```
public class Alumno implements Comparable<Alumno>{
    ...

    public int compareTo(Alumno al) {
        if (NP < al.NP)
            return -1;
        else if (NP > al.NP)
            return 1;
        else
            return 0;
    }
    ...
}
```

Ejemplo 6.10. Métodos que permiten comparar dos alumnos por NP, utilizando el método `compareTo()`.

En una clase que implementa la interfaz `Comparable` si también se sobrescribe `equals()` debería implementarse de forma que cuando el método `equals()` devuelva `true`, el método `compareTo()` devuelve `0`, indicando también que son iguales. Esta forma de implementarlos genera un *orden total* en la clase.

La clase Comparable se ha indicado que se realiza sobre la clase Alumno. Se trata de instanciación de genéricos que se tratarán con detalle en el Capítulo 9.

COLECCIONES

En un programa que no sea trivial siempre es necesario guardar cierta cantidad de datos. Como ya se ha visto en secciones anteriores se puede hacer utilizando arrays, ya sean de una dimensión o de varias dimensiones. Sin embargo, en muchas ocasiones son necesarias otras estructuras de datos. En este libro no se va a tratar sobre el tema de las distintas estructuras de datos que se pueden utilizar pero sí se van a comentar en esta sección un conjunto de estructuras predefinidas y la filosofía que hay tras ellas.

PRECAUCIÓN: Para entender cómo funcionan las colecciones debería haber estudiado primero el Capítulo 8 sobre interfaces y tener cierto manejo sobre la extensión de clases.

La estructura de las colecciones puede verla resumida en la Figura 6.1. En ella se ven las principales interfaces y las clases más útiles que heredan de ellas. En particular, resulta interesante destacar las propiedades de la interfaz `Collection`, ya que la mayoría de las implementaciones más utilizadas derivan de ella. En la Tabla 6.1 puede examinar los principales métodos que puede utilizar.

Las implementaciones que se utilizan en los programas que derivan de Collection a través de las distintas interfaces que extienden Collection se resumen en la Figura 6.2.

Para poder realizar recorridos sobre todos los elementos de una clase se utiliza la interfaz `Iterator`. Como ya ha podido ver en la Tabla 6.1, se puede obtener un objeto `Iterator` de cualquier colección lo que permite, por ejemplo, imprimir los elementos de una colección de la siguiente forma:

```
public void imprimir(){
    Iterator iterador = coleccion.iterator();
    while (iterador.hasNext())
        System.out.println(iterador.next());
}
```

De todas formas para esta tarea resulta muy sencillo también utilizar directamente un bucle `for` que vaya tomando los valores de los elementos de la colección de la siguiente forma:

```
public void imprimir(){
    for(Object elem : coleccion){
        System.out.println((Elemento)elem);
    }
}
```

Tabla 6.1. Métodos de uso frecuente de la interfaz `Collection`

Métodos	Funcionalidad
`int size()`	Devuelve el número de objetos en la colección.
`bolean add(E elemento)`	Añade el elemento de la clase E a la colección. Devuelve true si la colección ha cambiado tras la operación.
`boolean remove (Object elemento)`	Elimina el elemento dado que case con uno de la colección. Devuelve true si la colección ha cambiado tras la operación.
`Iterator<E> iterato()`	Devuelve un iterados para los elementos de esta colección.
`void clear()`	Elimina todos los elementos de la colección.
`boolean contains (Object elemento)`	Devuelve true si la colección contiene un elemento que case con el dado.
`Object[] toArray()`	Devuelve un array con todos los elementos de la colección.

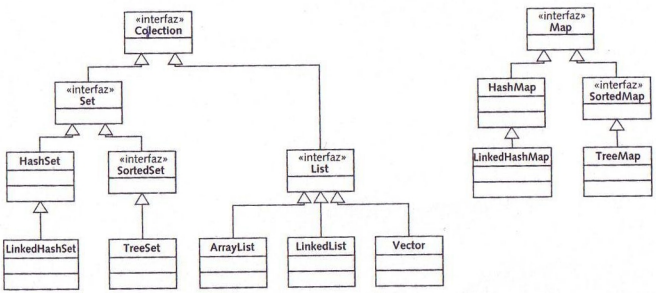

Figura 6.1. Estructura de las colecciones.

Como ejemplo de uso de las colecciones vamos a implementar la clase Grupo vista en la sección anterior utilizando la clase ArrayList en lugar de utilizar un array para guardar los Alumnos del grupo. Esta clase reescrita se puede ver en el Ejemplo 6.11.

```
import java.util.*;

public class Grupo {
    private String nombre;
    private ArrayList<Alumno> alumnos;

    public Grupo(String nombre, int tamaño) throws Exception {
        if (tamaño < 1)
            throw new Exception("Tamaño insuficiente");
        this.nombre = nombre;
        alumnos = new ArrayList<Alumno> (20);  // Se crea el grupo con una
cierta capacidad
    }

    public boolean estáVacío() {
        return alumnos.isEmpty();
    }

    public boolean estáLleno() {
        return false;
    }

    public void añadir(Alumno alumno) throws Exception {
        alumnos.add(alumno);
    }

    public boolean eliminar(Alumno alumno) {
        return alumnos.remove(alumno);
    }

    public int buscar(Alumno alumno) {
        return alumnos.indexOf(alumno);
    }

    public void imprimePorNP(){
        ArrayList copia = (ArrayList)alumnos.clone();
        Collections.sort(copia);
        Iterator iterador = copia.iterator();
        while (iterador.hasNext())
            System.out.println(iterador.next());

        for(Object alumno : copia){
            System.out.println((Alumno)alumno);
        }
    }
}
```

Ejemplo 6.11. La clase Grupo utilizando un `ArrayList`.

Fíjese que al declarar los atributos de la clase se ha utilizado la notación ArrayList<Alumno>, indicando que se van a guardar Alumnos. De esta forma se puede adaptar el ArrayList a lo que va a guardar. El tema de los genéricos se tratará en el Capítulo 9.

		Implementaciones				
		Tabla Hash	Array redimensionable	Árbol balanceado	Listas enlazadas	Tabla Hash + Listas enlazadas
Interfaces	Set	HashSet		TreeSet		LinkedHashSet
	List		ArrayList		LinkedList	
	Map	HashMap		TreeMap		LinkedHashMap

Figura 6.2. Cuadro de las clases que implementan la interfaz Collection o sus derivadas.