



# Taller de Programación

---

**Parte V**

**Estructura de Datos**

Jhonny Felípez Andrade

[jrfelizamigo@yahoo.es](mailto:jrfelizamigo@yahoo.es)



# Contenido

---

- El paquete de Java `java.util`
- Pilas.
- Listas enlazadas.
- Diccionarios.
- Cola de prioridad.
- Conjuntos.
- Árboles



# Paquete de Java `java.util`

---

- Muchos objetos útiles del Java estándar están incluidos en el paquete `java.util`.
- Para declarar un objeto que sea una estructura de datos general.

```
Map miMap = new HashMap();
```

# Paquete de Java java.util

- Una adecuada implementación de las estructuras de datos básicas debe incluir:

Estructura De datos	Clases abstractas	Clases concretas	Funciones
Pila	Sin interface	Stack	pop, push, empty, peek
Cola	List	ArrayList, LinkedList	add, remove, isEmpty
Diccionarios	Map	HashMap, HashTable	put, get, size
Cola de prioridad	Queue	PriorityQueue	add, remove
Conjuntos	Set	HashSet	add, remove, contains



# Pilas

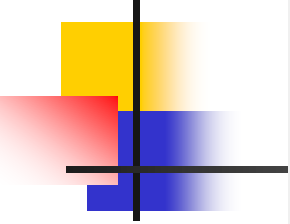
---



# Clase Stack de Java

---

Método	Descripción
push(E elemento)	Agrega un elemento a la pila.
pop()	Extrae el último elemento introducido.
peek()	Lee el último elemento introducido.
empty()	Devuelve true si la pila está vacía.



```
public class Alumno implements Comparable<Alumno> {
    int codigo;
    String nombre;

    public Alumno(String nombre, int codigo) {
        this.nombre = nombre;
        this.codigo = codigo;
    }

    public String verNombre() {
        return nombre;
    }

    public int verCodigo() {
        return codigo;
    }

    public int compareTo(Alumno other) {
        return codigo - other.codigo;
    }

    public String toString() {
        return (codigo + " = " + nombre);
    }
}
```

```
import java.util.Stack;
public class Pila {
    public static void main(String[] args) {
        // definir una pila de alumnos
        Stack<Alumno> pila = new Stack<Alumno>();
        // almacenar los alumnos
        Alumno alum1 = new Alumno("Juan", 11);
        pila.push(alum1);
        alum1 = new Alumno("Maria", 17);
        pila.push(alum1);
        alum1 = new Alumno("Jose", 25);
        pila.push(alum1);
        // recuperar el elemento de encima
        System.out.println(pila.peek().verNombre());
        // imprimir la pila
        while (!pila.empty())
            System.out.println(pila.pop().verNombre());
    }
}
```





```
<terminated> Pila [Java Application] C:\
```

```
Jose
```

```
Jose
```

```
Maria
```

```
Juan
```



# Listas Enlazadas

---



# Clase LinkedList de Java

---

Método	Descripción
add(E elemento)	Agrega un elemento a la lista.
add(int i, E elemento)	Agrega un elemento en la posición i.
addFirst(E elemento)	Agrega un elemento al principio.
addLast(E elemento)	Agrega un elemento al final.
clear()	Borra la lista.
addAll(int i,colección)	Agrega toda una colección en la posición i.
remove(int i)	Quita y devuelve el elemento de la posición i.



# Clase LinkedList de Java

---


Método	Descripción
removeFirst()	Quita y devuelve el primer elemento.
removeLast()	Quita y devuelve el último elemento.
set(int i, E elemento)	Cambia el elemento de la posición i.
get(int i)	Obtiene el elemento de la posición i.
isEmpty()	Devuelve true si la lista esta vacía.
contains(E elemento)	Devuelve true si la lista contiene el elemento.
size()	Devuelve el número de elementos.



# Clase Iterador de Java

## Recorre la Lista Enlazada

Método	Descripción
ListIterator(E)	Crea un iterador para visitar elementos de la lista.
set(E elemento)	Cambia el último elemento visitado.
hasPrevious()	Devuelve true si tiene un elemento anterior.
previous()	Devuelve el elemento anterior.
nextIndex()	Devuelve el índice que sería llamado en la próxima iteración.



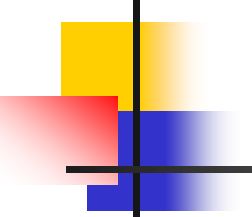
```
import java.util.List;
import java.util.LinkedList;
import java.util.ListIterator;
public class ListaEnlazada {
    public static void main(String[] args) {
        Alumno alum;
        LinkedList<Alumno> uno = new LinkedList<Alumno>();
        uno.add(new Alumno("Juan", 11));
        uno.add(new Alumno("Maria", 17));

        ListIterator<Alumno> aIter = uno.listIterator();

        // desplegar la lista uno
        System.out.println("Listar la lista UNO");
        while (aIter.hasNext()) {
            alum = aIter.next();
            System.out.println(alum.verNombre());
        }

        List<Alumno> dos = new LinkedList<Alumno>();
        dos.add(new Alumno("Jose", 25));
        dos.add(new Alumno("Laura", 8));
        ListIterator<Alumno> bIter = dos.listIterator();
```

```
// agregar los alumnos de lista dos  
// a la lista uno  
while (bIter.hasNext()) {  
    uno.add(bIter.next());  
}  
System.out.println("Despues de agregar la lista DOS");  
// Reiniciar el iterador y listar la lista uno  
aIter = uno.listIterator();  
while (aIter.hasNext()) {  
    alum = aIter.next();  
    System.out.println(alum.verNombre());  
}  
  
// quitar los elementos de la lista dos de uno  
uno.removeAll(dos);  
  
System.out.println("Listar despues de quitar la lista DOS");  
// Reiniciar el iterador y listar la lista uno  
aIter = uno.listIterator();  
while (aIter.hasNext()) {  
    System.out.println(aIter.next().verNombre());  
}  
}  
}
```



```
<terminated> ListaEnlazada [Java Application] C:\Progran
```

```
Listar la lista UNO
```

```
Juan
```

```
Maria
```

```
Despues de agregar la lista DOS
```

```
Juan
```

```
Maria
```

```
Jose
```

```
Laura
```

```
Listar despues de quitar la lista DOS
```

```
Juan
```

```
Maria
```





# Diccionarios


---



# Clase HashMap de Java

---

Método	Descripción
put(K clave, V valor)	Reemplaza valor del mapa.
get(Object clave)	Retorna el valor asociado a la clave.
remove(Object clave)	Retira un elemento del mapa.
size()	Retorna el numero de elementos.



```
import java.util.HashMap;
import java.util.Map;
public class PruebaMap {
    public static void main(String[] args) {
        Map<String, String> preferencias = new HashMap<String, String>();
        preferencias.put("color", "rojo");
        preferencias.put("ancho", "640");
        preferencias.put("alto", "480");
        //listar todo
        System.out.println(preferencias);

        // Quitar color
        preferencias.remove("color");

        // cambiar una entrada
        preferencias.put("ancho", " 1024");

        // recuperar un valor
        System.out.println("Alto = " + preferencias.get("alto"));
        System.out.println("Tamaño = " + preferencias.size());

        // iterar por todos los elementos
        for (Map.Entry<String, String> datos : preferencias.entrySet()) {
            String clave = datos.getKey();
            String valor = datos.getValue();
            System.out.println("clave = " + clave + ", valor = " + valor);
        }
    }
}
```



```
<terminated> PruebaMap [Java Application] C:\Pr
```

```
{color=rojo, ancho=640, alto=480}
```

```
Alto = 480
```

```
Tamaño = 2
```

```
clave = ancho, valor = 1024
```

```
clave = alto, valor = 480
```



# Cola de prioridad

---



# Clase PriorityQueue de Java

---

Método	Descripción
add(E e)	Agrega un elemento en la cola.
clear()	Elimina todos los elementos de la cola.
remove()	Elimina un elemento de la cola.
isEmpty()	Devuelve true si la cola está vacía.
size()	Devuelve el número de elementos de la cola.

```
import java.util.PriorityQueue;
import java.util.Queue;
public class ColaPrioridad {
    public static void main(String[] args) {
        Queue<Alumno> cp = new PriorityQueue<Alumno>();
        cp.add(new Alumno("Juan", 11));
        cp.add(new Alumno("Maria", 17));
        cp.add(new Alumno("Jose", 25));
        cp.add(new Alumno("Laura", 8));

        System.out.println("Sacando elementos...");
        Alumno datos;
        while (!cp.isEmpty()) {
            datos = cp.remove();
            System.out.println(datos.verCodigo() + " = " +
                               datos.verNombre());
        }
    }
}
```



```
<terminated> ColaPrioridad [Java Application]
```

```
Sacando elementos...
```

```
8 = Laura
```

```
11 = Juan
```

```
17 = Maria
```

```
25 = Jose
```





# Conjuntos

---



# Clase Set de Java

---

Método	Descripción
add(E elemento)	Agrega un elemento al conjunto.
addAll(coleccion)	Agrega toda una colección.
remove(Object o)	Elimina el objeto del conjunto. Devuelve true si ha sido eliminado.
removeAll(coleccion)	Elimina toda una colección.
isEmpty()	Devuelve true si el conjunto está vacío.
contains(Object o)	Devuelve true si el conjunto contiene el elemento.
size()	Devuelve el número de elementos del conjunto.



```
import java.util.Iterator;
import java.util.Set;
import java.util.HashSet;
public class Conjuntos {
    public static void main(String[] args) {
        Set<String> conjA = new HashSet<String>();
        conjA.add("aaa");
        conjA.add("bbb");
        conjA.add("aaa");
        conjA.add("ccc");
        conjA.add("ddd");
        Set<String> conjB = new HashSet<String>();
        conjB.add("aaa");
        conjB.add("bbb");
        conjB.add("bbb");
        conjB.add("xxx");
        conjB.add("yyy");
        //hallar conjB interseccion conjA
        Set<String> conjC = new HashSet<String>();
        conjC.addAll(conjA);
        // para intersectar A y B, hacemos C=A-B y luego A-C
        conjC.removeAll(conjB);
        conjA.removeAll(conjC);
        //listar
        Iterator<String> iter = conjA.iterator();
        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```



```
<terminated> Conjuntos [Java Application] C:\P
```

```
aaa
```

```
bbb
```



# Arboles

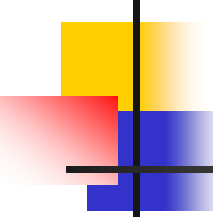
---



# Clase TreeSet de Java

---

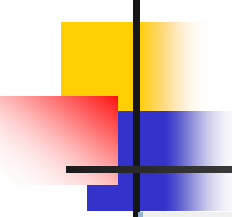
Método	Descripción
add(E elemento)	Agrega un elemento al árbol.
clear()	Borra el árbol.
addAll(coleccion)	Agrega toda una colección.
remove(Object o)	Elimina el objeto del árbol. Devuelve true si ha sido eliminado.
isEmpty()	Devuelve true si el árbol está vacío.
contains(E elemento)	Devuelve true si el conjunto contiene el elemento.
size()	Devuelve el número de elementos del árbol.



```
import java.util.TreeSet;
import java.util.Iterator;
import java.util.Comparator;
public class Arbol {
    public static void main(String[] args) {
        TreeSet<Alumno> alum = new TreeSet<Alumno>();

        alum.add(new Alumno("Juan", 11));
        alum.add(new Alumno("Maria", 17));
        alum.add(new Alumno("Jose", 25));
        alum.add(new Alumno("Laura", 8));
        System.out.println(alum);
        TreeSet<Alumno> ordenarPorNombre = new TreeSet<Alumno>(
            new Comparator<Alumno>() {
                public int compare(Alumno a, Alumno b) {
                    String nombreA = a.verNombre();
                    String nombreB = b.verNombre();
                    return nombreA.compareTo(nombreB);
                }
            });

        ordenarPorNombre.addAll(alum);
        System.out.println(ordenarPorNombre);
        Iterator<Alumno> iter = alum.iterator();
        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```



<terminated> Arbol [Java Application] C:\Program Files (x86)\Java\jre

[8 = Laura, 11 = Juan, 17 = Maria, 25 = Jose]

[25 = Jose, 11 = Juan, 8 = Laura, 17 = Maria]

8 = Laura

11 = Juan

17 = Maria

25 = Jose





# Bibliografía

---

- Fundamentos de Programación, Jorge Teran Pomier, 2006.
- Concursos Internacionales de Informática y Programación, Miguel Revilla, Universidad de Valladolid, 2006.



# Taller de Programación

---

Gracias