



Daniel Ivan Anaya Alvarez

Collections Activity

In this activity we are going to see some basic knowledge about the collections that we can find in Java.

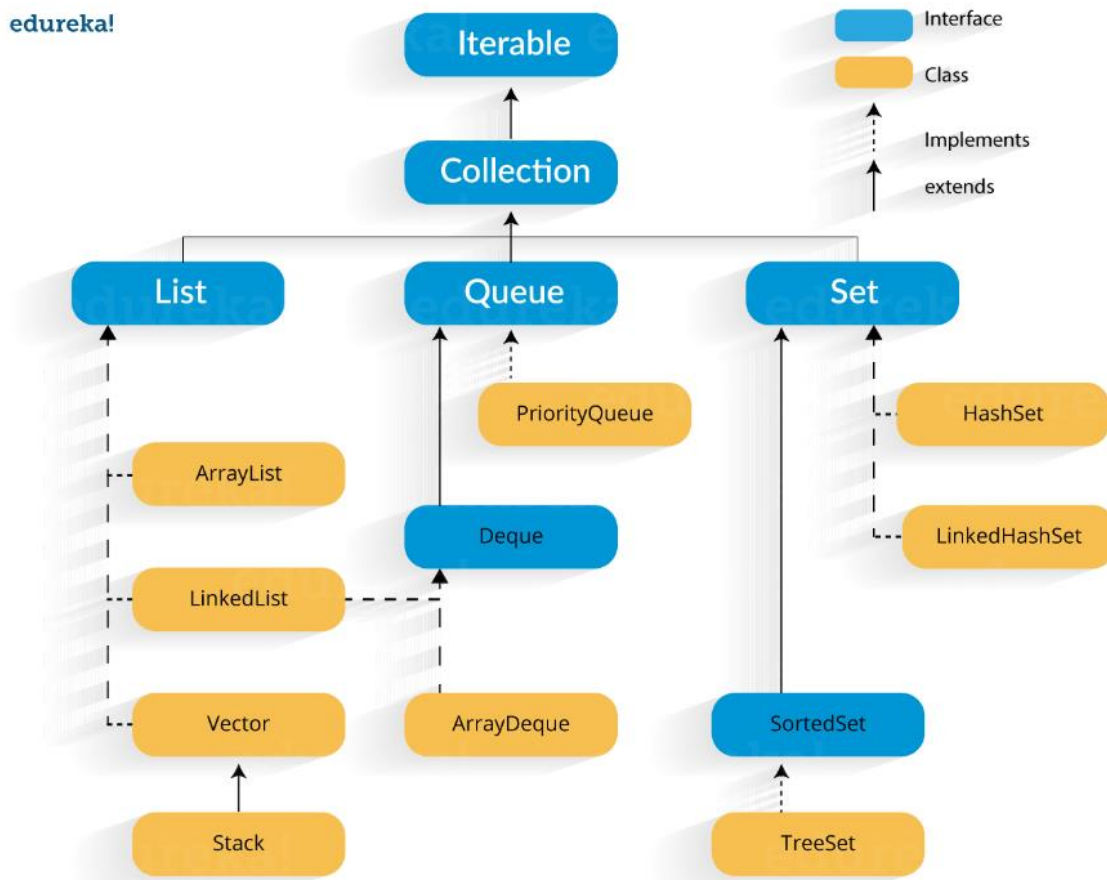


Illustration 1 Flowchart about collections

Iterable (Interface)

The root interface of the collection framework that represents a collection of elements that can be iterated over. Any class implementing Iterable can be used in a "for-each" loop. The way to call this is to use its key method that is: `iterator()`

Collection (Interface)

A more specialized interface that extends Iterable. It represents a group of objects, known as elements. It is the root interface for all other collections like List, Set, and Queue. Its key methods are: `add()`, `remove()`, `size()`, `clear()`, `contains()`

List (Interface)

A type of collection that allows duplicate elements and maintains the order of insertion. It represents an ordered sequence of elements. The way to implements are: ArrayList, LinkedList, vector and stack, i'm going to skip the explanation of vector and stack since we are not going to use it.

- ArrayList: Resizable array implementation of the List interface. It provides fast random access and is generally preferred for read-heavy operations.
- LinkedList: Doubly-linked list implementation of the List interface. It is more efficient than ArrayList for insertion and deletion operations in the middle of the list.

Queue (Interface)

A type of collection designed for holding elements before processing, it typically orders elements in an order of First one and first out manner. The implementations are: PriorityQueue and Deque.

- PriorityQueue: A priority heap implementation of the Queue interface. Elements are ordered based on their natural ordering or by a Comparator provided at queue construction time.
- Deque: A double-ended queue that supports element insertion and removal at both ends.

Set (Interface)

A type of collection that does not allow duplicate elements. The implementations are: HashSet, LinkedHashSet, SortedSet and TreeSet.

- HashSet: Backed by a hash table, it offers constant time performance for basic operations like add, remove, and contains, assuming the hash function disperses elements properly.
- LinkedHashSet: Extends HashSet and maintains a doubly-linked list running through all of its entries, preserving the order of insertion.
- SortedSet: Extends Set and maintains the elements in a sorted order, according to their natural ordering or by a Comparator provided at set creation time.
- TreeSet: It sorts the elements according to their natural ordering or by a Comparator provided at set creation time.

Examples:

I'm going to show some simple examples from each implementation of the interfaces, its going to be a simple Java code and print out the result. for priority queue and array deque and linkedhadset we are going to use some sample codes that can be found on the web.

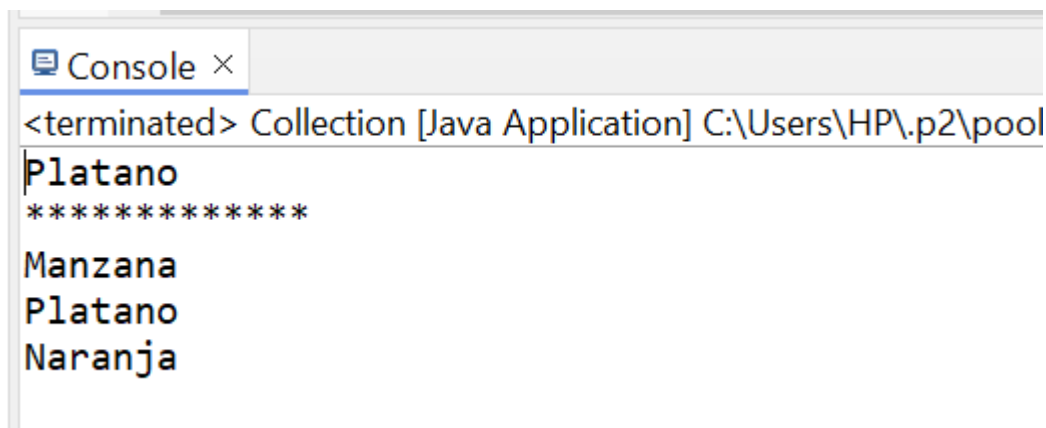
ArrayLists

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

// Agregamos los elementos
arrayList.add("Manzana");
arrayList.add("Platano");
arrayList.add("Naranja");

// Vemos el elemento 1
System.out.println(arrayList.get(1)); // Salida del
segundo elemento debido a que empieza a contar de 0 elemento
System.out.println("*****");
// Imprimimos la lista
for (String fruit : arrayList) {
    System.out.println(fruit);
}
```

ArrayList Print



```
<terminated> Collection [Java Application] C:\Users\HP\.p2\pool
Platano
*****
Manzana
Platano
Naranja
```

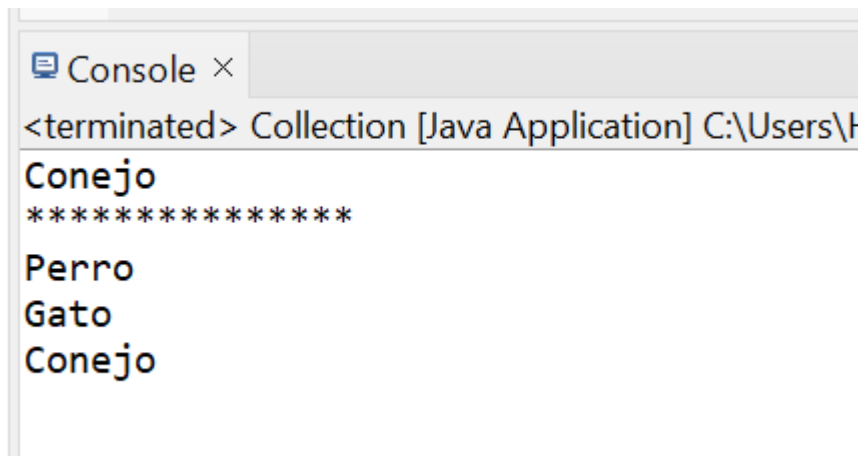
LinkedList

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

// AgregamosElementos
linkedList.add("Perro");
linkedList.add("Gato");
linkedList.add("Conejo");

// Observamos el elementos
System.out.println(linkedList.get(2)); // Salida: Conejo.
Empieza a contar de 0
System.out.println("*****");
// Imprimimos la lista con interacciones
for (String animal : linkedList) {
    System.out.println(animal);
}
```

LinkedListPrint



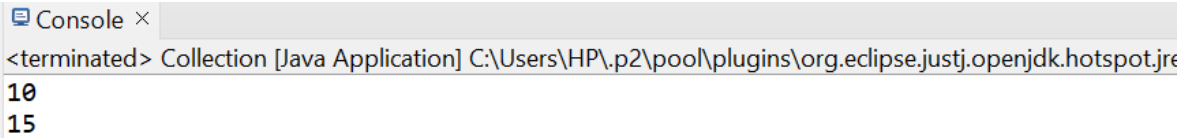
PriorityQueue

```
Queue<Integer> priorityQueue = new PriorityQueue<>();

// Agregamos los elementos
priorityQueue.add(10);
priorityQueue.add(20);
priorityQueue.add(15);

// Accediendo y removiendo los elementos
System.out.println(priorityQueue.poll()); // Salida: 10
(Elemento mas pequeño )
System.out.println(priorityQueue.peek()); // Salida: 15
(Siguiente elemento mas pequeño)
```

PriorityQueue Print



```
<terminated> Collection [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre
10
15
```

ArrayDeque

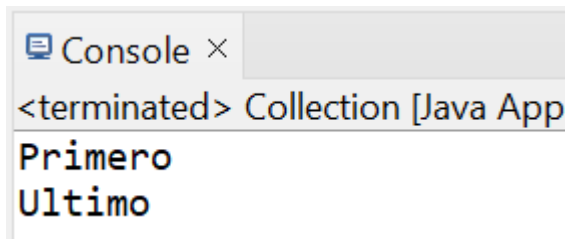
```
Deque<String> arrayDeque = new ArrayDeque<>();

// Se agregan los elementos de los dos lados
arrayDeque.addFirst("Primero");
arrayDeque.addLast("Ultimo");

// Se acceden a los elementos
System.out.println(arrayDeque.getFirst()); // Outputs: Primero
System.out.println(arrayDeque.getLast()); // Outputs: Ultimo

// Removemos elementos
arrayDeque.removeFirst();
arrayDeque.removeLast();
```

ArrayDeque Print



```
<terminated> Collection [Java App
Primero
Ultimo
```

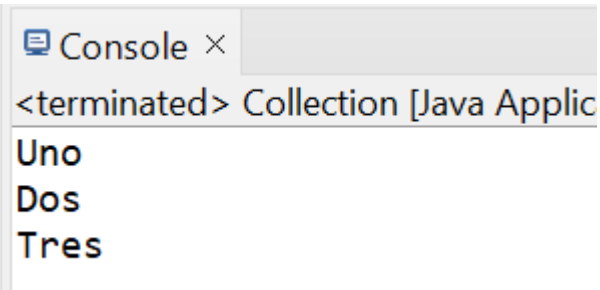
HashSet

```
Set<String> hashSet = new HashSet<>();

// Agregamos elementos
hashSet.add("Uno");
hashSet.add("Dos");
hashSet.add("Tres");
hashSet.add("Uno"); // No sera agregado por ser duplicado

// Iteramos para imprimir
for (String number : hashSet) {
    System.out.println(number);
}
```

HashSet Print



```
Console ×
<terminated> Collection [Java Applic
Uno
Dos
Tres
```

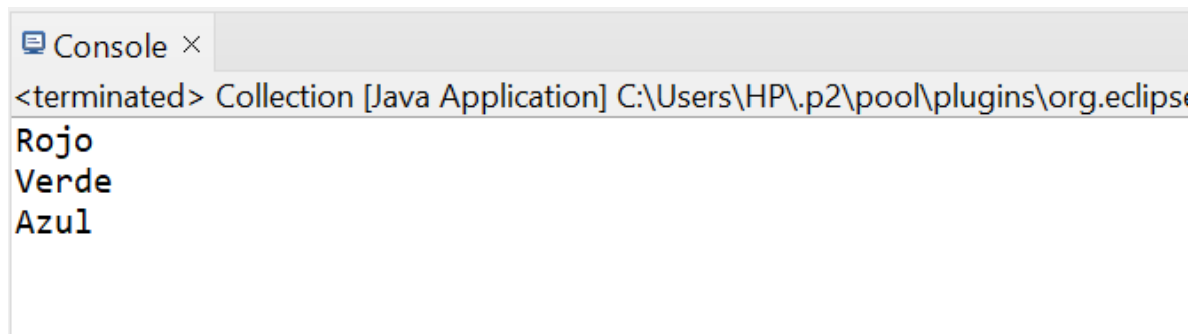
LinkedHashSet

```
Set<String> linkedHashSet = new LinkedHashSet<>();

// Agregamos elementos
linkedHashSet.add("Rojo");
linkedHashSet.add("Verde");
linkedHashSet.add("Azul");

// Itera segun el orden
for (String color : linkedHashSet) {
    System.out.println(color);
}
```

LinkedHashSet Print



```
Console ×
<terminated> Collection [Java Application] C:\Users\HP\.p2\pool\plugins\org.eclips
Rojo
Verde
Azul
```

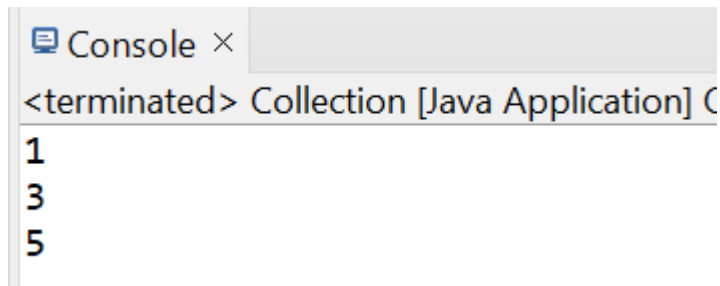
TreeSet

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

// Agregamos los elementos
treeSet.add(5);
treeSet.add(1);
treeSet.add(3);

// Iteramos el treeset (Los ordena según prioridad)
for (Integer number : treeSet) {
    System.out.println(number); // Se espera salidas de 1,3 y 5
}
```

TreeSet Print



```
Console ×  
<terminated> Collection [Java Application] C  
1  
3  
5
```

Conclusion

At the end of this activity we could see the different types of arrays and list that we can do in Java, most of the functions accept arrays and do something different, some sort it some ways like the TreeSet and others execute it by order like the queue interfaces ones.