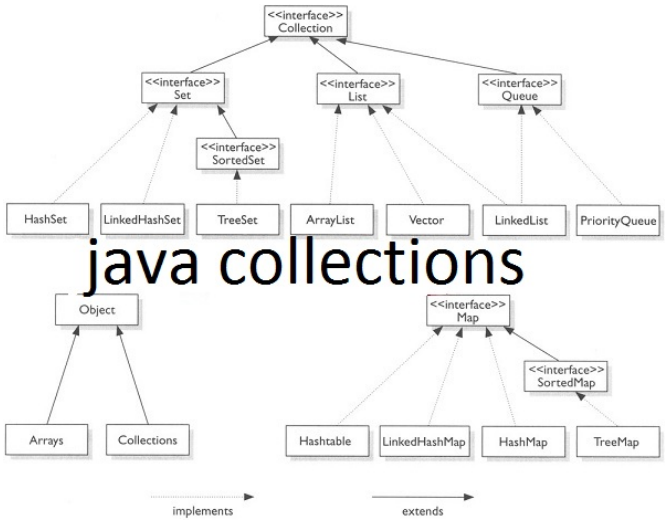
Colecciones

Dr. Mario Marcelo Berón

Índice

Colecciones



Concepción

Es un objeto que agrupa objetos en una unidad.

Usos

Las colecciones son usadas para almacenar, recuperar y comunicar datos.

Colecciones

Infraestructura de Colecciones

Es una arquitectura para representar y manipular colecciones. Toda la infraestructura de colecciones contiene lo siguiente:

Interfaces

Tipos de datos abstractos que representan colecciones. Las interfaces permiten que las colecciones sean manipuladas independientemente de los detalles de su representación.

Colecciones

Infraestructura de Colecciones

Es una arquitectura para representar y manipular colecciones. Toda la infraestructura de colecciones contiene lo siguiente:

```
interface Foo {
  default void someMethod() {
    System.out.println("Foo#someMethod()");
  }

  default void someOtherMethod() {
    System.out.println("Foo#someOtherMethod()");
  }
}

interface Bar {
  default void someMethod() {
    System.out.println("Bar#someMethod()");
  }
}

class FooBar implements Foo, Bar {

  public static void main(String... args) {
    System.out.println("FooBar#main");

    FooBar fooBar = new FooBar();
    fooBar.someMethod();
    fooBar.someOtherMethod();
  }
}
```

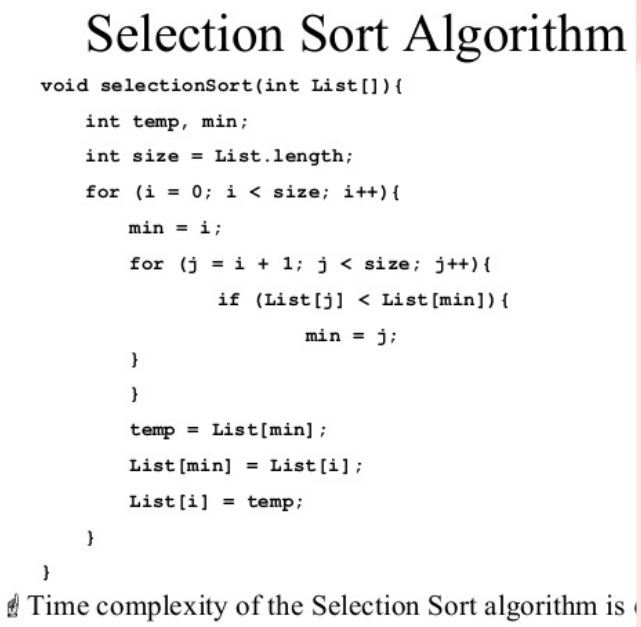
Implementaciones

Esas son implementaciones concretas de las interfaces de colección. En esencia son estructuras de datos reusables.

Colecciones

Infraestructura de Colecciones

Es una arquitectura para representar y manipular colecciones. Todas la infraestructura de colecciones contiene lo siguiente:



Selection Sort Algorithm

```
void selectionSort(int List[]){
    int temp, min;
    int size = List.length;
    for (i = 0; i < size; i++){
        min = i;
        for (j = i + 1; j < size; j++){
            if (List[j] < List[min]){
                min = j;
        }
        }
        temp = List[min];
        List[min] = List[i];
        List[i] = temp;
    }
}
```

Time complexity of the Selection Sort algorithm is

Algoritmos

Son métodos que realizan computaciones útiles, tales como buscar, clasificar, sobre los objetos que implementan la interfaces de colección. Los algoritmos son polimórficos, es decir, el mismo método puede ser usado sobre diferentes implementaciones de la interfaz de colección apropiada. En esencia son una funcionalidad reusable.

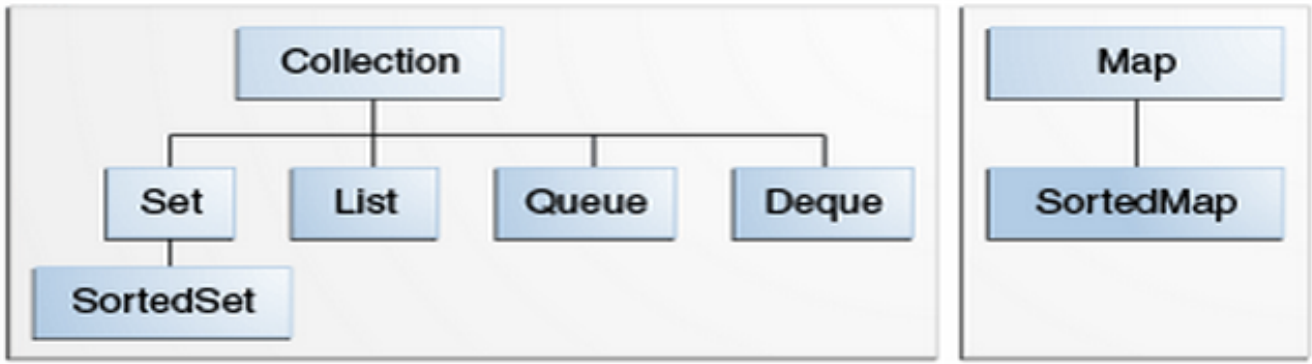
Colecciones

Beneficios de la Infraestructura de Colecciones de Java

1. Reduce el esfuerzo de programación.
2. Incrementa la velocidad y calidad del programa.
3. Permite la interoperabilidad entre APIs poco relacionadas.
4. Reduce el esfuerzo de aprendizaje y como usar nuevas APIs.
5. Etc.

Colecciones

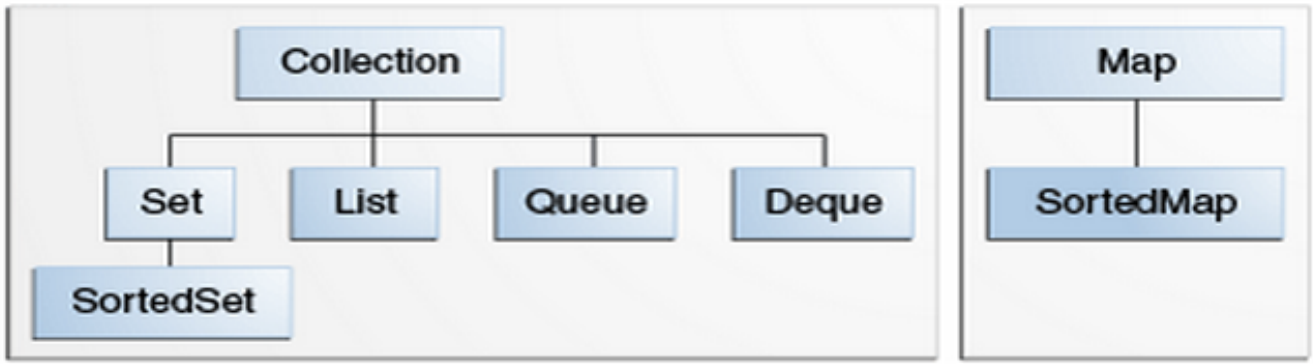
El núcleo de las interfaces de colección encapsulan diferentes tipos de colecciones. Esta peculiaridad permite que las colecciones sean manipuladas independientemente de los detalles de su representación.



Importante

Todas las interfaces de colección son genéricas, por consiguiente cuando se usan las colecciones se deben especificar el tipo de dato del objeto contenido en la colección.

Collection



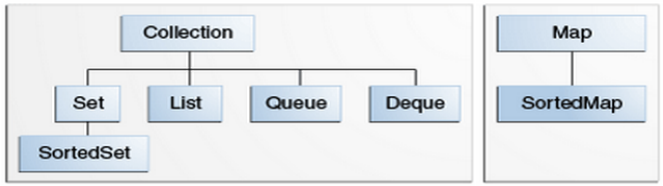
Concepto

Representa un grupo de objetos conocidos como sus elementos.

Uso

Se usa cuando se desea el máximo nivel de generalidad.

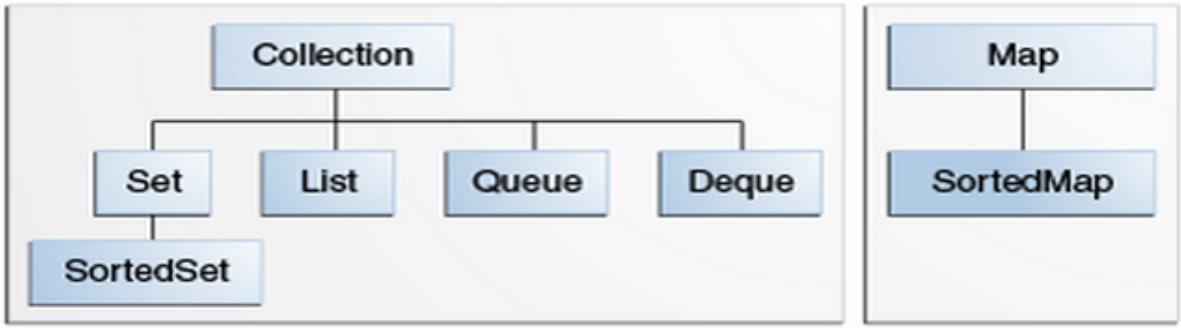
Collection



Métodos-Operaciones Básicas

- int size(): retorna el tamaño de la colección.
- boolean isEmpty(): verdadero si la colección está vacía falso en otro caso.
- boolean contains(Object element): verdadero si la colección contiene a element. Falso en otro caso.
- boolean contains(Object element): verdadero si la colección contiene element. Falso en otro caso.
- etc.

Collection



Métodos que trabajan sobre colecciones completas

- boolean containsAll
- boolean addAll
- boolean removeAll
- boolean retainAll
- void clear

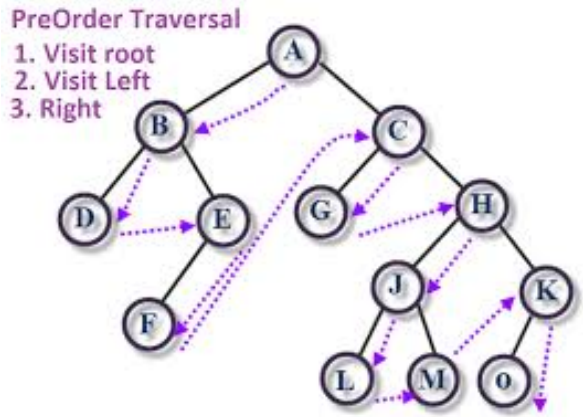
Collection

Recorrido

Se pueden recorrer de tres formas:

1. Construcción for-each
2. Iteradores

Collection



Recorrido: Construcción for-each

La construcción for-each permite concisamente recorrer una colección o arreglo usando un loop for.

```
for (Object o : collection)
    System.out.println(o);
```

Collection

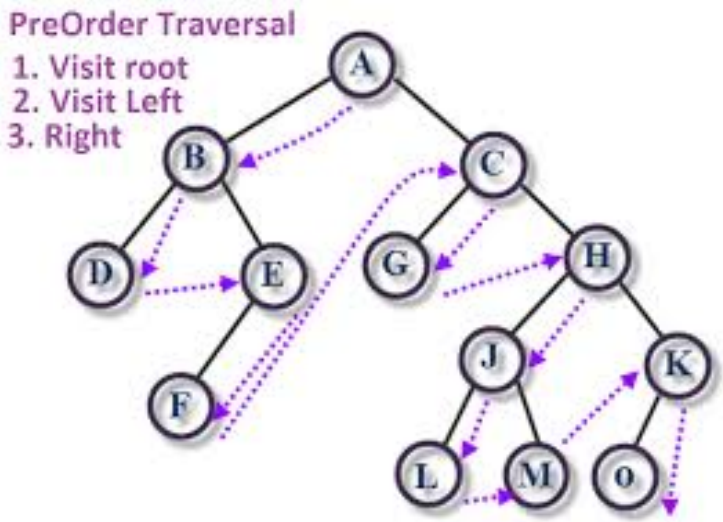
Recorrido: Iteradores

1. Permite recorrer una colección y eliminar elementos si es que es necesario.
2. La interface iterador tiene los siguientes métodos:
   1. boolean hasNext(): retorna verdadero si hay más elementos para procesar.
   2. E next(): retorna el próximo elemento a procesar.
   3. void remove(): es opcional y permite eliminar un elemento de la colección.

Importante

Iterator.remove() es la única forma segura de eliminar un elemento de una colección. El comportamiento no está especificado si la colección subyacente se modifica de cualquier otra manera mientras la iteración está en progreso.

Collection



Recorrido: Iteradores

Un iterador se usa cuando:

1. Se elemina un elemento de la colección. El for-each oculta el iterador y por lo tanto no se pueden eliminar elementos.
2. Itere sobre múltiples colecciones.

Collection

Ejemplos

```
public static void main(String[] args) {
 Collection co1;
  co1= new ArrayList();
  co1.add(1);
  co1.add(3.4);
  co1.add("String");
  System.out.println("Tamaño:"+co1.size());
  System.out.println("Colección 1:"+co1);
  ....
}
```

Collection

Recorrido

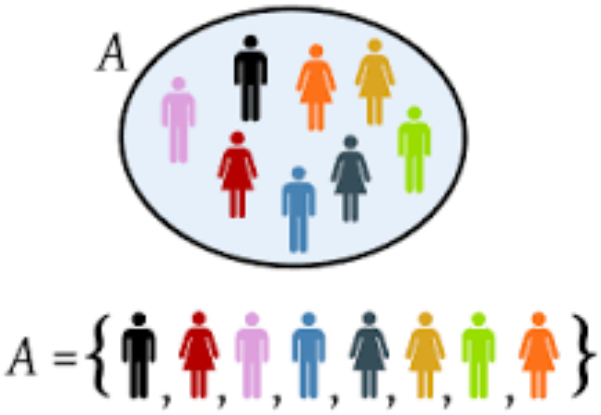
```
public static void main(String[] args) {
 Collection co1;
  co1= new ArrayList();
  co1.add(1);
  co1.add(3.4);
  co1.add("String");
  for(Object o: co1){
   System.out.println("Objeto:"+o);
  }
  ....
}
```

Collection

Iteradores

```
public static void main(String[] args) {
 Collection co1;
 Iterator i;
  co1= new ArrayList();
  co1.add(1);
  co1.add(3.4);
  co1.add("String");
  i=co2.iterator();
  while (i.hasNext()){
   System.out.println("Colección 2:"+i.next());
  };
  ....
}
```

Colecciones: Set



Concepción

Es una colección que no puede tener elementos duplicados. Esta interface modela un conjunto en el sentido matemático. La interface Set contiene solo los métodos heredados de Collection y agrega la restricción de que no pueden haber elementos duplicados.

Set

Operaciones de la Interface Set

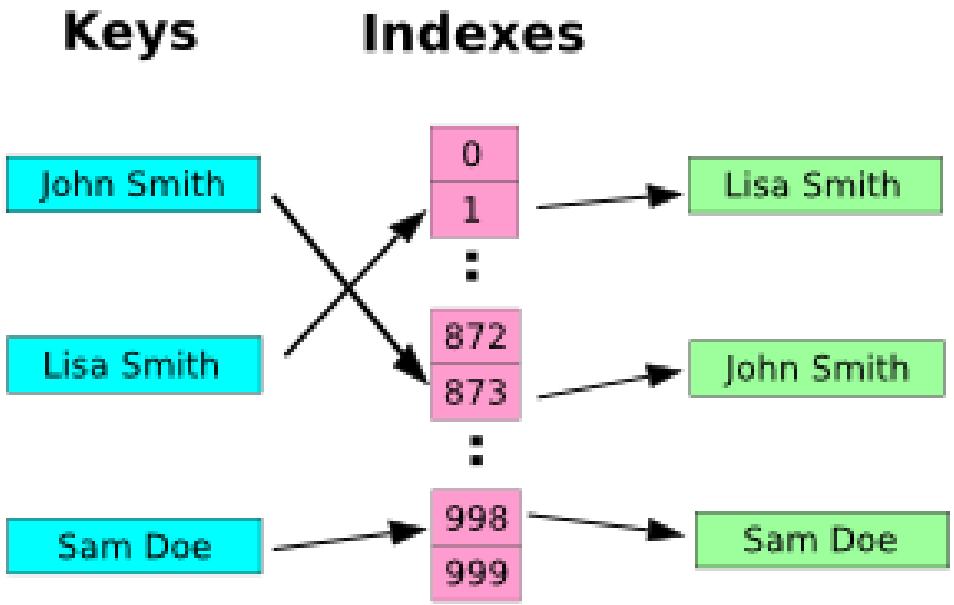
1. size: retorna el número de elementos en el conjunto.
2. add: incorpora el elemento especificado en el conjunto si no está presente. Esta operación retorna true si el elemento fue incorporado y false en otro caso.
3. remove: elimina el elemento especificado del conjunto. Esta operación retorna true si el elemento se eliminó y false en otro caso.

Set

Operaciones de la Interface Set

1. s1.containsAll(s2): retorna como resultado true si s2 es un sub-conjunto de s1.
2. s1.addAll(s2): transforma s1 en la unión de s1 y s2.
3. s1.retainsAll(s2): transforma s1 en la intersección de s1 y s2.
4. s1.removeAll(s2): transforma s1 en diferencia s1 y s2.

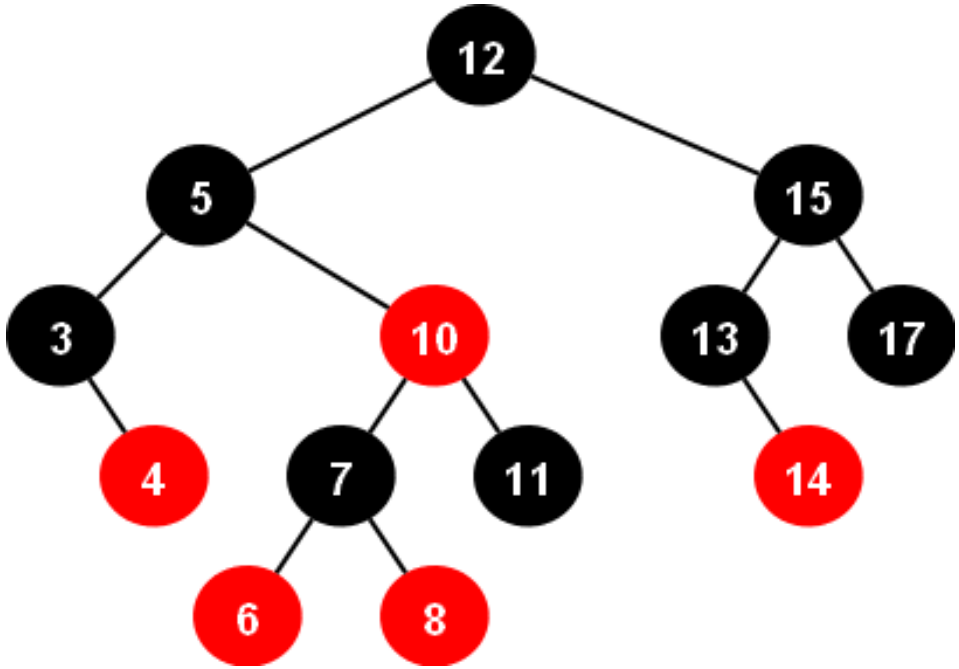
Implementeaciones de Conjuntos



HashSet

Almacena los elementos en una tabla de hash es implementación tiene un muy buen desempeño. No garantiza el orden en las iteraciones.

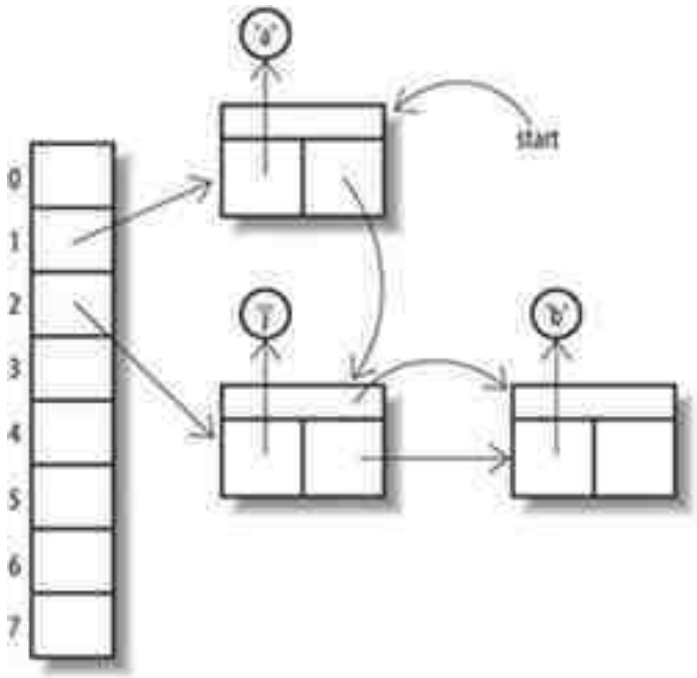
Implementaciones de Conjuntos



TreeSet

Almacena los elementos en árbol Rojo-Negro. Dicha estructura de datos ordena los elementos en base a sus valores. Es sustancialmente más lento que HashSet.

Implementaciones de Conjuntos



LinkedHashSet

Es implementada con una tabla de hash con una lista vinculada. El orden de sus elementos está dado por el orden en el cual fueron insertados.

Set

Ejemplo

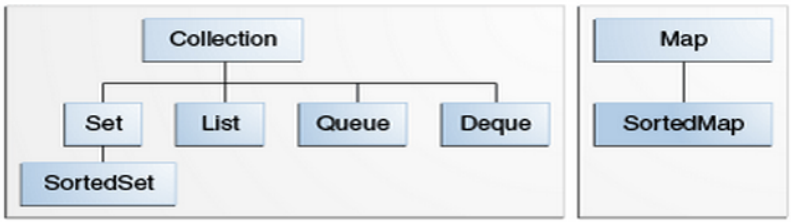
```
public static void main(String[] args) {
Set <Integer> c = new HashSet<Integer>();
c.add(2);
c.add(3);
System.out.println("Conjunto:"+c+
                   "Cardinalidad:"+c.size());
}
```

Set

Iteradores

```
Set c= new HashSet();
 c.add(1);
 c.add("Hola");
 System.out.println("Conjunto:"+c);
 i=c.iterator();
 while (i.hasNext()){
  System.out.println("Elemento: "+i.next());
 }
```

SortedSet



Concepción

Un SortedSet es un conjunto que mantiene sus elementos ordenados en orden ascendente de acuerdo al orden natural de los elementos o bien de acuerdo a un comparador provisto por en tiempo de creación de SortedSet.

SortedSet

public interface **SortedSet<E>**
extends Set<E>

```
public interface SortedSet<E> extends Set<E> {
    // Range-view
    SortedSet<E> subSet(E fromElement, E toElement);
    SortedSet<E> headSet(E toElement);
    SortedSet<E> tailSet(E fromElement);

    // Endpoints
    E first();
    E last();

    // Comparator access
    Comparator<? super E> comparator();
}
```

Operaciones

1. Range View: permite operaciones de rango arbitrarias sobre el conjunto ordenado.
2. Endpoints: retorna el primer o el último elemento en el conjunto ordenado.
3. Acceso de Comparador: retorna el comparador usado para ordenar el conjunto.

SortedSet

Operaciones de Set

Las operaciones de SortedSet heredadas desde Set se comportan de la misma forma en conjuntos ordenados con dos excepciones:

- El iterador recorre el conjunto de forma ordenada.
- El arreglo retornado por toArray contiene los elementos ordenados.

SortedSet

Operaciones Range-View

1. subSet: toma dos puntos finales (objetos) que deben poder ser comparados con los elementos en el conjunto ordenado usando el comparador de conjunto o el orden natural de sus elementos. El resultado de esta operación es un SortedSet que incluye los elementos que entre el límite inferior y el límite superior.
2. headSet: retorna un SortedSet que contiene los elementos del receptor desde el inicio, pero no incluye, el elemento especificado (parámetro).
3. tailSet: retorna un SortedSet que contiene los elementos del receptor que son mayores o iguales que el objeto recibido como parámetro.

SortedSet

Ejemplos

```
SortedSet <Integer> ss= new TreeSet<Integer>();
 ss.add(1);
 ss.add(-1);
 ss.add(-44);
 ss.add(0);
 System.out.println("SortedSet:"+ss);
 System.out.println("subSet:"+ss.subSet(-10, 1));
 System.out.println("headSet:"+ss.headSet(0));
 System.out.println("tailSet:"+ss.tailSet(0));
```

SortedSet

Operaciones Endpoints

La interface SortedSet contiene operaciones que permiten retornar el primer y último elemento en el conjunto ordenado llamadas *first* y *last*.

Comparador

La interface SortedSet contiene un método llamado comparator que retorna el comparador usado para ordenar el conjunto o null si el conjunto está ordenado de acuerdo al orden natural de sus elementos.

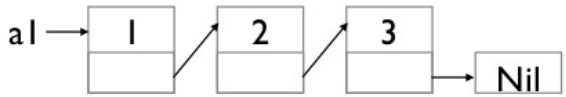
SortedSet

Ejemplos

```
SortedSet <Integer> ss= new TreeSet<Integer>();
ss.add(1);
ss.add(-1);
ss.add(-44);
ss.add(0);
System.out.println("Primero:"+ss.first());
System.out.println("Ultimo:"+ss.last());
```

List

Lists




```
List<Integer> nil = Lists.nil();

List<Integer> a3 = nil.prepend(3);
List<Integer> a2 = a3.prepend(2);
List<Integer> a1 = a2.prepend(1);
```

@ignasi35

Concepción

Una List es una colección ordenada la cual puede contener elementos duplicados.

List

Operaciones Específicas

1. Acceso Posicional: manipula los elementos basados en su posición en la lista. Esto incluye métodos tales como: *get*, *set*, *add*, *addAll* y *remove*.
2. Búsqueda: busca por un objeto especificado en la lista y retorna su posición. Esto incluye métodos tales como: *indexOf* y *lastIndexOf*.
3. Iteración: extiende la semántica de Iterator tomando ventajas de la naturaleza secuencial de la lista. El método *listIterator* provee este comportamiento.
4. Range-view: el método *sublist* realiza operaciones de rango arbitrarias sobre la lista.

List

Operaciones de Collection

Las operaciones heredadas de Collection hacen lo que se espera deben hacer.

1. La operación *remove* elimina de la lista la primera ocurrencia del elemento especificado.
2. Las operaciones *add* y *addAll* incorporan nuevos elementos al final de la lista.
3. etc.

List

Iteradores

1. Como se espera un iterador retorna los elementos de la lista en secuencia. List también provee un iterador, llamado ListIterator, que permite recorrer la lista en otra dirección, modificar la lista durante la iteración, y obtener la posición corriente del iterador.
2. Los tres métodos que ListIterator hereda de Iterador (hasNext, next y remove) hacen exactamente lo mismo en ambas interfaces.
3. El método *hasPrevious* y *previous* son los análogos a hasNext y next.

List

Iteradores

```
for (ListIterator<Type> it =
      list.listIterator(list.size());
      it.hasPrevious(); ) {
    Type t = it.previous();
    ...
}
```

Notas y Comentarios

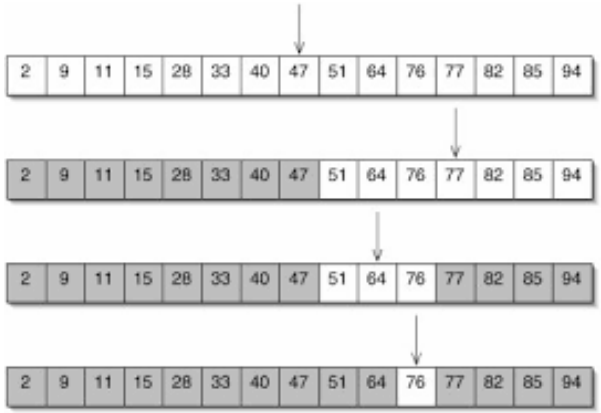
ListIterator tiene dos formas. Cuando se utiliza sin argumentos, en este caso se posiciona en el comienzo de la lista. Cuando se utiliza con argumento, el iterador se posiciona en un índice especificado de la lista.

List

Algoritmos de Listas

- sort: ordena una lista.
- shuffle: permuta aleatoreamente los elementos en una lista.
- reverse: inverte el orden de los elementos de una lista.
- rotate: rota los elementos de una lista.
- swap: intercambia los elementos en posiciones especificadas en una lista.
- replaceAll: reemplaza todas las ocurrencias de un valor en una lista por un valor especificado.
- fill: sobreescribe todos los elementos en una lista con un valor especificado.
- copy: copia la lista fuente en la lista destino.

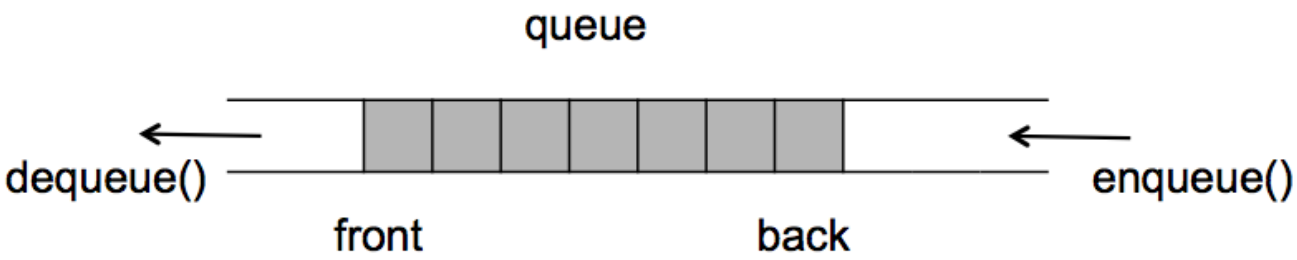
List



Algoritmos de Listas

- binarySearch: busca un elemento en una lista ordenada usando el algoritmo de búsqueda binaria.
- indexOfSubList: retorna el índice de la primer sublista de una lista.
- lastIndexSubList: retorna el índice de la última sublista de una lista.

Queue



Concepción

Es una colección para mantener elementos para un posterior procesamiento. Además de las operaciones heredadas de Collection, las filas proveen las operaciones de *inserción*, *eliminación*, e *inspección*.

Queue

Operaciones

Los métodos de Fila reaccionan de dos formas:

1. Disparan una excepción si la operación fracasa.
2. Retornan un valor especial si la operación fracasa.

Queue

Estructura de la Interface Queue

Tipo de Operación	Dispara una Excepción	Retorna Valor Especial
Insertar	add(e)	offer(e)
Eliminar	remove()	poll()
Examinar	element()	peek()

Notas y Comentarios

Las filas comúnmente, pero no necesariamente, ordenan los elementos de una forma FIFO. Entre las excepciones están las colas de prioridad que ordenan sus elementos de acuerdo a sus valores.

Queue

Ejemplo

```
Queue <Integer>q = new LinkedList<Integer>();
q.add(1);
q.add(-10);
q.add(29);
System.out.println("Queue:"+q);
System.out.println("Elimina de Queue:"+q.remove());
System.out.println("Queue:"+q);
```

Dequeue

Concepción

Es una cola de doble fin. Es una colección lineal que soporta la inserción y eliminación de elementos en ambos extremos.

Implementaciones

Las clases *ArrayDequeue* y *LinkedList* implementan la interface Dequeue.

Dequeue

Notas y Comentarios

La interface Dequeue puede ser usada para implementar pilas y filas.

Dequeue

Métodos

Operación	Primer Elemento	Último Elemento
Insertar	addFirst(e); offerFirst(e)	addLast(e); offerLast(e)
Eliminar	removeFirst(); pollFirst()	removeLast(); pollLast()
Examinar	getFirst(); peekFirst()	getLast(); peekLast()

Notas y Comentarios

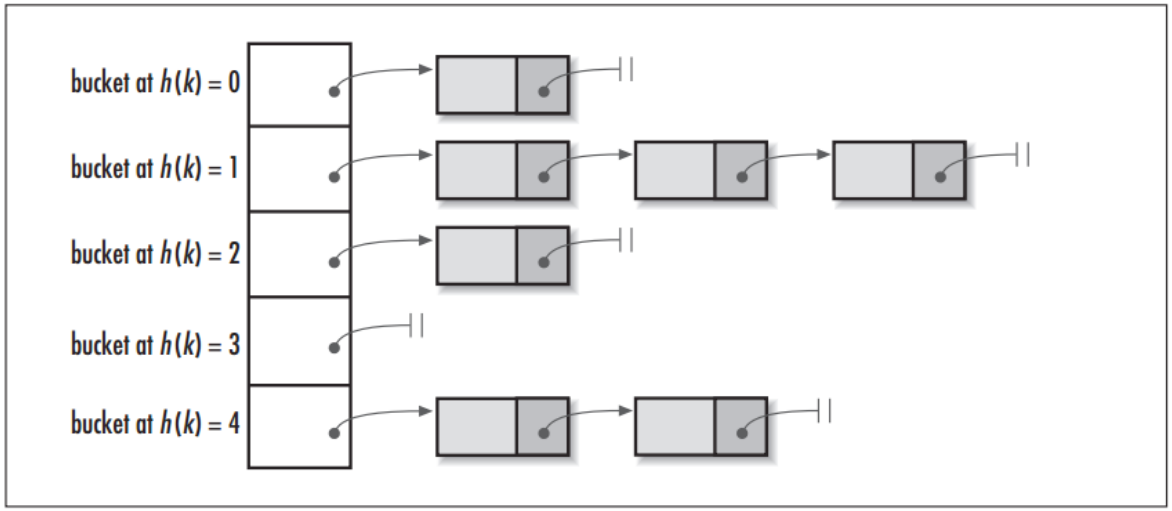
Las métodos de Dequeue siguen la misma política que los métodos de Queue en lo que respecta a los valores retornados.

Dequeue

Ejemplo

Elaborar ejemplos que usen esta colección

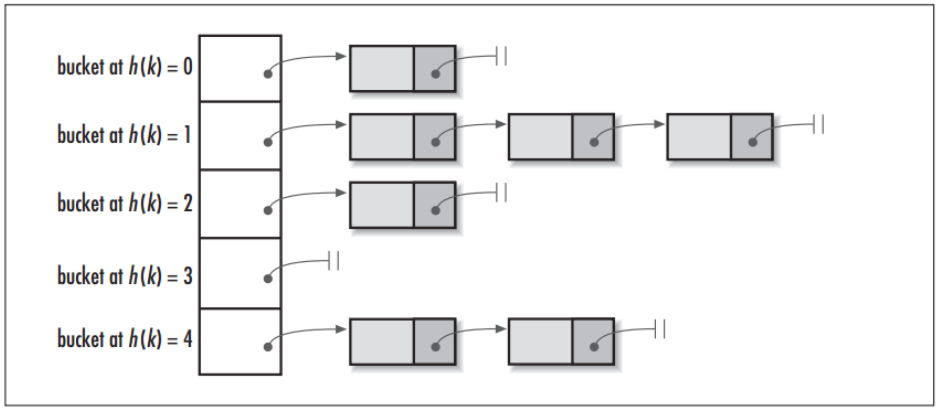
Colecciones: Maps



Concepción

Un mapa es un objeto que mapea claves a valores.

Maps



Características

- Un mapa no puede contener claves duplicadas: Cada clave mapea a lo sumo un valor.
- Modela la abstracción de una función matemática.
- Algunas clases que implementan la interfaz: HashMap, TreeMap, LinkedHashMap, etc.

Colecciones: Maps

Operaciones Básicas

put, *get*, *remove*, *containsKey*, *containsValue*, *size* y *empty*.

Operaciones Sobre Todos los Elementos

putAll y *clear*.

Vistas de Colección

keySet, *entrySet*, y *values*.

Maps

Ejemplo

```
Map m= new HashMap();
 m.put(1,"Mario");
 m.put("Juan", 18);
  System.out.println("Map:"+m);
```

La Interface SortedMap

Concepción

Es un mapa que mantiene sus entradas en orden ascendente, de acuerdo al orden natural de las claves, o de acuerdo a un comparador provisto en tiempo de creación del SortedMap.

La Interface SortedMap

Operaciones

La interface SortedMap provee operaciones mapas comunes y además:

- Range view: realiza operaciones de rango arbitrarias sobre el SortedMap.
- Endpoints: retorna la primer o última clave en el SortedMap.
- Comparator Access: retorna el comparador si existe usado en el SortedMap.

La Interface SortedMap

Operaciones de Maps

Las operaciones heredadas desde Map se comportan de la misma forma en los mapas ordenados con dos excepciones:

- El iterador recorre la colección en orden.
- La operación toArray contiene las claves, valores o entradas en orden.

SortedMap

Ejemplo

Elabore un ejemplo para esta colección

Implementaciones de Propósito General

General-purpose Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap