```
        // Operaciones algebraicas: unión y diferencia
        TreeSet union = (TreeSet)cor2.clone();
        union.addAll(cor1);
        System.out.println("cor2 + cor1: " + union);
        TreeSet dif = (TreeSet)cor2.clone();
        dif.removeAll(cor1);
        System.out.println("cn2 - cn1: " + dif);

    }
}
```

**Ejecución**

```
Conjunto ordenado, sin comparador: [Libro, Mesa, Papel, Papelera, ar-
mario, globo, mes]
Conjunto ordenado, con comparador: [angela, Esperanza, Julian, lupi,
maite, Maria, marta, Miguel]
No se encuentra  - Mes - en conjunto 1 [Libro, Mesa, Papel, Papelera,
armario, globo, mes]
Eliminado - MAITE - de conjunto 2 [angela, Esperanza, Julian, lupi,
Maria, marta, Miguel]
Subconjunto : [Libro, Mesa, Papel, Papelera]
Subconjunto modificado: [Libro, Marta, Mesa, Papel, Papelera]
Conjunto origen modificado: [Libro, Marta, Mesa, Papel, Papelera, ar-
mario, globo, mes]
Subconjunto modificado: [Libro, Luna, Marta, Mesa, Papel, Papelera]
Conjunto origen modificado: [Libro, Luna, Marta, Mesa, Papel, Papelera,
armario, globo, lupi, mes]
cor2 + cor1: [angela, armario, Esperanza, globo, Julian, Libro, Luna,
lupi, Maria, marta, mes, Mesa, Miguel, Papel, Papelera]
cn2 - cn1: [angela, Esperanza, Julian, Maria, marta, Miguel]
```

## 17.9. MAPAS Y DICCIONARIOS

Un diccionario agrupa elementos identificados mediante claves únicas. En cierto modo, un diccionario es una colección cuyos elementos son pares y están formados por un dato y su clave, que identifica de manera unívoca al elemento. Por ejemplo, el *número de matricula* del conjunto de alumnos puede considerarse un campo clave para organizar la información relativa al alumnado de una universidad. Los diccionarios son contenedores asociativos, también denominados *mapas*.

Las colecciones históricas definen los diccionarios con la clase abstracta `Dictionary` y la clase concreta `HashTable`. La interfaz `Map` especifica los métodos comunes a los mapas de las colecciones desarrolladas en Java 2.

### 17.9.1. Dictionary

La clase abstracta `Dictionary` es la interfaz para crear diccionarios en las colecciones históricas, y procesa la colección como si fuera un *array* asociativo al que se accede por una clave. Tanto el campo dato como la clave de cada elemento del diccionario son objetos. Todos los métodos de la clase son abstractos (se debería haber declarado como interfaz):

```
abstract public boolean isEmpty();
abstract public int size();
abstract public Enumeration keys();          crea una enumeración para las claves.
```

```
abstract public Enumeration elements();        crea una enumeración para los valores.
abstract public Object get(Object clave);      devuelve valor asociado a clave.
abstract public Object put(Object clave, Object valor);    añade clave, valor.
abstract public Object remove(Object clave);   elimina el elemento de la clave.
```

Se especifica que `get()` devuelva `null` si no está la clave en el diccionario. El método `put()` no inserta el par `(clave, valor)` si la `clave` está en el diccionario, en cuyo caso devuelve el valor asociado. `remove()` devuelve `null` si la `clave` no se encuentra. Estos métodos levantan la excepción `NullPointerException` si la clave es `null`.

Se puede crear un tipo propio de diccionario extendiendo `Dictionary` y definiendo los métodos abstractos de la interfaz. Por ejemplo:

```
public class MiDiccionario extends Dictionary
{
   private Vector valor;
   private Vector clave;
   public MiDiccionario()   // constructor
   {
      valor = new Vector();
      clave = new Vector();
   }
   public int size()
   {
      return clave.size();  // número de elementos
   }
   public boolean isEmpy()
   {
      return clave.isEmpty();
   }
   public Object put(Object clave, Object valor)

   ...

}
```

### 17.9.2. Hashtable

Normalmente no es necesario definir una nueva clase diccionario, la clase histórica (Java 1.0) `Hashtable` extiende `Dictionary` y se puede utilizar para agrupar datos asociativos. `Hashtable` se comporta como una tabla *hash*; dispersa los elementos según el código que devuelve el método `hashCode()` del objeto `clave`.

Java 2 ha cambiado la declaración histórica de `Hashtable`, ahora implementa la interfaz `Map` y por consiguiente es compatible con las nuevas clases diccionario. La declaración:

```
public class Hashtable extends Dictionary implements Map, Cloneable,
                                                      Serializable
```

> ***Nota de programación***
>
> La clase de los objetos `clave` debe disponer del método `int hashCode()`, utilizado para dispersar el elemento. También el método `equals()` para poder realizar búsquedas y comparaciones.

### Constructores

El diseño de `Hashtable` se ha realizado con un factor de carga, por defecto, de `0.75`. El constructor de la clase permite establecer una capacidad inicial, y también establecer otro factor de carga (véase el Capítulo 12). Además, se puede inicializar un `Hashtable` con los elementos de un mapa.

```
public Hashtable();                                  tabla vacía, capacidad inicial 11, factor de carga 0.75.
public Hashtable(int cap);                           tabla vacía, capacidad cap, factor de carga 0.75.
public Hashtable(int cap, float factorCarga);        tabla vacía.
public Hashtable(Map mapa);                          crea una tabla con los elementos de mapa.
```

### Métodos

Los métodos de `Hashtable` están especificados en la clase `Dictionary`, y en la interfaz `Map` a partir de Java 2. Además, el método `clone()` y los métodos para *serializar*. A continuación se hace mención a los más interesantes derivados de `Map`.

```
public void putAll(Map mapa);                        inserta lo elementos de mapa.
public boolean containsValue(Object valor);          devuelve true si encuentra valor en
                                                     el conjunto de valores de la tabla.
public boolean containsKey(Object clave);            devuelve true si encuentra clave en
                                                     el conjunto de claves de la tabla.
public Set keySet();                                 devuelve un conjunto con las claves
                                                     de la tabla.
public Collection values();                          devuelve una colección con los
                                                     valores de la tabla.
public Set entrySet();                               devuelve un conjunto con los
                                                     elementos de la tabla.
```

Además, redefine el método `toString()` para obtener una cadena con todos los pares de elementos de la tabla.

---

### Ejemplo 17.12

*Se crea un diccionario `HashTable` cuyos elementos están formados por un campo valor de tipo `String` y una clave de tipo `Double`. Se realizan diversas operaciones con los métodos de `Hashtable`.*

Cada elemento representa a un ciclista, su nombre es el campo valor y el tiempo realizado en una crono es el campo clave. Los nombres y sus tiempos están en sendos *arrays;* para insertar en el diccionario se llama al método `put()`. Una vez creado, se realizan operaciones de búsqueda y eliminación de un elemento, obtener conjunto de claves, etc. Los elementos del diccionario se escriben en pantalla para mostrar los resultados de las operaciones.

```
import java.util.*;

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

```
    String [] ciclo = {"Lansen", "Messaria", "Ripolles", "Waritten",
                       "Delgado", "Kloster", "Bustaria", "Animador",
                       "Sanroma", "Juliani"};
    double [] tiempo = {45.40, 50.40, 47.0, 49.0, 51.20, 46.0, 48.0,
                        45.6, 52.30, 53.25};
    Hashtable tab = new Hashtable();
    // inserta los elementos en la tabla
    for (int i = 0; i < ciclo.length; i++)
      tab.put(new Double(tiempo[i]), ciclo[i]);
    System.out.println("Tabla hash creada: " + tab);
    // búsqueda por clave
    if (tab.containsKey(new Double(46.0)))
      System.out.println("Corredor encontrado: " +
                                      tab.get(new Double(46.0)));
    else
      System.out.println("Clave no está en la tabla");
    // elimina un elemento
    Object q = tab.remove(new Double(45.6));
    if (q != null) System.out.println("Elemento " +
                                                  q + " eliminado");
    // Conjunto de claves
    Set cv;
    cv = tab.keySet();
    System.out.println("Conjunto de claves: " + cv);
    // Enumeración de valores
    Enumeration en;
    en = tab.elements();
    System.out.print("Ciclistas(valores): " );
    while (en.hasMoreElements())
    {
      System.out.print(en.nextElement());
      if (en.hasMoreElements()) System.out.print(", ");
    }
    System.out.println();
  }
}
```

**Ejecución**

```
Tabla hash creada: {53.25=Juliani, 46.0=Kloster, 50.4=Messaria,
47.0=Ripolles, 45.4=Lansen, 48.0=Bustaria, 52.3=Sanroma, 45.6=Animador,
49.0=Waritten, 51.2=Delgado}
Corredor encontrado: Kloster
Elemento Animador eliminado
Conjunto de claves: [53.25, 46.0, 50.4, 47.0, 45.4, 48.0, 52.3, 49.0, 51.2]
Ciclistas(valores): Juliani, Kloster, Messaria, Ripolles, Lansen,
Bustaria, Sanroma, Waritten, Delgado
```

### 17.9.3. Map

Los mapas o diccionarios almacenan información formada por parejas *(valor, clave)*. Java 2 enriquece los mapas creando una jerarquía exclusiva para ellos, la raíz es la interfaz `Map`, que, a diferencia de `Set` y `List`, no deriva de `Collection`. La Figura 17.5 muestra esta jerarquía.

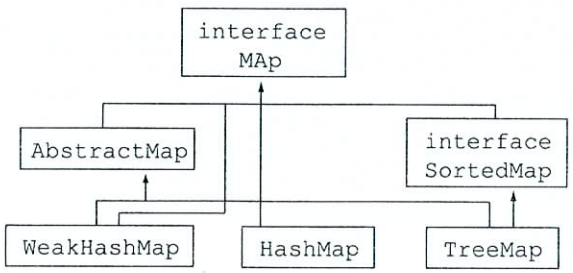


**Figura 17.5** Jerarquía de *mapas*

La interfaz `Map` establece el comportamiento común de todos las implementaciones concretas de diccionarios, declara los siguientes métodos:

```
int size();
boolean isEmpty();
// métodos de búsqueda, por clave o por valor
boolean containsKey(Object clave);
boolean containsValue(Object valor);
Object get(Object clave);
// métodos que modifican el mapa, añaden o eliminan
Object put(Object clave, Object valor);   Si la clave existe reeplaza el valor antiguo
                                          y devuelve éste, en caso contraio null.
void putAll(Map mapa);
Object remove(Object clave);   Si la clave existe elimina el elemento y devuelve el
                               valor asociado, en caso contraio null.
// métodos para obtener una "visión" del mapa, el conjunto
// de clave, o los valores, o bien el conjunto de elementos.
Set keySet();
Collection values();
Set entrySet();
```

### AbstractMap

Esta clase abstracta implementa todos los métodos de la interfaz `Map`, excepto `entrySet()` que es necesario definirlo en las clases concretas. Además, define el método `toString()` de tal forma que devuelve una cadena con los pares de elementos que forman el mapa.

## 17.9.4. HashMap

Los mapas tipo `HashMap` organizan los elementos en una *tabla hash*, proporciona una eficiencia constante a las operaciones de búsqueda e inserción. Esta clase permite que haya claves y valores `null`, a diferencia de `Hashtable` que levanta una excepción si la clave es `null`. En general, el comportamiento de `HashMap` es similar a `Hashtable`. La declaración de la clase es la siguiente:

```
public class HashMap extends AbstractMap implements Map, Cloneable,
                                                     Serializable
```

Los elementos pares de un mapa, `(clave, valor)`, se guardan en un *array* de objetos de tipo `Entry`: `Entry [] tabla`; siendo esta una clase `static` declarada en `HashMap` que agrupa los

dos campos del elemento junto a su dirección *hash*. Además dispone de ciertos métodos que en ocasiones resultan útiles, los más interesantes son los siguientes:

| | |
|---|---|
| `public Object getKey();` | devuelve la clave de la entrada (elemento). |
| `public Object getValue();` | devuelve el valor de la entrada (elemento). |
| `public Object setValue(Object nuevoVal);` | cambia el valor de la entrada por `nuevoVal` y devuelve el valor antiguo. |
| `public String toString();` | cadena con elemento, "clave = valor". |

El acceso a los elementos del mapa sólo se puede realizar mediante un iterador al conjunto que devuelve el método `entrySet()`.

### Constructores

El factor de carga de `HashMap` es, por defecto, `0.75`, aunque se puede poner otro en el constructor. La capacidad de la tabla se establece en el constructor, o bien se supone la capacidad por defecto si se utiliza el constructor vacío. También se puede crear un `HashMap` a partir de los elementos de otro *mapa*. Entonces los constructores son los siguientes:

| | |
|---|---|
| `public HashMap();` | tabla vacía, factor de carga 0.75 y capacidad por defecto. |
| `public HashMap (int cap);` | tabla vacía de capacidad cap y factor de carga 0.75. |
| `public HashMap (int cap, float factorCarga);` | tabla vacía. |
| `public HashMap (Map mapa);` | crea una tabla con los elementos de mapa. |

### Métodos

La clase dispone de los métodos heredados de `AbstractMap` y los definidos al implementar los interfaz `Map`, `Cloneable` y `Serializable`. Por consiguiente, se puede buscar por clave o por valor con `containsKey()` y `containsValue()` respectivamente, insertar un elemento con `put()`, o bien insertar todos los elementos de otro mapa con `putAll()`. El método `remove()` elimina el elemento cuya clave es el argumento. Se puede obtener el conjunto de claves con `keySet()`, la colección de valores con `values()` y el conjunto de elementos con `entrySet()`. También, el método `get()` que obtiene el valor correspondiente a una clave.

La clase declara el método `static int hash(Object x)` para obtener una dirección *hash* adicional a `hasCode()`. El Ejemplo 17.13 crea un mapa `HashMap` con el que se realizan operaciones.

---

### Ejemplo 17.13

*Crear un mapa cuyos elementos son alumnos de una universidad; la clave de cada alumno es su número de matrícula.*

Se simplifica el ejemplo de tal forma que un alumno es un `String` con su nombre y la clave es un número aleatorio de `1` a `99`. Los alumnos se encuentran en un *array* inicializado con nombres al azar. La entrada de elementos se realiza en un bucle de tantas iteraciones como número de nombres; al ser claves aleatorias, puede repetirse alguna de ellas. La redefinición de `toString()` permite escribir el mapa en pantalla en una sola acción; también se escribe elemento a elemento con un iterador del conjunto de claves. Se realiza la búsqueda de un elemento utilizando un iterador al conjunto de elementos; si se encuentra se cambia el valor (en este caso, el nombre) por otro nuevo.

```
import java.util.*;

public class MapaUniv
{
  public static void main(String[] args)
  {

    String [] univ = {"Ramiro", "Melendez", "Santos", "Armando",
     "Delgado", "Martina", "Bueno", "Alonso A", "Samuel", "Julian";
    Map mp;
    mp = new HashMap(16);  // mapa vacío de capacidad inicial 16
    for (int i = 0; i < univ.length; i++)
    {
      int matricula;
      matricula = (int)(Math.random()*99 +1);
      if (! mp.containsKey(matricula))
        mp.put(new Integer(matricula), univ[i]);
      else i--;
    }
    System.out.println("Mapa creado: " + mp);
    // Iterador del conjunto de claves
    Set cv;
    cv = mp.keySet();
    Iterator it = cv.iterator();
    System.out.println("Recorre el Mapa con iterador de claves. ");
    while (it.hasNext())
    {
      Object clave;
      clave = it.next();
      System.out.print("(" + clave + "," + mp.get(clave) + ")");
      if (it.hasNext()) System.out.print(", ");
    }
    System.out.println();
    // cambio del valor de un elemento
    Set cel = mp.entrySet();
    Iterator ite = cel.iterator();
    System.out.println("Cambio de primera clave. ");
    Map.Entry elemento = (Map.Entry)ite.next();
    System.out.println("Valor elemento:" + elemento.getValue()
                       + ", clave: " + elemento.getKey());
    elemento.setValue("Zacarias");
    System.out.println("Mapa modificado: " + mp);
  }
}
```

### Ejecución

```
Mapa creado: {15=Alonso A, 68=Bueno, 62=Armando, 19=Ramiro,
 31=Delgado, 95=Julian, 45=Martina, 97=Melendez, 70=Samuel, 88=Santos}
Recorre el Mapa con iterador de claves.
(15,Alonso A), (68,Bueno), (62,Armando), (19,Ramiro), (31,Delgado),
(95,Julian), (45,Martina), (97,Melendez), (70,Samuel), (88,Santos)
Cambio de primera clave.
Valor elemento:Alonso A, clave: 15
Mapa modificado: {15=Zacarias, 68=Bueno, 62=Armando, 19=Ramiro,
31=Delgado, 95=Julian, 45=Martina, 97=Melendez, 70=Samuel, 88=Santos}
```