
- Su inserción y búsqueda es rápida

Clase LinkedHashMap<K,V>

- Extiende de `HashMap<K,V>` y utiliza una lista doblemente enlazada para recorrerla en el orden en que se añadieron
- Es ligeramente más rápida a la hora de acceder a los elementos que su superclase, pero más lenta a la hora de añadirlos

Declaración de un diccionario

```
HashMap<String,String> drae = new HashMap<>();
```

(Algunos) métodos de Map<K,V>

Método	Operación
`clear()`	Elimina todos los mapeos del diccionario
`containsKey(Object clave)`	Devuelve `true` si el diccionario contiene un mapeo igual que el objeto pasado por parámetro
`containsValue(Object valor)`	Devuelve `true` si el diccionario mapea a una o más claves el objeto pasado por parámetro
`get(Object clave)`	Devuelve el objeto valor de la clave pasada por parámetro Si el diccionario no encuentra ningún mapeo con esta clave, devuelve `null`
`isEmpty()`	Devuelve true si el diccionario está vacío
`put(Clave clave, Valor valor)`	Asigna el valor pasado con la clave especificada a otro objeto valor
`remove(Object clave)`	Elimina el mapeo que tenga la clave pasada por parámetro, si existe, devolviendo el valor de dicho mapeo
`size()`	Devuelve un entero con el número de mapeos del diccionario

`https://docs.oracle.com/javase/8/docs/api/java/util/Map.html`

Operaciones sobre diccionarios

```
drae.put("casa","Edificio para habitar.");

System.out.println("Definición de Casa: "+drae.get("casa"));
```

Recorrido de un diccionario

Recorrido de un diccionario

```
Map<Integer, String> productos = new HashMap<>();

productos.put(1111, "Tornillos");
productos.put(2222, "Arandelas");
productos.put(4444, "Tuercas");
productos.put(3333, "Muelles");
```

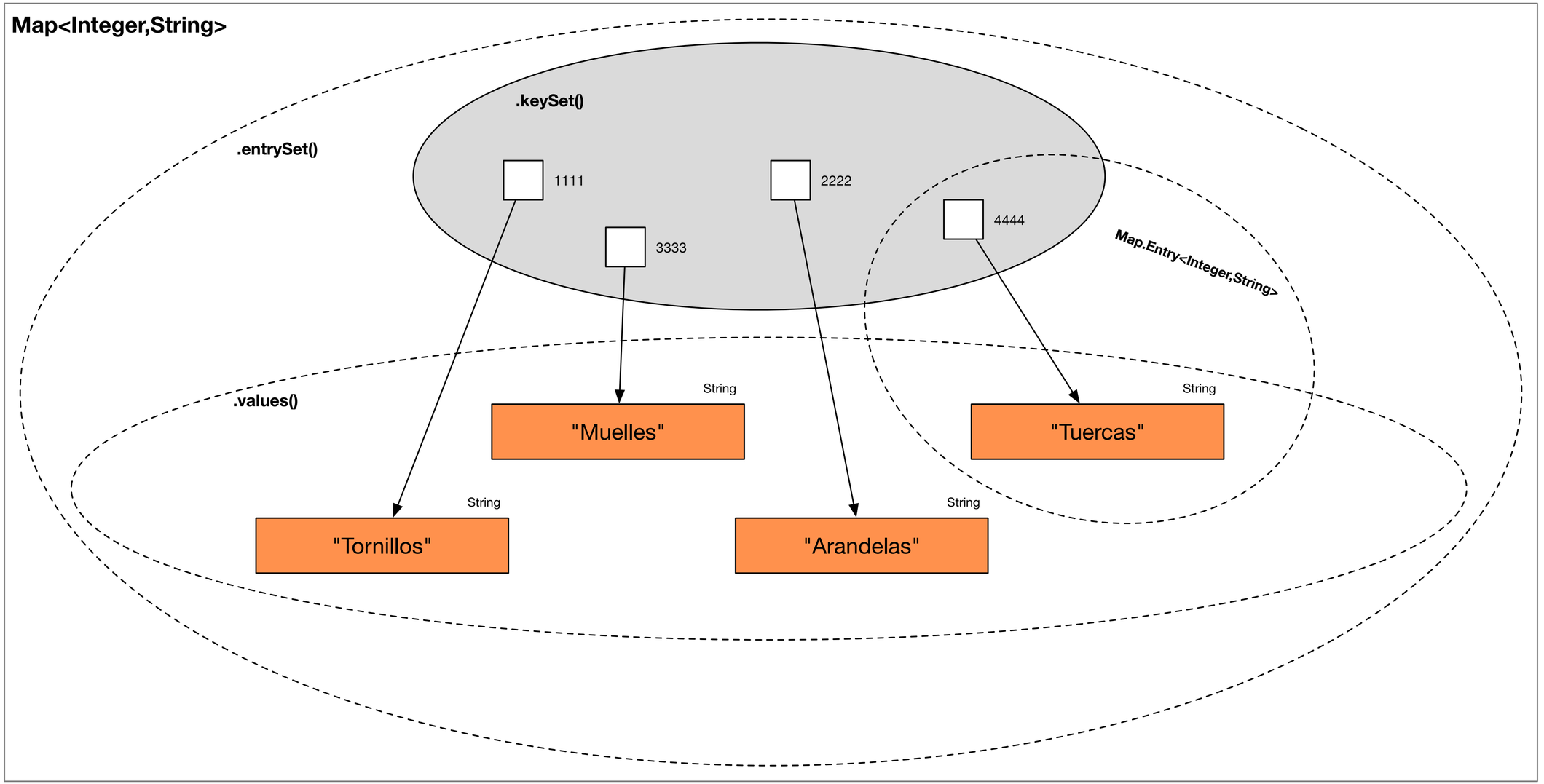
Bucle for recorriendo cada pareja clave-valor

```
for (Map.Entry<Integer, String> producto : productos.entrySet()) {
    System.out.println("Clave: " + producto.getKey()
            + ", Valor: " + producto.getValue());
}
```

Bucle for recorriendo claves o valores

```
for (Integer clave : productos.keySet()) {
    System.out.println("Clave: " + clave);
}

for (String valor : productos.values()) {
    System.out.println("Valor: " + valor);
}
```

Subelementos de un diccionario



Iterador

```
Iterator<Map.Entry<Integer, String>> it = productos.entrySet().iterator();

while (it.hasNext()) {
    Map.Entry<Integer, String> producto = it.next();
    System.out.println("Clave: " + producto.getKey()
            + ", Valor: " + producto.getValue());
}
```

Recorrer las claves y accede

Ineficiente

```
for (Integer clave : productos.keySet()) {
    System.out.println("Clave: " + clave
            + ", Valor: " + productos.get(clave));
}
```

.forEach() y expresión λ

Java 8+

```
productos.forEach((clave, valor) -> {
    System.out.println("Clave: " + clave + ", Valor: " + valor);
});
```

すべての質問は?

Any questions?

Galderarik?

¿Alguna pregunta?

任何問題?

Alguma pergunta?

أي أسئلة؟

질문 있나요?