



PROGRAMACIÓN ORIENTADA A OBJETOS:

TEMA 7. PROGRAMACIÓN GENÉRICA

Presenta: Mtro. David Martínez Torres

Universidad Tecnológica de la Mixteca

Instituto de Computación

Oficina No. 37

dtorres@gs.utm.mx



Contenido

1. Concepto de programación genérica.
2. Implementación de programación genérica.
3. Uso de bibliotecas estándar de programación genérica.

1. Concepto de programación genérica

La programación genérica es un estilo de programación enfocado más en los algoritmos que en los datos.

El objetivo es independizar el proceso del tipo de datos sobre los que se aplica.

Ejemplo, cualquier método de ordenamiento es el mismo, y se aplica a distintos tipos de datos.



1. Concepto de programación genérica

Las listas, colas o árboles son estructuras de datos que pueden trabajar con cualquier tipo de datos

Se pueden tener listas de Persona, de String, etc. La funcionalidad que la lista ofrece es independiente del tipo de datos que maneja

En POO, este comportamiento se puede lograr utilizando genéricos.

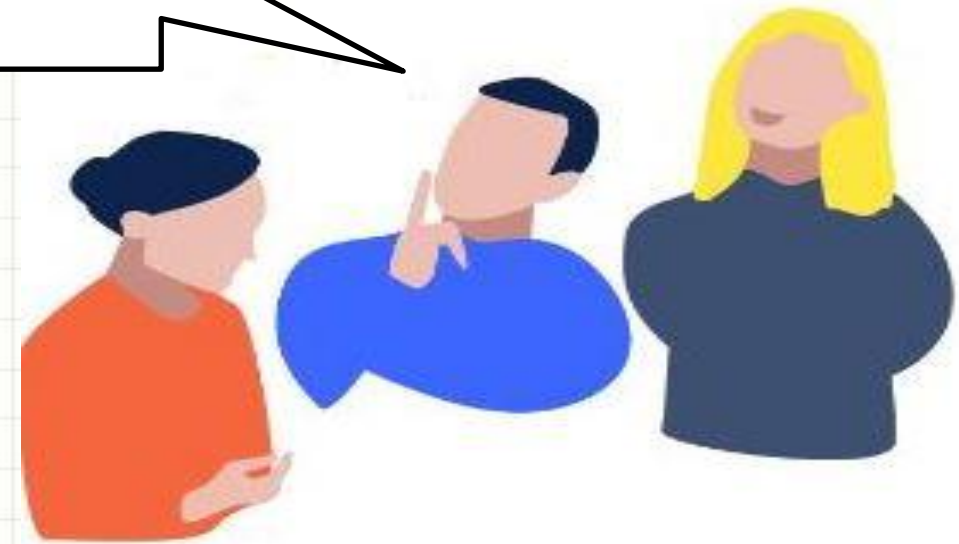


1. Concepto de programación genérica

Una clase genérica es una estructura que normalmente almacena y gestiona objetos, por ejemplo puede ser una bolsa, una lista ,un árbol, un diccionario etc.

Cualquier estructura servirá en java para un tipo de objeto concreto. Por cada tipo de objeto se necesitaría una clase nueva.

Con la construcción de un **genérico** se necesitaría una única clase para todos los tipos de objetos



Aproximación de programación genérica

- Cualquier objeto en Java es derivado de la clase *Object*, por tanto, es posible describir algoritmos en función de esta clase.

```
package cursoPOO.persistencia;

public class CeldaMemoria {
    private Object valor;

    public Object getValor() {
        return valor;
    }

    public void setValor(Object valor) {
        this.valor = valor;
    }
}
```

Aproximación de programación genérica

```
package cursoPOO.presentacion;

import cursoPOO.persistencia.CeldaMemoria;

public class PruebaCeldaMemoria {
    public static void main(String []args){
        CeldaMemoria m = new CeldaMemoria();

        m.setValor(new Integer(10));
        System.out.println("El contenido es: ");
        System.out.println(((Integer) m.getValor()).intValue());

        m.setValor("Ahora es una cadena");
        System.out.println("El contenido es: ");
        System.out.println((String) m.getValor());

    }
}
```

Aproximación de programación genérica

- Algunos problemas de la programación genérica basada en Object:
 - Para recuperar la información, el usuario debe conocer el tipo de dato almacenado.
 - No hay forma de especificar que se va a crear un contenedor de determinado tipo.

2. Implementación de programación genérica: método genérico

Implementación de **clase**
con **métodos genéricos**

```
package cursoPOO.persistencia;

public class ClaseConMetodoGenerico {
    //genéricos en métodos
    public static <T> void imprimir(T dato){
        System.out.println(("El dato es: "+dato.toString() +
            " , de tipo " + dato.getClass().getName()));
    }

    public static <T,U> boolean tiposIguales(T primero, U segundo){
        return primero.getClass().equals(segundo.getClass());
    }
}
```

Los genéricos trabajan solo con **objetos** no con tipos primitivos.

```
package cursoP00.presentacion;
```

```
import cursoP00.persistencia.ClaseConMetodoGenerico;
```

```
public class PruebaGenericosEnMetodos {
```

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

```
        System.out.println("Llamada de método genérico");
```

```
        ClaseConMetodoGenerico.imprimir(1);
```

```
        ClaseConMetodoGenerico.imprimir("hola");
```

```
        System.out.println("Comparación de datos");
```

```
        System.out.println(ClaseConMetodoGenerico.tiposIguales("hola",  
            "bienvenido"));
```

```
        System.out.println(ClaseConMetodoGenerico.tiposIguales("hola", 1));
```

```
    }
```

```
}
```

2. Implementación de programación genérica: método genérico

Console Tasks

PruebaGenericosEnMetodos (7) [Java Application] C:\Program File

Llamada de método genérico

El dato es: 1 , de tipo java.lang.Integer

El dato es: hola , de tipo java.lang.String

Comparación de datos

true

false

2. Implementación de clase con parámetro genérico

```
package cursoPOO.persistencia;

public class CeldaMemoria<E> {
    private E valor;

    //constructor con argumentos
    public CeldaMemoria(E valor) {
        this.valor = valor;
    }

    //constructor vacío.
    public CeldaMemoria() {
    }

    public E getValor() {
        return valor;
    }

    public void setValor(E valor) {
        this.valor = valor;
    }
}
```

Parámetro genérico

Nombre de la clase genérica

```
package cursoPOO.presentacion;

import cursoPOO.persistencia.CeldaMemoria;

public class PruebaCeldaMemoria {

    public static void main(String[] args) {
        CeldaMemoria<Integer> cmi = new CeldaMemoria<Integer>(1);
        CeldaMemoria<Float> cmf= new CeldaMemoria<Float>(2.1f);
        CeldaMemoria<String> cms= new CeldaMemoria<String>();
        int i;

        System.out.println("valor de cmi: " + cmi.getValor());
        System.out.println("valor de cmf: " + cmf.getValor());

        cmi.setValor(new Integer(4));
        i = cmi.getValor();
        System.out.println("valor obtenido de cmi: " + i);

        //si se descomenta la siguiente linea,
        //se produce un error de compilación debido que se
        //especificó como tipo concreto Integer.
        //cmi.setValor("Esto es una cadena");
        //solución declarar un objeto para tipo concreto
        //String
        cms.setValor("Esto es una cadena");
        System.out.println("valor de cms: "+ cms.getValor());
    }
}
```

2. Implementación de programación genérica

- Así mismo, una clase genérica puede tener más de un parámetro de tipo genérico.

```
public class Nombre<E, T, ...>{  
    private E variableInstancia1;  
    private T variableInstancia2;  
    ...  
}
```

2. Implementación de programación genérica

- Tener en cuenta que los genéricos de java solo funcionan con objetos
 - Lo siguiente sería un error:
`ClaseGenerica<int> obj = new ClaseGenerica<int>(53);`
- Convenciones para nombrar a los genéricos:
 - E: Element (usado frecuentemente por Java Collections Framework)
 - K: key (llave, usada en mapas)
 - N: Number (para números)
 - T: Type (Representa un tipo, es decir, una clase)
 - V: Value (Representa el valor, usado en mapas).
 - S, U, V, etc: usado para representar otros tipos.

2. Implementación de programación genérica

- Restricción de tipos genéricos:
- Es posible restringir el tipo genérico para trabajar con un tipo específico y sus descendientes

```
package cursoP00.persistencia;

/**
 * Clase generica con Tipo genérico que restringe el
 * uso de tipos numéricos.
 * @param <T> recibe tipos concretos.
 */
public class CajaNumeros <T extends Number> {
    private T dato;
    public CajaNumeros(){

    }

    /**
     * Constructor con argumentos de v. de instancia.
     * @param dato recibe el valor para dato de
     * tipo concreto.
     */
    public CajaNumeros(T dato) {
        this.dato = dato;
    }
    public T obtener(){
        return dato;
    }
    public void agrega(T dato){
        this.dato=dato;
    }
}
```

2. Implementación de programación genérica

```
package cursoP00.presentacion;

import cursoP00.persistencia.CajaNumeros;

public class Prueba {

    public static void main(String[] args) {
        //se especifica el tipo concreto cuando se declara la
        //referencia o cuando se crea la instancia
        //El tipo de caja es CajaNumeros<Double>,
        //no sólo CajaNumeros.
        CajaNumeros<Double> caja = new CajaNumeros<Double>(4.3);

        System.out.println("valor de caja: " + caja.obtener());

        //si se descomenta la siguiente línea, marca error,
        //debido que la clase CajaNumeros se restringió a numeros
        //CajaNumeros<String> caja2 = new CajaNumeros<String>();
    }
}
```

3. Uso de bibliotecas estándar de programación genérica.

Principales Colecciones del API Java:



Conjuntos

Un conjunto (**Set**) es una colección desordenada (no mantiene un orden de inserción) y no permite elementos duplicados.

Clases de este tipo: HashSet, TreeSet, LinkedHashSet.

Todas las colecciones se encuentran en `java.util`

3. Uso de bibliotecas estándar de programación genérica.

Principales Colecciones del API Java:

Listas

Una lista (**List**) es una colección ordenada (debido a que mantiene el orden de inserción) pero permite elementos duplicados.

Clases de este tipo: ArrayList, LinkedList y Vector



3. Uso de bibliotecas estándar de programación genérica.

Principales Colecciones del API Java:



Mapas

Un mapa (**Map** también llamado arreglo asociativo) es un conjunto de elementos agrupados con una llave y un valor:

<llave, valor>

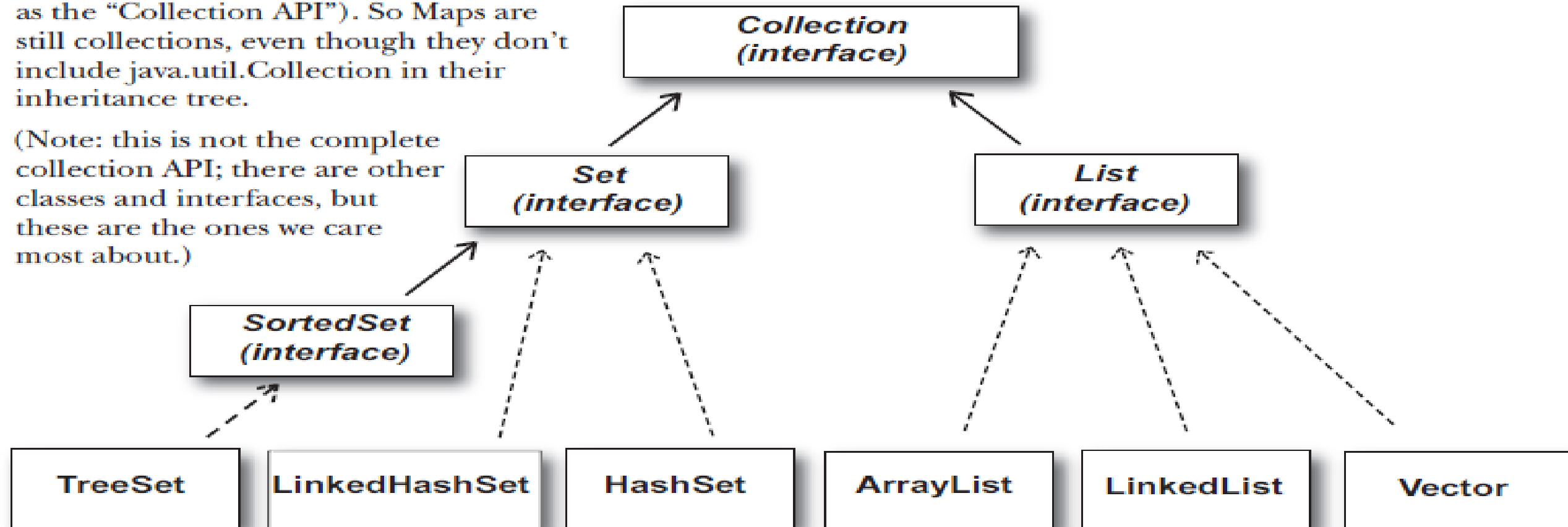
Donde las llaves no pueden ser repetidas y a cada valor le corresponde una llave. La columna de valores sí puede repetir elementos.

Clases de este tipo: HashMap, Hashtable, TreeMap, LinkedHashMap.

The Collection API (part of it) [4]

Notice that the Map interface doesn't actually extend the Collection interface, but Map is still considered part of the "Collection Framework" (also known as the "Collection API"). So Maps are still collections, even though they don't include `java.util.Collection` in their inheritance tree.

(Note: this is not the complete collection API; there are other classes and interfaces, but these are the ones we care most about.)



3. Uso de bibliotecas estándar de programación genérica. [4]

KEY



extends



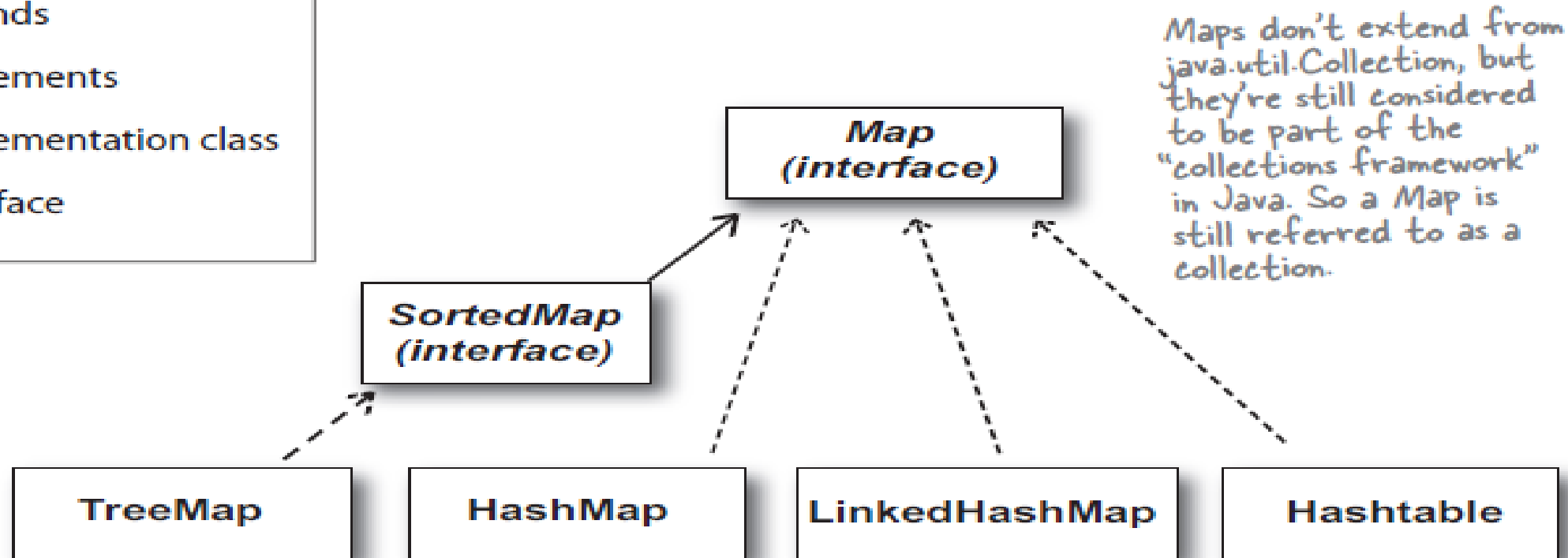
implements

HashSet

implementation class

Set

interface



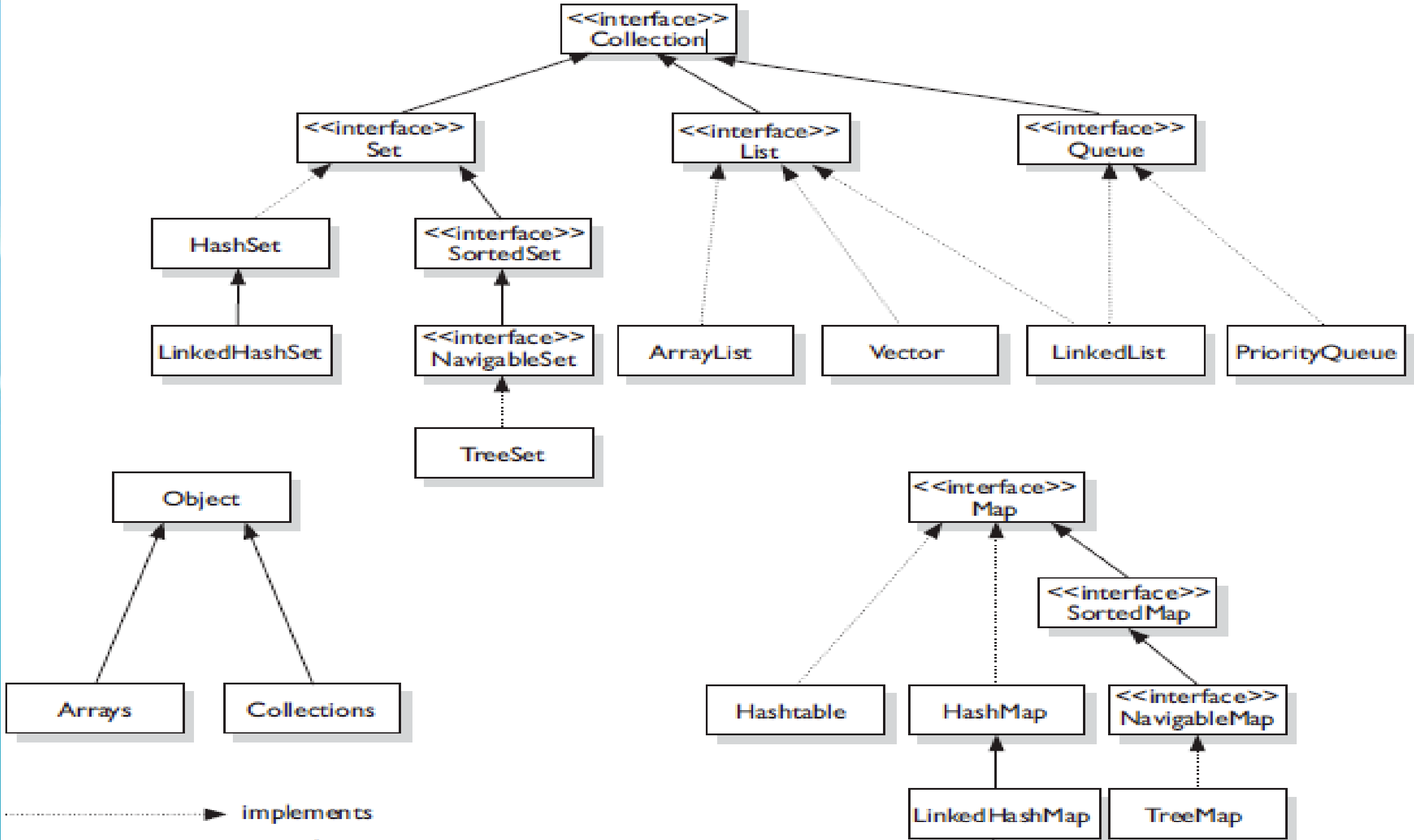


Diagrama de clases List [7]

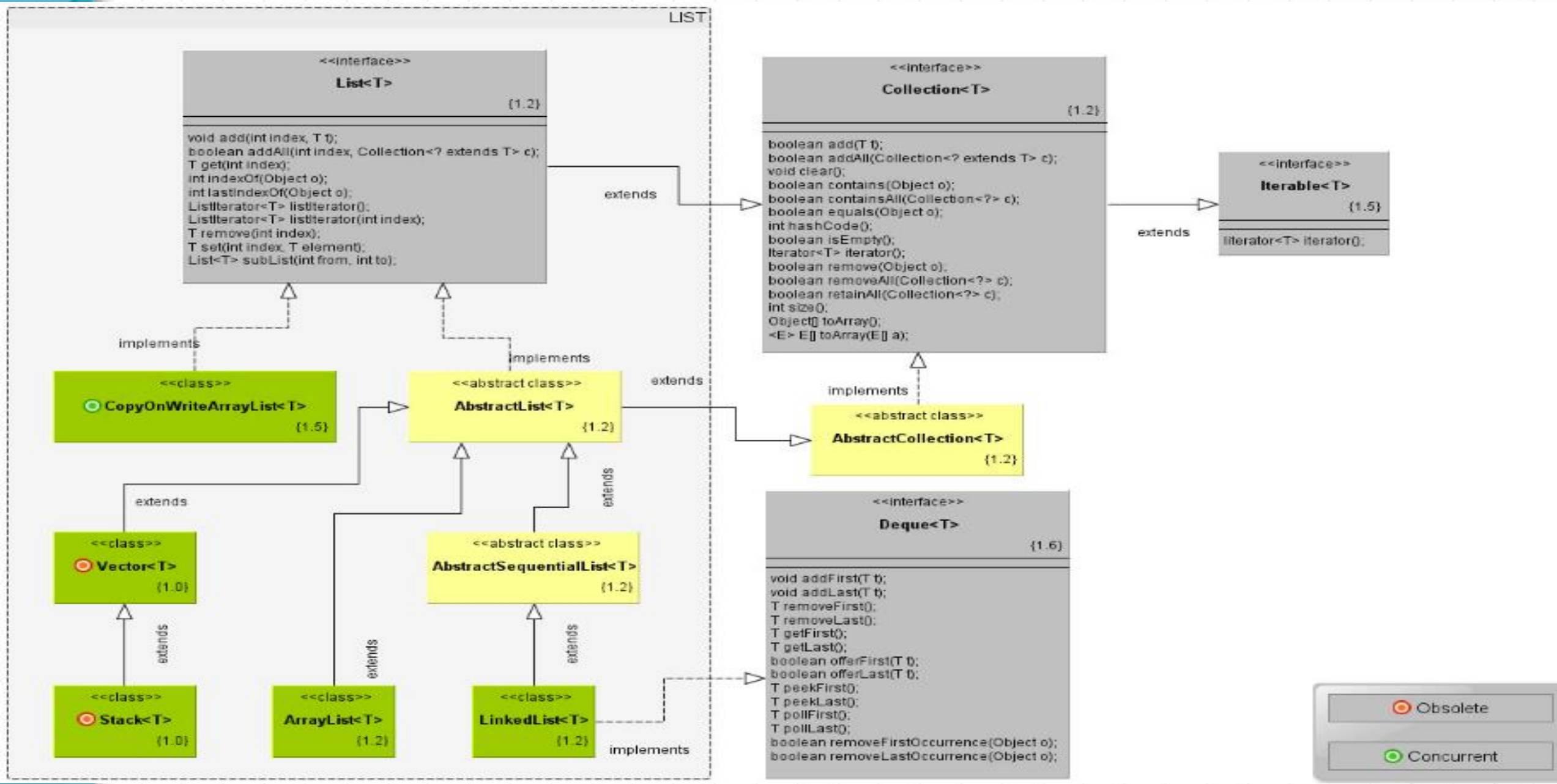
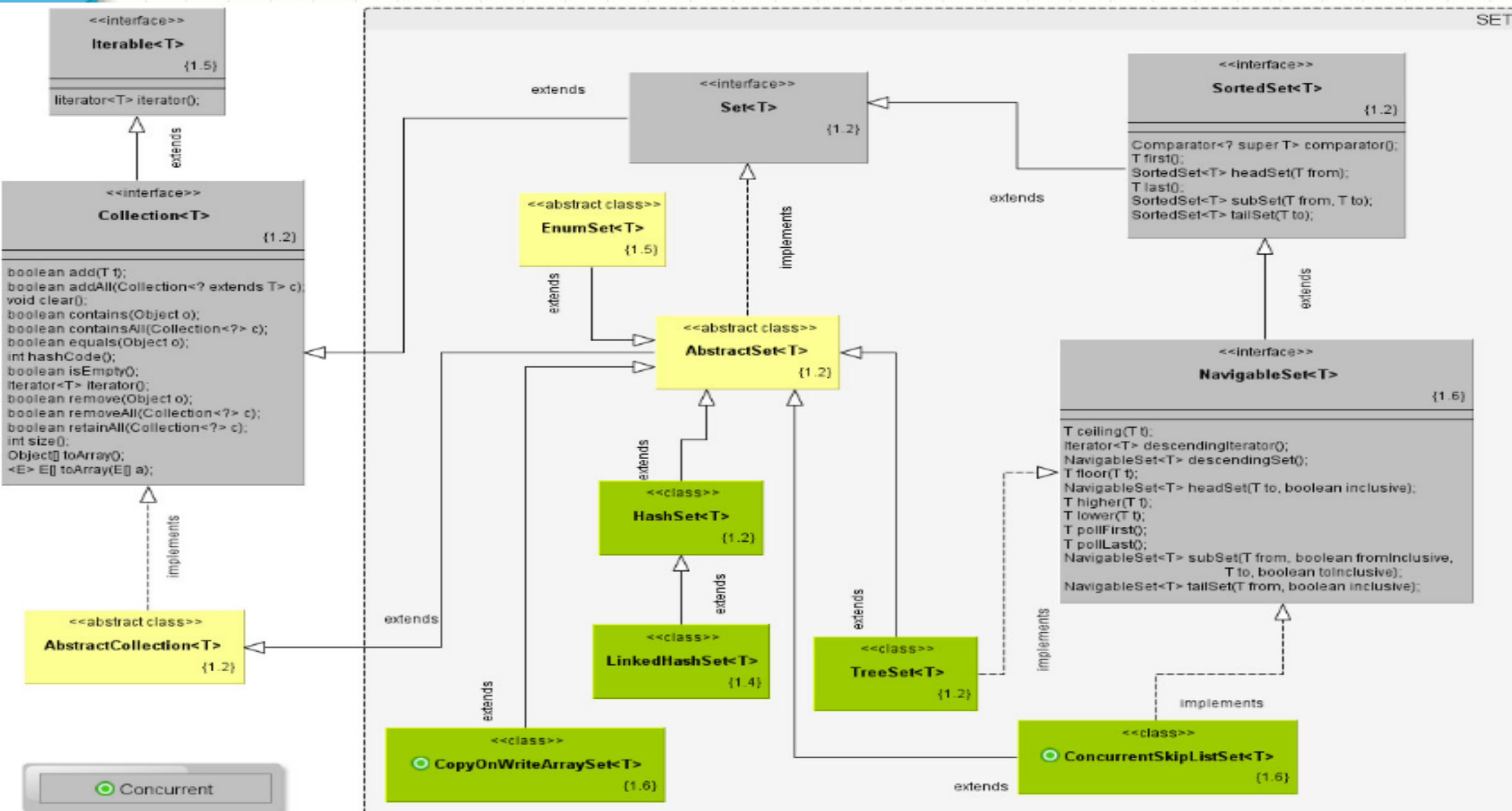
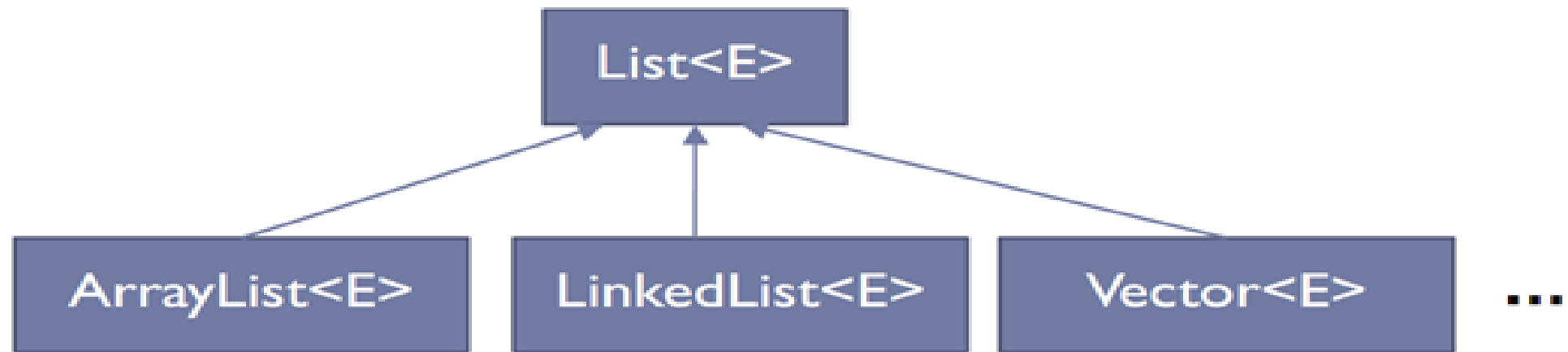


Diagrama de clases Set [7]



3. Uso de bibliotecas estándar de programación genérica.

- La interfaz `List<E>` permite definir una colección de elementos:
 - El usuario controla en que punto de la lista se realizan las inserciones (por defecto se añaden al final)



3. Uso de bibliotecas estándar de programación genérica.

- La clase genérica estándar `ArrayList`
 - Define un grupo de objetos de tipo `E` a los que se accede a través de un índice
 - Permite el crecimiento dinámico de la estructura de datos

```
List<String> list = new ArrayList<String>();  
list.add("hola");  
String s = list.get(0);
```
 - El método `get` puede lanzar `IndexOutOfBoundsException`

Ejemplo 1. Ordenamiento de un ArrayList de cadenas

```
public class JukeBox {  
    ArrayList<String> lista = new ArrayList<String>();  
  
    public void inicializa() {  
        String arreglo[] = new String[5];  
        arreglo[0] = new String("y sin embargo");  
        arreglo[1] = new String("solo si me perdonas");  
        arreglo[2] = new String("ojala");  
        arreglo[3] = new String("nos sobran los motivos");  
        arreglo[4] = new String("el extranjero");  
  
        for (int i = 0; i < 5; i++) {  
            lista.add(arreglo[i]);  
        }  
    }  
  
    public void ejecutar() {  
        System.out.println("Datos desordenados:");  
        System.out.println(lista);  
        Collections.sort(lista);  
        System.out.println("Datos ordenados");  
        System.out.println(lista);  
    }  
}
```

El ArrayList por si solo no soporta un método de ordenamiento, sin embargo, al ser una colección, se puede invocar el método estático *sort* de la clase Collections

Ejemplo 1. Ordenamiento de un ArrayList de cadenas

Se presenta una clase de prueba para ver el ordenamiento canciones de tipo String.

```
public class PruebaJukebox {  
  
    public static void main(String args[]) {  
        Jukebox jukebox=new Jukebox();  
        jukebox.inicializa();  
        jukebox.ejecutar();  
    }  
}
```

```

public class Cancion{
    private String titulo;
    private String artista;

    public Cancion(String titulo, String artista) {
        this.titulo = titulo;
        this.artista = artista;
    }

    public String getTitulo() {
        return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getArtista() {
        return artista;
    }

    public void setArtista(String artista) {
        this.artista = artista;
    }

    public String toString() {
        return titulo + " / " + artista;
    }
}

```

Ejemplo 2. Ordenamiento de objetos Cancion.

Para esto se crearan las
siguientes clases:

- Cancion(titulo, artista)
- JukeboxCancion y,
- PruebaJukeboxCancion

```

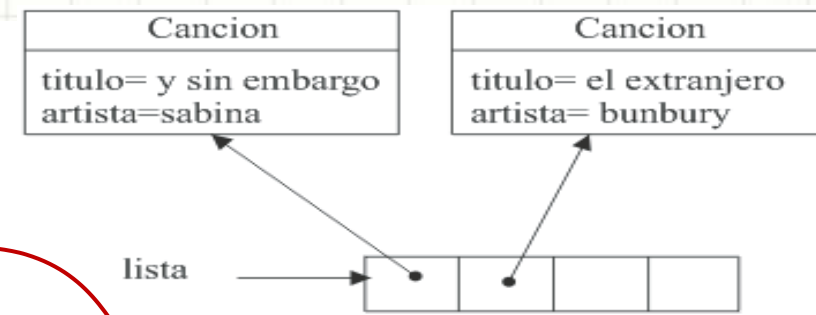
public class JukeboxCancion {
    ArrayList<Cancion> lista = new ArrayList<Cancion>();

    public void inicializa() {
        Cancion canciones[] = new Cancion[5];
        canciones[0] = new Cancion("y sin embargo", "sabina");
        canciones[1] = new Cancion("solo si me perdonas", "bunbury");
        canciones[2] = new Cancion("ojala", "silvio");
        canciones[3] = new Cancion("nos sobran los motivos", "sabina");
        canciones[4] = new Cancion("el extranjero", "bunbury");

        for (int i = 0; i < 5; i++) {
            lista.add(canciones[i]);
        }

        public void ejecutar() {
            System.out.println("Datos desordenados:");
            System.out.println(lista);
            Collections.sort(lista);
            System.out.println(lista);
            System.out.println(lista);
        }
    }
}

```



En la clase JukeBoxCancion, en arrayList, String por Cancion, pero el método sort detecta un error que enseguida se corrige.

The method sort(List<T>) in the type Collections is not applicable for the arguments (ArrayList<Cancion>)

1 quick fix available:

(+) [Cast argument 'lista' to 'List<T>'](#)

Ejemplo 2. Ordenamiento de Objetos Canción en un ArrayList

Method Detail

sort

```
public static <T extends Comparable<? super T>> void sort(List<T> list)
```

Sorts the specified list into ascending order, according to the natural ordering of its elements. All elements in the list must implement the Comparable interface. Furthermore, all elements in the list must be *mutually comparable* (that is, `e1.compareTo(e2)` must not throw a `ClassCastException` for any elements `e1` and `e2` in the list).

Documentación de Java

String (Java Platform SE 8)

docs.oracle.com/javase/8/docs/api/index.html

Aplicaciones Bookmarks PROBLEMAS RESUE... Iusacell ... Libert... Prodigy MSN Méxic... El Laboratorio de la... Portal de usuarios E... PSP Developers - C...

java.beans
java.beans.beancontext
java.io
java.lang
java.lang.annotation
java.lang.instrument
java.lang.invoke
java.lang.management
java.lang.ref
java.lang.reflect
java.math
Streamable
StreamableValue
StreamCorruptedException
StreamFilter
StreamHandler
StreamPrintService
StreamPrintServiceFactory
StreamReaderDelegate
StreamResult
StreamSource
StreamSupport
StreamTokenizer
StrictMath
String

Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods	Deprecated Methods
Modifier and Type	Method and Description			
char	charAt (int index) Returns the char value at the specified index.			
int	codePointAt (int index) Returns the character (Unicode code point) at the specified index.			
int	codePointBefore (int index) Returns the character (Unicode code point) before the specified index.			
int	codePointCount (int beginIndex, int endIndex) Returns the number of Unicode code points in the specified text range of this String.			
int	compareTo (String anotherString) Compares two strings lexicographically.			
int	compareToIgnoreCase (String str)			

```
package cursoP00.persistencia;
```

```
public class Cancion implements Comparable<Cancion> {  
    private String titulo;  
    private String artista;  
  
    public Cancion(String titulo, String artista) {  
        this.titulo = titulo;  
        this.artista = artista;  
    }  
    @Override  
    public int compareTo(Cancion o) {  
        return titulo.compareTo(o.getTitulo());  
    }  
    public String getTitulo() {  
        return titulo;  
    }  
    public void setTitulo(String titulo) {  
        this.titulo = titulo;  
    }  
    public String getArtista() {  
        return artista;  
    }  
    public void setArtista(String artista) {  
        this.artista = artista;  
    }  
    public String toString(){  
        return titulo+ " / "+artista;  
    }  
}
```

Ejemplo 2. Ordenamiento de Objetos Canción en un ArrayList

El método *sort*, requiere que los objetos implementen la interfaz Comparable y el método compareTo

- Ahora cree la clase de prueba para ver los resultados

Ejemplo 3. Ordenamiento de objetos Cancion por título y después por artista.

Para esto solo realice lo siguiente a su proyecto anterior:

- Agregue la clase ArtistaCompara que implemente la interfaz Comparator con el tipo Cancion
- JukeboxCancion y,
- PruebaJukeboxCancion

Ejemplo 3. Ordenamiento de objetos Cancion por título y después por artista.

En persistencia agregue la clase ArtistaCompara y que implemente la interfaz Comparator, la cual requerirá que se implemente el método compare, donde comparará los artistas de 2 canciones.

```
public class ArtistaCompara implements Comparator<Cancion> {  
  
    @Override  
    public int compare(Cancion uno, Cancion dos) {  
        return uno.getArtista().compareTo(dos.getArtista());  
    }  
  
}
```

Ejemplo 3. Ordenamiento de objetos Cancion por título y después por artista.

Ahora actualice el método **ejecutar** de la clase **JukeboxCancion**, note el objeto de la clase **ArtistaCompara**, y como parámetro en la segunda llamada al método **sort**, el ordenamiento por artista. Por último consulte los resultados con la clase **Prueba**

```
public void ejecutar() {  
    System.out.println("Datos desordenados:");  
    System.out.println(lista);  
    Collections.sort(lista);  
    System.out.println("Datos ordenados por título de canción");  
    System.out.println(lista);  
  
    ArtistaCompara artistaCompara = new ArtistaCompara();  
    Collections.sort(lista, artistaCompara);  
    System.out.println("Ahora también se han ordenado por artista");  
    System.out.println(lista);  
}
```

3. Uso de bibliotecas estándar de programación genérica: Iteradores



- Un iterador es un mecanismo que permite recorrer los elementos de una estructura de datos
- La **interfaz Iterator** es parte del entorno de colecciones en Java, con métodos para desplazarse hacia adelante o hacia atrás y realizar modificaciones en una lista durante las iteraciones
- Los **métodos** son:
 - **hasNext()**, devuelve true si hay más elementos.
 - **next()**, devuelve el siguiente elemento.
 - **remove()**, elimina el último elemento devuelto.

Ejemplo módulo Cliente en capa de persistencia

```
public class Cliente {  
    private int id;  
    private String nombre;  
  
    public Cliente(){  
    }  
    public Cliente(int id, String nombre) {  
        this.id = id;  
        this.nombre = nombre;  
    }  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
    public String getNombre() {  
        return nombre;  
    }  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
}
```

```
package cursoP00.persistencia;  
  
import java.util.List;  
  
public class RegistroCliente {  
  
    public void listar(List<Cliente> lista){  
        for(Cliente temp:lista){  
            System.out.println("id: " +temp.getId());  
            System.out.println("nombre: "+temp.getNombre());  
        }  
    }  
}
```

```
public class PruebaCliente {  
    public static void main(String[] args) {  
        List<Cliente> lista = new ArrayList<Cliente>();  
        Cliente cliente1 = new Cliente(1, "Pedro");  
        Cliente cliente2 = new Cliente(2, "Juan");  
        Cliente cliente3 = new Cliente(3, "Ana");  
        RegistroCliente temp = new RegistroCliente();  
  
        lista.add(cliente1);  
        lista.add(cliente2);  
        lista.add(cliente3);  
        temp.listar(lista);
```

Ejemplo módulo Cliente en capa de presentación usando ArrayList

```
    //Creamos el iterador para recorrer la lista  
    Iterator<Cliente> iterador=lista.iterator();
```

Uso de Iterador

```
    for(int i=0; iterador.hasNext(); i++){  
        Cliente cliente = new Cliente();  
        cliente=(Cliente) iterador.next();  
        //Se intenta eliminar el 3er elemento  
        if(cliente.equals(cliente3)){  
            //Esta linea provoca una excepción al eliminar  
            //el elemento de tipo ConcurrentModificationException  
            //lista.remove(i);  
            //Si se comenta la línea anterior y se descomenta  
            //la siguiente se elimina la excepción.  
            iterador.remove();  
        }  
    }  
    System.out.println("Listados de los clientes después de eliminar el cliente3");  
    temp.listar(lista);  
}
```

3. Uso de bibliotecas estándar de programación genérica.

- La clase genérica estándar LinkedList
 - Se trata de una lista enlazada que proporciona un óptimo acceso secuencial, permitiendo inserciones y borrado de elementos en medio de la lista muy rápidos
 - Sin embargo, el acceso aleatorio es muy lento, en comparación con ArrayList
 - Cuenta entre otros métodos:
 - `addLast()`, `getFirst()`, `getLast()`, `removeFirst()`, `removeLast()`
 - Estos métodos permiten utilizar la lista enlazada como una pila, cola o lista doble.

```
public class PruebaLinkedList {  
  
    public static void main(String[] args) {  
        LinkedList<String> listaEnlazada = new LinkedList<String>();  
  
        listaEnlazada.add("elemento1");  
        listaEnlazada.add("elemento2");  
        listaEnlazada.add("elemento3");  
  
        System.out.println("Contenido de la lista enlazada: " +  
            listaEnlazada);  
        listaEnlazada.addFirst("primer elemento");  
        listaEnlazada.addLast("último elemento");  
        System.out.println("Contenido de la lista enlazada "  
            + "después de la inserción: " +  
            listaEnlazada);  
  
        listaEnlazada.add(1, "en la posición 1");  
        System.out.println("Contenido de la lista enlazada "  
            + "después de agregar en posición 1: " +  
            listaEnlazada);  
  
        listaEnlazada.removeFirst();  
        listaEnlazada.removeLast();  
        System.out.println("Contenido de la lista enlazada después "  
            + "de eliminar 1er y último elemento: " +  
            listaEnlazada);  
    }  
}
```

Ejemplo del uso de LinkedList

3. Uso de bibliotecas estándar de programación genérica.

- La clase Vector (Se aconseja utilizar ArrayList)
 - Útil cuando es necesario un arreglo, pero no se conoce la cantidad de elementos a contener.
 - Un objeto de tipo Vector puede crecer y decrecer dinámicamente conforme se vaya necesitando
 - Algunos métodos:
 - Vector(), constructor, crea un vector inicialmente vacío.
 - void addElement(Object obj), inserta obj al final del vector
 - void setElementAt(Object obj, int indice), inserta obj en la posición del índice.

3. Uso de bibliotecas estándar de programación genérica.

- `void removeElementAt(int indice)`, elimina el objeto de la posición del índice.
- `void clear()`, elimina todos los objetos del vector
- `Object elementAt(int indice)`, retorna el objeto posicionado en el índice.
- `boolean isEmpty()`, retorna true si el vector está vacío
- `int size()`, retorna el número de elementos en el vector.

```
package cursoP00.presentacion;

import java.util.Vector;

public class PruebaVector {
    public static void main(String [] args){
        Vector<String> estados = new Vector<>();

        estados.addElement("Oaxaca");
        estados.addElement("Puebla");
        estados.addElement("Guerrero");
        estados.addElement("Veracruz");

        System.out.println("Contenido del vector: " + estados);
        estados.removeElement("Guerrero");

        System.out.println("Contenido del vector después de eliminacion: "+estados);
        System.out.println("En la posicion 1 está: "+estados.elementAt(1));

        estados.insertElementAt("Tabasco", 2);
        System.out.println("Tamaño de los estados después de inserción"+estados.size());

        System.out.println("Contenido: ");
        for(int i=0; i<estados.size();i++)
            System.out.println(estados.elementAt(i));
    }
}
```

Ejemplo de HashMap

- Un Mapa permite valores duplicados, pero no las llaves <llave,valor>.
- Retomando el ejercicio de la programación y sintonización de canales. Se utilizará un Mapa (Map).

Television

- marca : String
- tamaño : int
- interruptor: int
- volumen : int
- canales : Map<Integer,Integer>
- canal : int

+ Television()
+ Television(String, int, int, int, Map<Integer,Integer>, int)
+ encendido()
+ apagado()
+ subirVolumen()
+ bajarVolumen()
+ programarCanales()
+ deshabilitarCanal()
+ imprimirDatosTelevision()
+ imprimirCanales()
// métodos gette's y setter's.

Ejemplo de HashMap

Atributo canales de tipo Map

```
package cursoP00.persistencia;

import java.util.HashMap;

public class Television {
    private String marca;
    private float tamaño;
    private int interruptor;
    private int volumen;
    private Map<Integer, Integer> canales = new HashMap<Integer, Integer>(); // no
                                                                    // aceptan
                                                                    // valores
                                                                    // duplicados.

    private int canal;

    public Television() {
    }

    public Television(String marca, float tamaño, int interruptor, int volumen, Map<Integer, Integer> canales,
        int canal) {
        this.marca = marca;
        this.tamaño = tamaño;
        this.interruptor = interruptor;
        this.volumen = volumen;
        this.canales = canales;
        this.canal = canal;
    }
}
```

Ejemplo de HashMap

```
public void programarCanales() {
    int i, canalAleatorio;
    Random random = new Random();
    System.out.println("En proceso Programación de canales ...");
    for (i = 0; i < 10; i++) {
        canalAleatorio = random.nextInt(10) + 1;
        System.out.println("Numero de canal generado: " + canalAleatorio);
        // hashmap no agrega claves repetidas.
        canales.put(canalAleatorio, 1);
    }
}

public void deshabilitarCanal(int num) {
    if (this.getCanales().containsKey(num)) {
        System.out.println("Se va deshabilitar el canal " + num);
        this.getCanales().put(num, 0);
    }
}
}
```

Ejemplo de HashMap

```
public void imprimirCanales() {  
    // Imprimimos el Map con un Iterador  
    Iterator<Integer> it = this.getCanales().keySet().iterator();  
    Integer key;  
    System.out.println("Canales que han sido programados");  
    while (it.hasNext()) {  
        key = (Integer) it.next();  
        System.out.println("Clave: " + key + " -> Valor: " + this.getCanales().get(key));  
    }  
}  
  
public void imprimirDatosTelevision() {  
    System.out.println("Datos del televisor");  
    System.out.println("Marca: " + this.marca);  
    System.out.println("Tamaño: " + this.tamano);  
    System.out.println("Estado: " + this.interruptor);  
    System.out.println("Canal: " + this.canal);  
    System.out.println("canales programados:");  
    imprimirCanales();  
}
```

Ejemplo de HashMap

```
public class PruebaTelevision {  
  
    public static void main(String[] args) {  
        List<Integer> listaCanales;  
        Television miTV = new Television();  
  
        miTV.setMarca("Sony");  
        miTV.setInterruptor(1);  
        miTV.setTamano(17);  
        miTV.setVolumen(3);  
        miTV.setCanal(100);  
  
        miTV.programarCanales();  
        miTV.subirVolumen();  
        miTV.imprimirDatosTelevision();  
  
        // Iterator<Integer> it = miTV.getCanales().keySet().iterator();  
        // while(it.hasNext()){  
        //     listaCanales.add((Integer)it.next());  
        // }  
        listaCanales = new ArrayList<Integer>(miTV.getCanales().keySet());  
  
        if(listaCanales.size()!=0){  
            miTV.deshabilitarCanal(listaCanales.get(0));  
            System.out.println("Después de deshabilitar un canal");  
            miTV.imprimirDatosTelevision();  
        }  
    }  
}
```

Ejemplo de HashSet

Un conjunto (**Set**) es una colección desordenada (no mantiene un orden de inserción) y no permite elementos duplicados.

```
package cursoPOO.presentacion;

import java.util.ArrayList;
import java.util.HashSet;

public class PruebaHashSet {

    public static void main(String args[]){
        ArrayList<String> numeros = new ArrayList<String>();
        numeros.add("1");
        numeros.add("2");
        numeros.add("3");
        numeros.add("2");
        System.out.println("Contenido del ArrayList: "+numeros);

        //Convertir un ArrayList en un HashSet
        HashSet<String> numerosSet = new HashSet<String>(numeros);
        System.out.println("Contenido del HashSet a partir del ArrayList: "+numerosSet);
        System.out.println("Intento de insertar el 2: "+numerosSet.add("2"));
        System.out.println("Consultando si se encuentra el 3: "+numerosSet.contains("3"));
    }
}
```

Referencias

1. Joyanes, Aguilar Luis. Programación Orientada a Objetos. 2ª edición. España, McGraw-Hill.
2. Nakayama C. Angélica, Solano Gálvez Jorge A. Guía práctica de estudio 07: Herencia. UNAM.
3. Rivera, López Rafael. Programación Orientada a Objetos con Java. Instituto Tecnológico de Veracruz.
4. Sierra Kathy & Bates Bert. Head First Java. O'Reilly Media. 2005
5. <https://en.proft.me/2013/11/3/java-collection-framework-cheat-sheet/>
6. <http://apuntes-java.blogspot.com/2011/10/vector-vs-arraylist.html>
7. <https://www.luaces-novo.es/guia-de-colecciones-en-java/>