

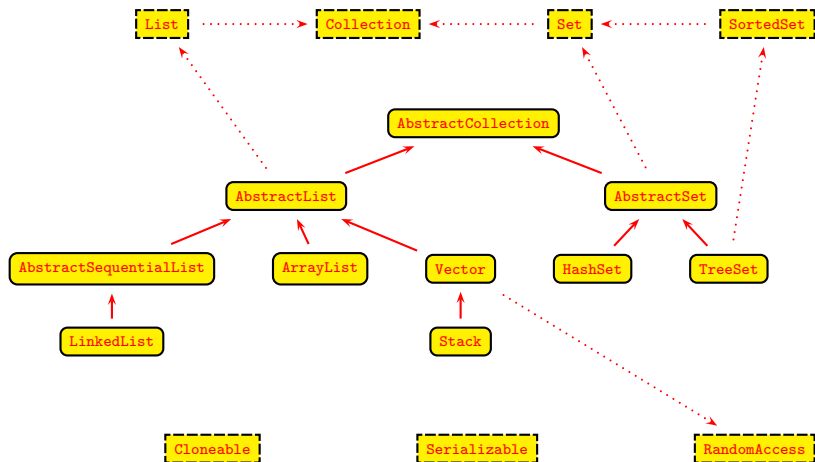
# Colecciones

Luis Fernando Llana Díaz

Departamento de Sistemas Informáticos y Computación  
Universidad Complutense de Madrid

23 de abril de 2007

# Jerarquía de clases de colecciones



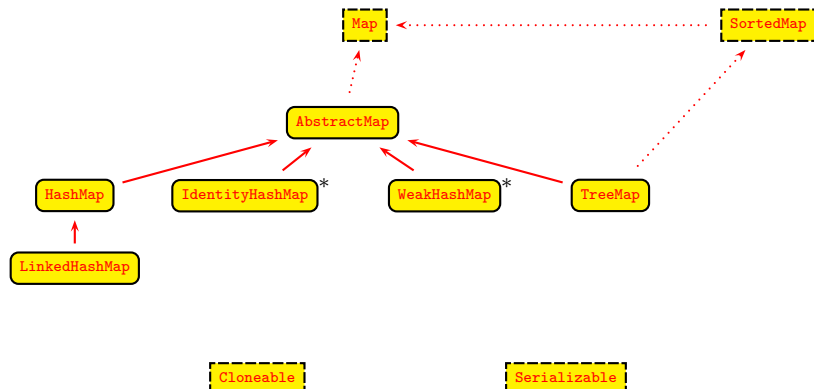
**Collection** Grupo de objetos.

**Set** Concepto matemático. El orden no importa, no hay repeticiones. **SortedSet**.

**List** El orden es importante, admite repeticiones.

**Map** Concepto de función matemática. **SortedMap**

# Jerarquía de clases: Map



# Interfaz Collection

```
public interface Collection<E> extends Iterable<E> {  
    .....  
    boolean add(E element);  
    Iterator<E> iterator();  
    .....  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
}
```

1  
2  
3  
4  
5  
6  
7  
8

# Interface Iterador

```
public Interface Iterator {  
    public boolean hasNext();  
    public Object next();  
    void remove();  
}
```

1  
2  
3  
4  
5

# Uso de listas

```
private static List leePuntos(String fichero)
    throws FileNotFoundException, IOException{
    1
    2
    3
    4
    5
    6
    7
    8
    9
    10
    11
    12
    13
    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    List listaPuntos=new LinkedList();
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        listaPuntos.add(p);
        datos=lector.readLine();
    }
    return listaPuntos;
}
```

```
private static void analizaListaPuntos(List c){
    1
    2
    3
    4
    5
    6
    7
    8
    Iterator itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
        Punto p = (Punto)itr.next();
        .....
    }
}
```

# Errores en tiempo de ejecución

¿Qué pasaría si añadiera un objeto que no sea de clase Punto?

```
Lista lista = leePuntos(fichero);  
lista.add(new Recta(new Punto(0,0), new Punto(1,0)));  
analizaPuntos(lista);
```

1  
2  
3

La línea

```
Punto p = (Punto)itr.next();
```

1

lanzaría un `java.lang.ClassCastException`.

# Clases genéricas

```
public Interface List<E> {  
    .....  
    public Iterator<E> iterator()  
    .....  
}
```

1  
2  
3  
4  
5

```
public Interface Iterator<E> {  
    public boolean hasNext();  
    public E next();  
    void remove();  
}
```

1  
2  
3  
4  
5

# Uso de listas con genéricos

```
private static List<Punto> leePuntos(String fichero)
    throws FileNotFoundException, IOException{

    BufferedReader lector = new BufferedReader(new FileReader(fichero));
    List<Punto> listaPuntos=new LinkedList<Punto>();
    String datos=lector.readLine();
    while (datos!=null){
        Punto p=new Punto(datos);
        listaPuntos.add(p);
        datos=lector.readLine();
    }
    return listaPuntos;
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13

```
private static void analizaListaPuntos(List<Punto> c){
    Iterator<Punto> itr=c.iterator();
    int i=1;
    while(itr.hasNext()){
        Punto p = itr.next();
        .....
    }
}
```

1  
2  
3  
4  
5  
6  
7  
8

# Errores en tiempo de compilación

¿Qué pasaría si añadiera un objeto que no sea de clase Punto?

```
Lista<Punto> lista = leePuntos(fichero);  
lista.add(new Recta(new Punto(0,0), new Punto(1,0)));
```

1  
2

No compila, la lista es de puntos y puedo añadir rectas.

# Interfaz `Collection`

```
public interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object element);
    boolean add(E element);
    boolean remove(Object element);
    Iterator<E> iterator();

    boolean containsAll(Collection<?> c);
    boolean addAll(Collection<? extends E> c);
    boolean removeAll(Collection<?> c);
    boolean retainAll(Collection<?> c);
    void clear();

    Object[] toArray();
    <T> T[] toArray(T[] a);
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17

Los métodos `contains` y `remove` comparan los objetos según el método `equals`.

# Interfaz **Set**

- No añade métodos.
- No puede haber elementos repetidos.

## Subclases

**HashSet** La mejor en eficiencia, los iteradores no garantizan el orden. Método **hashCode** en la clase **Object**.

**TreeSet** Implementación con árboles roji-negros. Los iteradores recorren el conjunto según el orden establecido por sus elementos

**LinkedHashSet** Otra implementación de las tablas hash.

# Método `equals`

Definido en la clase `Object`. Hay que sobrescribirlo si se va a usar una clase como elemento de una colección.

- Debe ser una relación de orden: `o1!=null`, `o2!=null` y `o3!=null`
  - Reflexivo `o1.equals(o1)`
  - Simétrico `o1.equals(o2)==o2.equals(o1)`
  - Si `o1.equals(o2)` y `o2.equals(o3)` entonces `o1.equals(o3)`
- Si `o!=null` entonces `o.equals(null)==false`.
- Debe ser consistente: distintas invocaciones a lo largo de un programa debe dar el mismo resultado si no cambian el valor de los objetos.

## Método `hashCode`

Definido en la clase `Object`. Hay que sobrescribirlo si se va a usar una clase como elemento de una colección tipo *hash*.

- Debe ser consistente.
- Si `o1.equals(o2)`, entonces `o1.hashCode()=o2.hashCode`.
- No debe ser necesariamente inyectiva, dos objetos distintos pueden tener el mismo valor.
- Debería distribuir uniformemente los objetos.

# Método `compareTo`

```
public class Fecha implements Comparable<Fecha>, Cloneable {  
    .....  
    public int compareTo(Fecha f) {  
        ....  
    }  
    .....  
}
```

1  
2  
3  
4  
5  
6  
7

Implementa relación de orden

$$x.compareTo \begin{cases} < 0 & \text{si } x < y \\ = 0 & \text{si } x = y \\ > 0 & \text{si } x > y \end{cases}$$

Debe ser implementada para tener colecciones ordenadas.

# Método `compareTo`

- $\text{sgn}(x.\text{compareTo}(y)) == -\text{sgn}(y.\text{compareTo}(x))$ .
- Debe ser reflexiva antisimétrica y transitiva, para objetos no nulos
  - si  $x.\text{compareTo}(x) == 0$ .
  - si  $x.\text{compareTo}(y) \leq 0$  y  $y.\text{compareTo}(x) \leq 0$  entonces  $x.\text{compareTo}(z) == 0$  y  $z.\text{compareTo}(x) == 0$ .
  - si  $x.\text{compareTo}(y) \leq 0$  y  $y.\text{compareTo}(z) \leq 0$  entonces  $x.\text{compareTo}(z) \leq 0$ .
- Debe ser consistente con el resto de los objetos. Si  $x.\text{compareTo}(y) == 0$  entonces para todo  $z$  entonces  $\text{sgn}(x.\text{compareTo}(z)) == \text{sgn}(y.\text{compareTo}(z))$ .
- es conveniente que sea consistente con equals:  $x.\text{equals}(y)$  sii  $x.\text{compareTo}(y) == 0$

# PrColeccion

```
public class PrColeccion {
    public static void lee(String nombre, Collection<String> conjunto)
        throws IOException {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(nombre));
            String linea = reader.readLine();
            while (linea!=null) {
                conjunto.add(linea);
                linea = reader.readLine();
            }
        } finally {
            if (reader!=null) {
                reader.close();
            }
        }
    }
    public static void muestra(Collection<String> conjunto) {
        for (String s : conjunto) {
            System.out.println(s);
        }
    }
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24

# PrConjunto

```
public class PrConjunto {
    public static void main(String [] args) throws Exception{
        String tipo = args[1];
        Set<String> conjunto = null;
        if (tipo.equals("hash")) {
            conjunto = new HashSet<String>();
        } else if (tipo.equals("tree")) {
//            Collator c = Collator.getInstance(new Locale("es", "ES"));
//            conjunto = new TreeSet<String>(c);
            conjunto = new TreeSet<String>();
        } else {
            throw new RuntimeException("Clase no implementada: "+tipo);
        }

        PrColeccion.lee(args[0], conjunto);
        PrColeccion.muestra(conjunto);
    }
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18

# Lista de nombres

Fichero `nombres.txt`

Luis	1
Almudena	2
Javier	3
Álvaro	4
Isabel	5
Marisa	6
Natalia	7
Alberto	8
Andrés	9
Andres	10
Andros	11
Luis	12
Almudena	13
Javier	14
Álvaro	15
Isabel	16
Marisa	17
Natalia	18
Alberto	19
Andrés	20
Andres	21
Andros	22
Luis	23
Almudena	24
Javier	25
Álvaro	26

# Ejecución de conjuntos

```
~/Java$ $JAVA_HOME/bin/java PrConjunto nombres.txt tree
```

Alberto

Almudena

Andres

Andros

Andrés

Isabel

Javier

Luis

Marisa

Natalia

Álvaro

1

2

3

4

5

6

7

8

9

10

11

12

# Iteradores

Un iterador es un objeto que sirve para recorrer una estructura

```
public interface Iterator<E> {  
    boolean hasNext();  
    E next();  
}
```

1  
2  
3  
4

```
for (Iterator<String> it = coleccion.iterator(); it.hasNext(); )  
    String s = s.next();  
    System.out.println();  
}
```

1  
2  
3  
4

```
for (String s : coleccion) {  
    System.out.println(s);  
}
```

1  
2  
3

# Listas

El orden de los objetos es importante, puede haber repeticiones de objetos.

```
public interface List<E> extends Collection<E> {
    // Acceso posicional
    E get(int index);
    E set(int index, E element); //optional
    boolean add(E element); //optional
    void add(int index, E element); //optional
    E remove(int index); //optional
    boolean addAll(int index,
        Collection<? extends E> c); //optional

    // Búsquedas
    int indexOf(Object o);
    int lastIndexOf(Object o);

    // Iteradores de listas
    ListIterator<E> listIterator();
    ListIterator<E> listIterator(int index);

    // Sublistas
    List<E> subList(int from, int to);
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21

# Iteradores de listas

```
public interface ListIterator<E> extends Iterator<E> {  
    boolean hasNext();  
    E next();  
    boolean hasPrevious();  
    E previous();  
    int nextIndex();  
    int previousIndex();  
    void remove(); //optional  
    void set(E e); //optional  
    void add(E e); //optional  
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11

# Implementaciones de listas

**ArrayList** Generalmente más eficiente.

**LinkedList** Eficiente cuando no se hacen accesos posicionales.

# Maps

```
public interface Map<K,V> {  
  
    // Operaciones básicas  
    V put(K key, V value);  
    V get(Object key);  
    V remove(Object key);  
    boolean containsKey(Object key);  
    boolean containsValue(Object value);  
    int size();  
    boolean isEmpty();  
  
    void putAll(Map<? extends K, ? extends V> m);  
    void clear();  
  
    // Vistas  
    public Set<K> keySet();  
    public Collection<V> values();  
    public Set<Map.Entry<K,V>> entrySet();  
  
    // Interface for entrySet elements  
    public interface Entry {  
        K getKey();  
        V getValue();  
        V setValue(V value);  
    }  
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26

# PrMap

```
public class PrMap {
    private static void lee(String nombre,
        Map<String,Integer> map) throws IOException {
        BufferedReader reader = null;
        try {
            reader = new BufferedReader(new FileReader(nombre));
            String linea = reader.readLine();
            while (linea!=null) {
                Integer i = map.get(linea);
                if (i==null) {
                    i = new Integer(1);
                } else {
                    i = new Integer(i.intValue()+1);
                }
                map.put(linea,i);
                linea = reader.readLine();
            }
        } finally {
            if (reader!=null) {
                reader.close();
            }
        }
    }
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23

# PrMap

```
private static void muestra(Map<String,Integer> map) {
    for (String s: map.keySet()) {
        System.out.println(s+"="+map.get(s));
    }
}
public static void main(String [] args) throws Exception{
    String tipo = args[1];
    Map<String,Integer> map = null;
    if (tipo.equals("hash")) {
        map = new HashMap<String,Integer>();
    } else if (tipo.equals("tree")) {
        map = new TreeMap<String,Integer>(Collator.getInstance());
    } else {
        throw new RuntimeException("Clase no implementada: "+tipo);
    }

    lee(args[0],map);
    muestra(map);
}
}
```

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21

# Implementaciones

**HashMap** Basadas en tablas hash.

**TreeMap** Basadas en árboles, cuando las claves se pueden ordenar.