

Creación de clases

Crear una clase denominada *Lista* cuyo miembro dato sea un array de números enteros y cuyas funciones miembro realicen las siguientes tareas:

- Hallar y devolver el valor mayor
- Hallar y devolver el valor menor
- Hallar y devolver el valor medio
- Ordenar los números enteros de menor a mayor
- Mostrar la lista ordenada separando los elementos por un tabulador

Definición de la clase *Lista*

Empezamos la definición de la clase escribiendo la palabra reservada **class** y a continuación el nombre de la clase *Lista*.

Los miembros dato

Los miembros dato de la clase *Lista* serán un array de enteros x , y opcionalmente la dimensión del array n .

```
public class Lista {  
    int[] x;        //array de datos  
    int n;          //dimensión
```

El constructor

Al constructor de la clase *Lista* se le pasará un array de enteros para inicializar los miembros dato

```
    public Lista(int[] x) {  
        this.x=x;  
        n=x.length;  
    }
```

Como apreciamos basta una simple asignación para inicializar el miembro dato x que es un array de enteros, con el array de enteros x que se le pasa al constructor. Por otra parte, cuando se le pasa a una función un array se le pasa implícitamente la dimensión del array, que se puede obtener a partir de su miembro dato *length*.

Las funciones miembro

Las funciones miembro tienen acceso a los miembros dato, el array de enteros x y la dimensión del array n .

- El valor medio

Para hallar el valor medio, se suman todos los elementos del array y se divide el resultado por el número de elementos.

```
double valorMedio() {
    int suma=0;
    for(int i=0; i<n; i++){
        suma+=x[i];
    }
    return (double)suma/n;
}
```

Para codificar esta función se ha de tener algunas precauciones. La suma de todos los elementos del array se guarda en la variable local *suma*. Dicha variable local ha de ser inicializada a cero, ya que una variable local contrariamente a lo que sucede a los miembros dato o variables de instancia es inicializada con cualquier valor en el momento en que es declarada.

La división de dos enteros *suma* y *n* (número de elementos del array) es un número entero. Por tanto, se ha de promocionar el entero *suma* de **int** a **double** para efectuar la división y devolver el resultado de esta operación.

- El valor mayor

```
int valorMayor() {
    int mayor=x[0];
    for(int i=1; i<n; i++){
        if(x[i]>mayor) mayor=x[i];
    }
    return mayor;
}
```

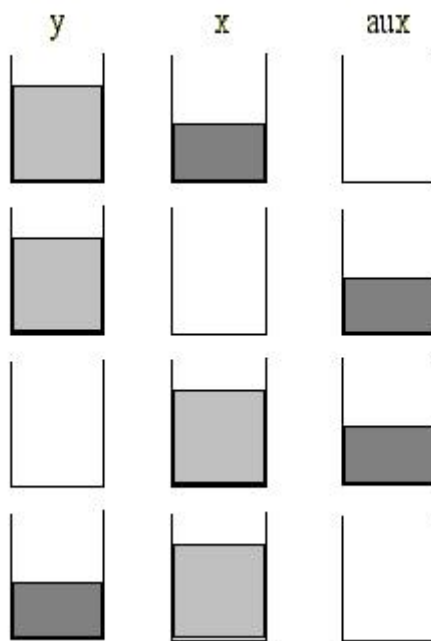
Se compara cada elemento del array con el valor de la variable local *mayor*, que inicialmente tiene el valor del primer elemento del array, si un elemento del array es mayor que dicha variable auxiliar se guarda en ella el valor de dicho elemento del array. Finalmente, se devuelve el valor *mayor* calculado

- El valor menor

```
int valorMenor() {
    int menor=x[0];
    for(int i=1; i<n; i++){
        if(x[i]<menor) menor=x[i];
    }
    return menor;
}
```

El código es similar a la función *valorMayor*

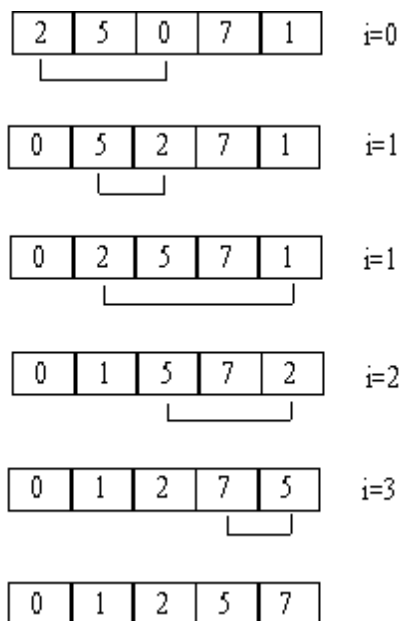
- Ordenar un conjunto de números (este apartado es opcional)



En el proceso de ordenación se ha de intercambiar los valores que guardan elementos del array. Veamos como sería el código correspondiente al intercambio de los valores que guardan dos variables *x* e *y*.

Para intercambiar el contenido de dos recipientes *x* e *y* sin que se mezclen, precisamos de un recipiente auxiliar *aux* vacío. Se vuelca el contenido del recipiente *x* en el recipiente *aux*, el recipiente *y* se vuelca en *x*, y por último, el recipiente *aux* se vuelca en *y*. Al final del proceso, el recipiente *aux* vuelve a estar vacío como al principio. En la figura se esquematiza este proceso.

```
aux=x;
x=y;
y=aux;
```



Para ordenar una lista de números emplearemos el método de la burbuja, un método tan simple como poco eficaz. Se compara el primer elemento, índice 0, con todos los demás elementos de la lista, si el primer elemento es mayor que el elemento *j*, se intercambian sus valores, siguiendo el procedimiento explicado en la figura anterior. Se continúa este procedimiento con todos los elementos del array menos el último. La figura explica de forma gráfica este procedimiento.

```
void ordenar(){
    int aux;
    for(int i=0; i<n-1; i++){
        for(int j=i+1; j<n; j++){
            if(x[i]>x[j]){
                aux=x[j];
                x[j]=x[i];
                x[i]=aux;
            }
        }
    }
}
```

Caben ahora algunas mejoras en el programa, así la función *ordenar* la podemos utilizar para hallar el valor mayor, y el valor menor. Si tenemos una lista ordenada en orden ascendente, el último elemento de la lista será el valor mayor y el primero, el valor menor. De este modo, podemos usar una función en otra funciones, lo que resulta en un ahorro de código, y en un aumento de la legibilidad del programa.

```

int valorMayor() {
    ordenar();
    return x[n-1];
}
int valorMenor() {
    ordenar();
    return x[0];
}

```

- Imprimir la lista ordenada

Imprimimos la lista ordenada separando sus elementos por un tabulador. Primero, se llama a la función *ordenar*, y después se imprime un elemento a continuación del otro mediante *System.out.print*. Recuérdese, que *System.out.println* imprime y a continuación pasa a la siguiente línea.

```

void imprimir() {
    ordenar();
    for(int i=0; i<n; i++){
        System.out.print("\t"+x[i]);
    }
    System.out.println("");
}

```

El código completo de la clase *Lista*, es el siguiente

```

public class Lista {
    int[] x;        //array de datos
    int n;          //dimensión
    public Lista(int[] x) {
        this.x=x;
        n=x.length;
    }
    double valorMedio() {
        int suma=0;
        for(int i=0; i<n; i++){
            suma+=x[i];
        }
        return (double)suma/n;
    }
    int valorMayor() {
        int mayor=x[0];
        for(int i=1; i<n; i++){
            if(x[i]>mayor) mayor=x[i];
        }
        return mayor;
    }
    int valorMenor() {
        int menor=x[0];
        for(int i=1; i<n; i++){
            if(x[i]<menor) menor=x[i];
        }
        return menor;
    }
    void ordenar() {
        int aux;
        for(int i=0; i<n-1; i++){
            for(int j=i+1; j<n; j++){

```

```

                if(x[i]>x[j]){
                    aux=x[j];
                    x[j]=x[i];
                    x[i]=aux;
                }
            }
        }
    }
    void imprimir(){
        ordenar();
        for(int i=0; i<n; i++){
            System.out.print("\t"+x[i]);
        }
        System.out.println("");
    }
}

```

Los objetos de la clase *Lista*

A partir de un array de enteros podemos crear un objeto *lista* de la clase *Lista*.

```

int[] valores={10, -4, 23, 12, 16};
Lista lista=new Lista(valores);

```

Estas dos sentencias las podemos convertir en una

```

Lista lista=new Lista(new int[] {10, -4, 23, 12, 16});

```

En el resto del código, el objeto *lista* llama a las funciones miembro

```

System.out.println("Valor mayor "+lista.valorMayor());
System.out.println("Valor menor "+lista.valorMenor());
System.out.println("Valor medio "+lista.valorMedio());
lista.imprimir();

```

Código de la aplicación

```

public class ListaApp {

    public static void main(String[] args) {
        int[] valores={60, -4, 23, 12, -16};
        Lista lista=new Lista(valores);
        System.out.println("Valor mayor "+lista.valorMayor());
        System.out.println("Valor menor "+lista.valorMenor());
        System.out.println("Valor medio "+lista.valorMedio());
        lista.imprimir();
    }
}

```

Modificadores de acceso

Este ejemplo ilustra una faceta importante de los lenguajes de Programación Orientada a Objetos denominada encapsulación. El acceso a los miembros de una clase está controlado. Para usar una clase, solamente necesitamos saber que funciones miembro se pueden llamar y a qué datos podemos acceder, no necesitamos saber como está hecha la clase, como son sus detalles internos. Una vez que la clase está depurada y probada, la clase es como una caja negra. Los objetos de dicha clase guardan unos datos, y están caracterizados por una determinada conducta. Este ocultamiento de la información niega a las entidades exteriores el acceso a los miembros privados de un objeto. De este modo, las entidades exteriores acceden a los datos de una manera controlada a través de algunas funciones miembro. Para acceder a un miembro público (dato o función) basta escribir.

```
objeto_de_la_clase_Fraccion.miembro_público_no_estático  
clase_Fraccion.miembro_público_estático
```

Delante de los miembros dato, como podemos ver en el listado hemos puesto las palabras reservadas **public** y **private**.

- Miembros públicos

Los miembros públicos son aquellos que tienen delante la palabra **public**, y se puede acceder a ellos sin ninguna restricción.

- Miembros privados

Los miembros privados son aquellos que tienen delante la palabra **private**, y se puede acceder a ellos solamente dentro del ámbito de la clase.

- Por defecto (a nivel de paquete)

Cuando no se pone ningún modificador de acceso delante de los miembros, se dice que son accesibles dentro del mismo paquete ([package](#)). Esto es lo que hemos hecho en los ejemplos estudiados hasta esta sección.

package es la primera sentencia que se pone en un archivo .java. El nombre del paquete es el mismo que el nombre del subdirectorío que contiene los archivos .java. Cada archivo .java contiene habitualmente una clase. Si tiene más de una solamente una de ellas es pública. El nombre de dicha clase coincide con el nombre del archivo.

Como el lector se habrá dado cuenta hay una correspondencia entre archivos y clases, entre paquetes y subdirectoríos. El Entorno Integrado de Desarrollo (IDE) en el que creamos los programas facilita esta tarea sin que el usuario se aperciba de ello.

Mejora de la clase *Lista* (opcional)

Veamos un ejemplo más, la clase [*Lista*](#), y hagamos uso de la función miembro *ordenar* para hallar el *valorMenor* y el *valorMayor*. Si tenemos una lista ordenada en orden creciente, el valor menor es el primer elemento de la lista $x[0]$, y el valor mayor es el último elemento de la lista $x[n-1]$. Podemos escribir el siguiente código

```
public int valorMayor() {
    ordenar();
    return x[n-1];
}
public int valorMenor() {
    ordenar();
    return x[0];
}
```

Podemos llamar una sóla vez a la función miembro *ordenar* en el constructor, después de haber creado el array, y evitar así la reiteración de llamadas a dicha función en *valorMayor*, *valorMenor* e *imprimir*.

```
public Lista(int[] x) {
    this.x=x;
    n=x.length;
    ordenar();
}
public int valorMayor() {
    return x[n-1];
}
public int valorMenor() {
    return x[0];
}
```

La función miembro *ordenar*, es una función auxiliar de las otras funciones miembro públicas, por tanto, podemos ponerle delante el modificador de acceso **private**. El usuario solamente está interesado en el valor medio, el valor mayor y menor de un conjunto de datos, pero no está interesado en el procedimiento que permite *ordenar* el conjunto de datos. Como ocurre en la vida moderna usamos muchos aparatos pero no tenemos por que conocer sus detalles internos y cómo funcionan por dentro. Una clase es como uno de estos aparatos modernos, el usuario solamente tiene que conocer qué hace la clase, a qué miembros tiene acceso, pero no como está implementada en software.

La función miembro *toString*

Si los miembros dato de la clase *Lista* son privados (**private**) hemos de definir una función que hemos denominado *imprimir* para mostrar los valores que guardan los miembros dato de los objetos de la clase *Lista*.

```
public class Lista {
    private int[] x;        //array de datos
    private int n;          //dimensión
    //...
```

```

public void imprimir(){
    for(int i=0; i<n; i++){
        System.out.print("\t"+x[i]);
    }
    System.out.println("");
}

```

La llamada a esta función miembro se efectúa desde un objeto de la clase *Lista*

```

Lista lista=new Lista(new int[]{60, -4, 23, 12, -16});
System.out.println("Mostrar la lista");
lista.imprimir();

```

Sustituimos la función miembro *imprimir* por la redefinición de *toString*. Para redefinir una función, tiene que tener el mismo nombre, los mismos modificadores, el mismo tipo de retorno y los mismos parámetros y del mismo tipo en la clase base y en la clase derivada. Para evitar errores, el mejor procedimiento es el de ir al código de la clase base *Object*, copiar la línea de la declaración de *toString*, pegarla en la definición de nuestra clase y a continuación, definir dicha función.

```

public class Lista {
    private int[] x;        //array de datos
    private int n;          //dimensión
    //...
    public String toString(){
        String texto="";
        for(int i=0; i<n; i++){
            texto+="\t"+x[i];
        }
        return texto;
    }
}

```

La llamada a la función *toString* se realiza implícitamente en el argumento de la función *System.out.println*, o bien, al concatenar un string y un objeto de la clase *Lista*.

```

Lista lista=new Lista(new int[]{60, -4, 23, 12, -16});
System.out.println("Mostrar la lista");
System.out.println(lista);

```

```

public class Lista {
    private int[] x;        //array de datos
    private int n;          //dimensión
    public Lista(int[] x) {
        this.x=x;
        n=x.length;
        ordenar();
    }
    public double valorMedio(){
        int suma=0;
        for(int i=0; i<n; i++){
            suma+=x[i];
        }
        return (double)suma/n;
    }
    public int valorMayor(){
        return x[n-1];
    }
}

```

```
public int valorMenor(){
    return x[0];
}
private void ordenar(){
    int aux;
    for(int i=0; i<n-1; i++){
        for(int j=i+1; j<n; j++){
            if(x[i]>x[j]){
                aux=x[j];
                x[j]=x[i];
                x[i]=aux;
            }
        }
    }
}
public String toString(){
    String texto="";
    for(int i=0; i<n; i++){
        texto+="\t"+x[i];
    }
    return texto;
}
}
```