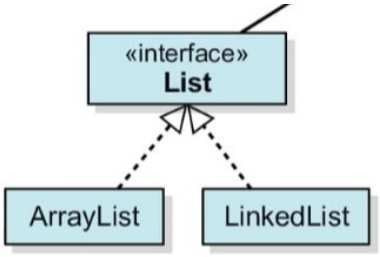
import java.util.*; // importar interfaces y clases del marco de trabajo Collection

Listas

// T tipo genérico o parametrizados (permite la comprobación en tiempo de compilación del tipo de datos pasados a diferencia del tipo Object que no la realiza), E para elementos de colecciones, K para claves, V para valores, N para números y ? puede ser cualquiera.

List<E> lista = new ArrayList<>();

List<Cliente> listaClientes = new ArrayList<>();



- ArrayList: más rápida en lectura y modificación.
- LinkedList: más rápida en inserción y eliminación.

Permiten elementos duplicados y acceso mediante índice

- Interface **Comparable** implementar método **compareTo()** en la clase, para definir criterio de ordenación por defecto.
- Interface **Comparator** implementar método **compare()**, para definir criterios de ordenación alternativos.

Métodos básicos de la interfaz Collection

(afecta a elementos individuales)

Collection<Cliente> coleccionClie = listaClientes;

1. Método de inserción
boolean add(E elem) //si insertara al final por ser lista
2. Métodos de eliminación
boolean remove(Object ob) //si esta repetido elimina el primero
void clear() //elimina todos los elementos

3. Métodos de comprobación
int size() //numeros de elementos de la colección
boolean isEmty() // true si esta vacia
boolean contains(Object ob) // true si objeto pertenece a la colección

4. Otros métodos
string toString() // devuelve cadena que representa la colección

Iterator<E> iterator()
Iterator<Cliente> it = coleccionClie.iterator();
boolean hasnext() // true si quedan elemento por visitar
E next() //avanza al siguiente elemento y nos lo devuelve

void remove() // elimina el último elemento devuelto por next()

Métodos globales de la interfaz Collection

(afecta a grupos de elementos)

boolean containsAll(Collection<?> c) // true si todos los elemento de c están en la colección

boolean addAll(Collection<? extend E> c) // añade a la colección todos los elementos de c

boolean removeAll(Collection<?> c) // elimina de la colección todos los elementos de c

boolean retainAll(Collection<?> c) // elimina de la colección todos los elementos excepto los que esten en c

Métodos de tabla de la interfaz Collection

Object[] toArray() // devuelve una tabla de tipo Object con los mismos elementos de la colección

<T>T[] toArray(T[] t) // devuelve una tabla de tipo genérico T
Cliente[] t2 = otrosClientes.toArray(new Cliente[0])

static <T> List<T> asList(Tabla) // recibe una tabla y devuelve lista inmutable
lista.addAll(Arrays.asList(tabla));

Métodos específicos de la interfaz List

List<Integer> listaEnteros = new ArrayList<>();

E get(int indice) // devuelve el elemento que ocupa el lugar indice
Integer a = listaEnteros.get(2);

E set(int indice, E elem) // guarda el elemento elem en la posición de indice
Integer y = listaEnteros.set(3,10);

void add(int indice, E elem) // inserta el valor elem en la posición indice y desplaza.
listaEnteros.add(3,5);

boolean addAll(int indice, Collection<? extends E> c) // inserta lista en la posición indice y desplaza.
listaEnteros.addAll(3, otrosEnteros);

E remove(int indice) // elimina elemento del indice y lo devuelve
// no confundir con "boolean remove(Integer valor)" elimina el valor y devuelve true

int indexOf(Object ob) // devuelve el índice de la primero ocurrencia de ob o -1 si no esta

int lastIndexOf(Object ob) // devuelve el índice de la última ocurrencia de ob o -1 si no esta

boolean equals(Object otraLista) // compara dos lista si son exactamente iguales.

void sort(Comparator <? super E> c) // ordena la lista invocante con el criterio de c

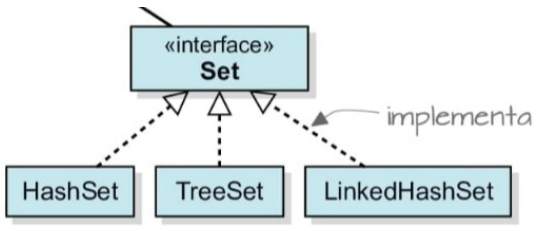
Interfaz Set

Elimina repeticiones y no permite índice

TreeSeet<Cliente> conjuntoClientes = new TreeSet<>();

- HashSet; no garantiza ningún orden de inserción pero mayor rendimiento
- TreeSet: garantiza el orden basado en el valor de los elementos insertados. El criterio de ordenación por defecto es el natural (el proporcionado por el método compareTo() de la clase generica E) o bien se especifica por medio de un comparador pasado como parámetro al constructor.
  Ejemplo de un comparador pasado como parámetro al constructor:

```
Comparator<Cliente> comparaNombres=new Comparator<>() {
                    @Override
                    public int compare( Cliente o1, Cliente o2) {
                       return o1.nombre.compareTo(o2.nombre);
                    }
                };
Set<Socio> sociosPorNombre =new TreeSet<>(comparaNombres);
```



- LinkedHasSet: garantiza el orden basado en la inserción (al final)

Métodos de la clase Collections

1. Métodos de ordenación

static <T extends Comparable<? Super T>> void sort(List<T> lista)
Collections.sort(lista);

static <T extends Comparable<? Super T>> void sort(List<T> lista, Comparator <? super E> c)
Collections.sort(lista, new ComparaNombres());

```
class ComparaNombres implements Comparator<Cliente>{
        public int compare(Cliente o1, Cliente o2){
                return o1.nombre.compareTo(o2.nombre);}
}
```

2. Métodos de búsqueda

static <T> int binarySearch(List<? Extends Comparable<? Super T>> list, T clave)
int indice = Collections.binarySearch(lista, new Cliente("112",null,null));

static <T> int binarySearch(List<? Extends Comparable<? Super T>> list, T clave, Comparator <? super E> c))
int indice = Collections.binarySearch(lista, new Cliente(null,"Carlos",null), new ComparaNombres()));

3.Métodos para la manipulación

static void swap(List<?> lista, int i, int j) // intercambia los valores de los indices
Collections.swap(datos,0,3);

static <T> booleaan replaceAll(List<T> lista, T antiguo, T nuevo) // reemplaza el elemento antiguo por el nuevo, en todos los lugares
Collections.replaceAll (datos,4,100);

static <T> void fill(List<? super T> lista, T valorRelleno) // rellena todos los elementos de la lista
Collections.fill(datos,7)

static <T> void copy(List<? super T> destino, List<? extends T> origen) // copia la lista origen en la lista destino

4. Otras utilidades

static void shufle(List<?> lista) // desordena la lista
Collections.shufle(datos);

static int frequency(Collection<?> col, Object ob) // número de veces que aparece objeto en la colección
Collections.frequency(datos,7);

static <T extends Comparable<? super T>> T max(Collection<? extends T> col) // busca y delvuelve el valor máximo de una colección
Integer maximo =Collections.max(datos);

static <T>> T max(Collection<? extends T> col, Comparator <? super T> comp) // devuelve el máximo utilizando comp como criterio de comparación
Cliente ultimo = Collections.max(conjuntoClie, new ComparaNombres());

```
Comparator<Cliente> c = new Comparator<>()  {
        @Override
        public int compare(Cliente o1, Cliente o2){
                return o1.nombre.compareTo(o2.nombre);}
}
```
Cliente ultimo = Collections.max(conjuntoClie, c);
Hay métodos análogos para el mínimo.

Integer mínimo =Collections.min(datos);
Cliente primero = Collections.min(conjuntoClie, new ComparaNombres());

static void reverse(List<?> lista) // invierte la lista colocando en orden inverso.

static <T> Set<T> singleton(T elem) // devuelve un conjunto con elem como único elemento.
datos.removeAll(Collections.singleton(7));

Métodos de la interfaz Map

HashMap; no garantiza ningún orden de inserción
TreeMap: garantiza el orden basado en el valor de la clave insertados
LinkedHashMap: garantiza el orden basado en la inserción (al final)

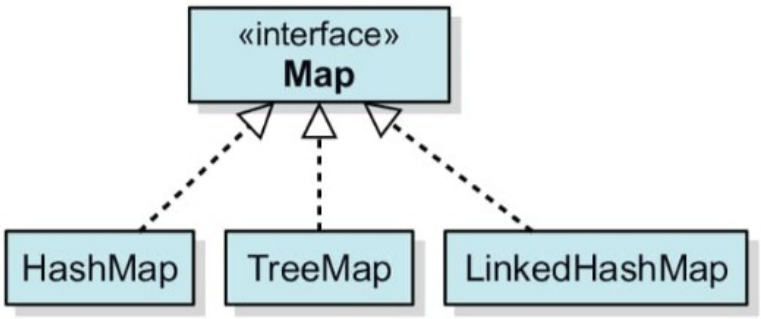
Map<K,V> m = new HashMap

1. Método de inserción
V put(K,V) // si ya esta la clave devuelve el valor antiguo
m.put("Ana",65)

2. Métodos de eliminación
V remove(K) // devuelve el valor de la clave eliminada

void clear() //elimina todas las entradas.

3. Método de busqueda
V get(Object k)
int a = m.get("Ana");



4. Métodos de comparación
boolean containsKey(Object k) // devuelve true si hay una entrada con k
m.containsKey ("Ana");

boolean containsValue(Object v) // devuelve true si hay un valor con v
m.containsValue("Ana");

Vistas Collection de los mapas

Set<K> keySet() // devuelve una vista, con estructura Set de las claves de un mapa
Set<String> claves = m,KeySet();

Collectios<V> values() // devuelve una vista Collection de los valores
Collection<Double> estaturas = m.values();

Set<Map.Entry<K,V>> entrySet() // devuelve una vista conjunto de las entradas, objetos del tipo Map
Set<Map.Entry<String, Double>> entradas = m. entrySet();

K getkey() // devuelve la clave de la entrada
V getValue() // devuelve el valor de la entrada
V setValue(V nuevoValor) // asigna nuevoValor a la entrada y devuelve el valor antiguo