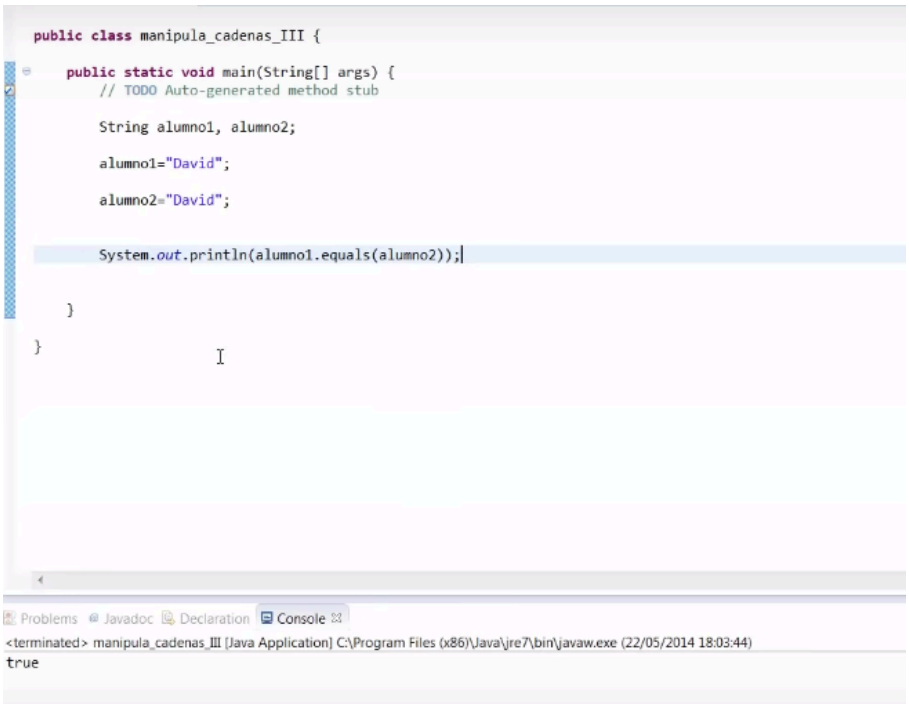
Comencemos haciendo un breve repaso de lo realizado en la sesión anterior:
Vamos a hacer un ejemplo para concatenar un texto con un substring

```
public class manipula_cadenas_II {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        String frase="Hoy es un estupendo día para aprender a programar en Java";

        String frase_resumen=frase.substring(0, 29) + " irnos a la playa y olvidarnos de todo...." + " y "
         frase.substring(29, 57);

        System.out.println(frase_resumen);
    }

}
```

Problems @ Javadoc Declaration Console

```
<terminated> manipula_cadenas_II [Java Application] C:\Program Files (x86)\Java\jre7\bin\javaw.exe (22/05/2014 18:00:54)
Hoy es un estupendo día para  irnos a la playa y olvidarnos de todo.... y aprender a programar en Java
```

corrijamos el ejercicio propuesto de comparar dos cadenas con el método .equals(cadena) devuelve un valor booleano, de la comparación de dos string
Partimos de dos variables de tipo String, y las iniciamos, ambas tendrán el mismo valor (David)
Le aplicamos un valor

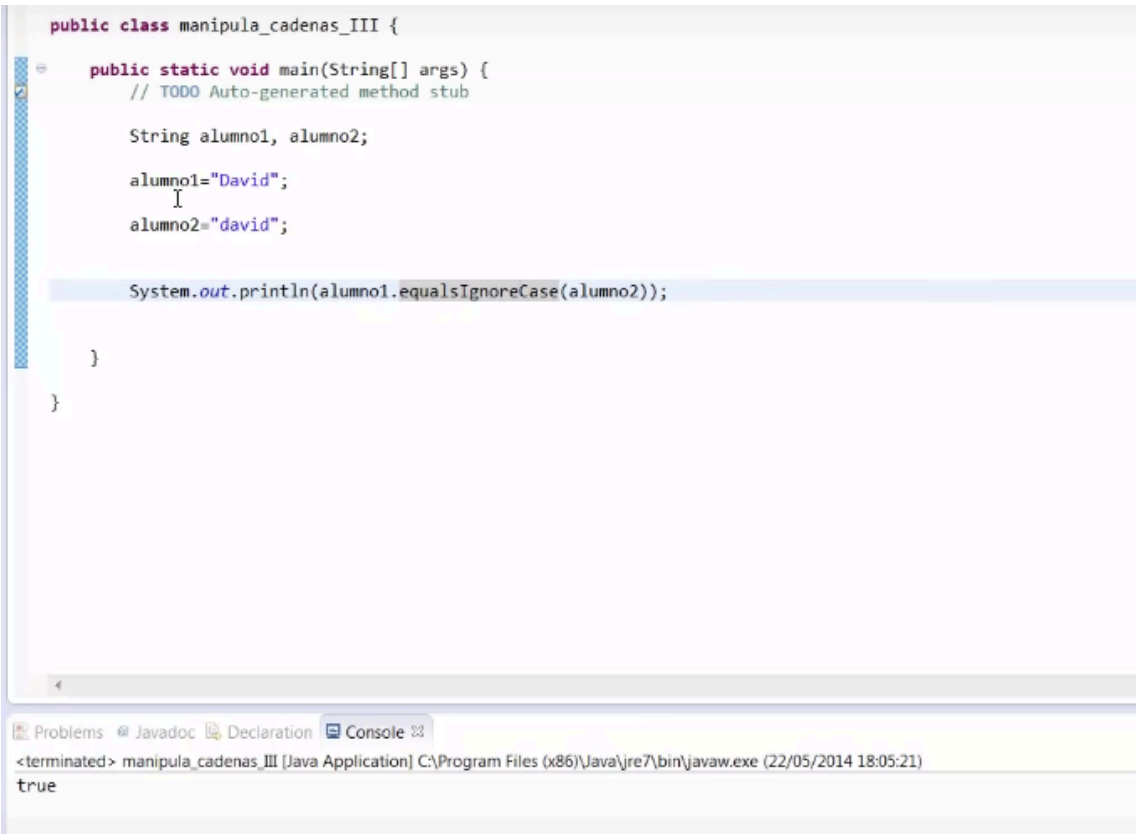


Si le aplicamos el equals, nos dará true porque los dos están exactamente igual escritos, pero en el momento en que cambiemos una letra, de uno de ellos, de may a min, ya no coincidirá por lo que devolverá **false**

EJERCICIO DE .equalsIgnoreCase()

Hace comparativa entre las dos cadenas pero ignorando si son mayúculas o minúsculas
ojo tener en cuenta que cuando es una palabra reservada del lenguaje JAVA, compuesto por varias palabras, se utiliza el CamelCase, cada palabra se pone en may la primera letra de cada palabra
En el ejemplo se trata de las mismas palabras escritas diferente. En este caso devuelve true. Lógicamente si ponemos palabras diferentes siempre devolverá

Hace comparativa entre las dos cadenas pero ignorando si son mayúculas o minúsculas
ojo tener en cuenta que cuando es una palabra reservada del lenguaje JAVA, compuesto por varias palabras, se utiliza el CamelCase, cada palabra se pone en may la primera letra de cada palabra
En el ejemplo se trata de las mismas palabras escritas diferente. En este caso devuelve true. Lógicamente si ponemos palabras diferentes siempre devolverá false



2. Uso estructuras de control

2.1. Estructuras de selección

2.1.1 Estructura If.

Permite ejecutar una instrucción (o secuencia de instrucciones) si se da una condición.

La sentencia if es la sentencia básica de selección.

Existen tres variantes, selección simple, selección doble y selección múltiple, pero las 3 se basan en la misma idea.

a) Selección Simple. Su sintaxis es la siguiente:

if (condición){
sentencias; }

Donde: condición es una expresión booleana sentencias representa un bloque de sentencias o una sentencia única.

Si es una sentencia única, se pueden quitar las llaves

Ejemplo:

```
if (numero%2!=0){
System.out.println("El numero es impar ");
cont++;
```

}

b) Selección Doble. Se añade una parte **else** a la sentencia **if** la cual se
ejecutara si la condición es falsa. Su sintaxis es:

```
if (condición){
sentencias1;
}
else{
sentencias2;
}
```

Ejemplo:

```
if (a>b)
System.out.println ("El número mayor es " + a);
else
System.out.println("El número mayor es " + b+ " o son iguales");
```

c) Selección Múltiple. Es habitual cuando hay más de una condición. Su sintaxis es la siguiente:

```
if (condición1){
sentencias1;
 }
else if (condición2){
 sentencias2;
} else if (condición3){
sentencias3;
else{
sentencias;
}
```

Ejemplo:

```
if (mes == 12 || mes == 1 || mes == 2)
System.out.println ( "Invierno");
else if (mes == 3 || mes == 4 || mes == 5)
System.out.println( "Primavera");
else if (mes == 6 || mes == 7 || mes == 8)
System.out.println ("Verano");
else
System.out.println ("Otoño");
```

2.1.2 Switch.

Es una estructura de selección múltiple más cómoda de leer y utilizar que
el else-if. Selecciona un bloque de sentencias dependiendo del valor de una
expresión. Su sintaxis es:

```
switch (expresion) {
case valor1:sentencias;
break;
case valor2:sentencias;
break;
case valor3:sentencias;
break;
...
default: sentencias;
break;
}
```

Donde: Expresión tiene que tomar un valor entero o un carácter. Break indica que ha acabado la ejecución de ese caso y seguiría ejecutando la sentencia siguiente al switch. Default es opcional y se ejecutara cuando la expresión tome un valor que no esté recogido en los distintos casos especificados.

Ejemplo:

```
 switch (mes)
{ case 4:
case 6:
case 9:
case 11: dias = 30;
break;
case 2: dias = 28;
break; default:
dias = 31; break;
}
```

Es posible juntar distintos casos, dejándolos en blanco y especificando las instrucciones en el último de los casos de grupo.
Por ejemplo, si la variable mes toma los valores 4, 6, 9 y 11 se realiza la misma sentencia, se le asigna a la variable día el valor 30

2.2. Estructuras de repetición

Estas estructuras se utilizan para repetir un bloque de sentencias mientras se cumpla una condición.
Son también llamadas sentencias de iteración o bucles.
Los tipos de bucles son: while, do-while y for .

2.2.1 Bucle While:

La sintaxis de la sentencia es:
while (condición){
sentencias;
}
Donde:
condición es una expresión booleana que se evalúa al principio del bucle y antes de cada iteración de las sentencias.
Si la condición es verdadera, se ejecuta el bloque de sentencias, y se vuelve al principio del bucle.
Si la condición es falsa, no se ejecuta el bloque de sentencias, y se continúa con la siguiente sentencia del programa.
Si la condición es falsa desde un principio, entonces el bucle nunca se ejecuta.
Si la condición nunca llega a ser falsa, se tiene un bucle infinito.
Ejemplo:

```
int i = 0;
while (i < 10) {
System.out.println (i);
 i++;
}
```

2.2.2 Bucle Do-While:

La sintaxis de la sentencia es:

```
do {
sentencias;
} while (condición);
```

La sentencia es muy parecida a while. El bloque de sentencias se repite mientras se cumpla la condición.
La condición se comprueba después de ejecutar el bloque de sentencias.
El bloque se ejecuta siempre al menos una vez.
Ejemplo:

```
int i=0;
do{
System.out.println (i);
i++;
}while (i<10);
```

2.2.3 Bucle For:

La sintaxis de la sentencia es:
for (inicialización;condición; incremento){
sentencias;
}
Donde
La inicialización se realiza sólo una vez, antes de la primera iteración.
La condición se comprueba cada vez antes de entrar al bucle. Si es cierta, se entra. Si no, se termina.
El incremento se realiza siempre al terminar de ejecutar la iteración, antes de volver a comprobar la condición.
El incremento puede ser positivo o negativo.
Ejemplo:

```
int i;
for (i=0; i<10; i++)
System.out.println (i);
```

3. Control de excepciones

Una excepción es un evento que ocurre durante la ejecución de un programa y detiene el flujo normal de la secuencia de instrucciones del programa.
El control de dichas excepciones se utiliza para la detección y corrección de errores. Si hay un error, la aplicación no debería "morirse".
Para manejar las excepciones en Java, se actúa de la siguiente manera: Se intenta **(try)** ejecutar la sentencia o bloque de sentencias que pueden producir algún error.
Se captura **(catch)** las posibles excepciones que se hayan podido producir, ejecutando una serie de sentencias que informen o intenten resolver el error.
Finalmente (**finally)** se puede ejecutar una serie de sentencias tanto si se ha producido un error como si todo ha ido bien.

El formato es:

```
try { Sentencias que pueden producir error
}catch(ClaseExcepción variableRecogeExcepción){ Sentencias
que informan o procuran solucionar
el error.
```

La variable variableRecogeExcepción no se
Tiene que declarar antes.
**}catch(ClaseExcepción2 variableRecogeExcepción2){ Puede haber varios catch. Uno para cada tipo
de Exception.**
}finally { Sentencias que deben ejecutarse en cualquier caso (opcional) }
El elemento ClaseExcepción que aparece junto a catch, debe ser una de las clases de excepción. Al generarse el error durante la ejecución se puede comprobar qué clase de excepción se ha producido.

De forma general, **la clase Exception** recoge todos los tipos de excepciones. Si se desea un control más exhaustivo del tipo de error que se produce, se debe concretar la clase de excepción correspondiente.
Por ejemplo, cuando se intenta convertir al tipo de dato numérico entero un dato introducido por el usuario en un campo de texto se utiliza una sentencia como:
int num = Integer.valueOf(campoNúmero.getText());

Si el valor introducido no es numérico, sino una cadena de caracteres, la llamada a Integer.valueOf produce una excepción, como se puede apreciar en la salida estándar:

```
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "hola"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:48)
        at java.lang.Integer.parseInt(Integer.java:447)
        at java.lang.Integer.valueOf(Integer.java:553)
        at ejemplostema3.EjemploTry.botónAceptarActionPerformed(EjemploTry.java:74)
        at ejemplostema3.EjemploTry.access$000(EjemploTry.java:20)
        at ejemplostema3.EjemploTry$1.actionPerformed(EjemploTry.java:44)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:1995)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2318)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:387)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:242)
        at javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:236)
        at java.awt.Component.processMouseEvent(Component.java:6134)
        at javax.swing.JComponent.processMouseEvent(JComponent.java:3265)
```

Se puede apreciar que se produce una excepción del tipo NumberFormatException, por tanto se debería capturar esa excepción para controlar el error.

```
try {
int num = Integer.valueOf(campoNúmero.getText());
} catch(NumberFormatException e) {
System.out.println("Error: El valor indicado no es un número");
}
```

En caso de no tener claro el tipo de excepción que se quiere controlar, o se quiere controlar cualquier tipo de excepción que se pueda producir, se indica en el catch la **clase Exception** que es genérica para todas las excepciones.

```
try {
int num = Integer.valueOf(campoNúmero.getText());
} catch(Exception e) {
System.out.println("Se ha detectado un error");
}
```

La variable que se indique en catch(), recoge información sobre la excepción que se ha producido. Es muy frecuente que a dicha variable se le asigne el nombre e.

Si se desea mostrar qué tipo de error se ha producido se puede utilizar el ***método getMessage*** o el ***método toString*** (similar a getMessage pero además incluye el nombre de la clase de excepción) sobre la variable que se ha utilizado en el catch:

```
System.out.println(e.getMessage());
System.out.println(e.toString());
```

Ejemplo:

```
 int dividendo=10, divisor=0, cociente;
try { cociente = dividendo/divisor; } catch (ArithmeticException e) {
 System.out.println("Error: Division por 0"); cociente = 0; }
// en cualquier caso el programa continúa por aquí
```

4. Clase y objetos

4.1 clase

Una clase es un tipo definido por el usuario que describe los atributos y los métodos de los objetos que se crearán a partir de ella.
El estado del *objeto* viene determinado por los *atributos* y los *métodos* son las operaciones que definen su comportamiento.
Dentro de las clases también se encuentran los *constructores* que permiten inicializar un objeto.
Los atributos y los métodos se denominan en general miembros de la clase.

La definición de una clase consta de dos partes: el nombre de la clase precedido por la palabra reservada class y el cuerpo de la clase entre llaves.

La sintaxis queda:
class nombre-clase{
cuerpo de la clase
}
Dentro del cuerpo de la clase se puede encontrar atributos y métodos.

```
Ej: class Circunferencia{
private double x,y, radio;
public Circunferencia(){ }
public Circunferencia(double cx, double cy, double r){
x=cx; y=cy; radio=r;
}
public void ponRadio(double r){
radio=r;
}

public double longitud(){
return 2*Math.PI*radio;
}
}
```

En el ejemplo se define la clase Circunferencia, que puede ser usada dentro de un programa de la misma manera que cualquier otro tipo. Un objeto de esta clase tiene tres atributos (coordenadas del centro y valor del radio), dos constructores y un método.
Los constructores se distinguen fácilmente porque tienen el mismo nombre que la clase.
Los atributos se declaran de la misma manera que cualquier variable. En una clase, cada atributo debe tener un nombre único.
Siguiendo las recomendaciones de la programación orientada a objetos, cada clase se debe implementar en un fichero .java, de esta manera es más sencillo modificar la clase.

4. 2. Métodos

Los métodos forman lo que se denomina interfaz de los objetos, definen las operaciones que se pueden realizar con los atributos. Desde el punto de vista de la Programación Orientada a Objetos, el conjunto de métodos se corresponde con el conjunto de mensajes que los objetos de una clase pueden responder.
Los métodos permiten al programador modularizar sus programas.
Todas las variables declaradas en las definiciones de métodos son variables locales, sólo se conocen en el método que las define. Casi todos los métodos tienen una lista de parámetros que permiten comunicar información

La sintaxis para definir un método es:
<modificador-acceso> tipoR Nombre-método(<parámetros>){ <cuerpodelmétodo> }

donde <modificador-acceso> indica como es el acceso a dicho método: public, private, protected y sin modificador.
<tipoR> es el tipo de dato que retorna el método. Es obligatorio indicar un tipo. Para los métodos que no devuelven nada se utiliza la palabra *reservada void*.
Nombre-método es como el programador quiere llamar a su método y <parámetros> son los datos que se van a enviar al método para trabajar con ellos, no son obligatorios.
Para retornar el valor se utiliza el operador **return**.

Ejemplo:
```
int Suma(int x, int y){
return x+y;
}
```

En este ejemplo el método retorna un valor de tipo int, recibe 2 parámetros también de tipo int y realiza su suma.
```
void Imprimir(){
System.out.println("Este método no devuelve nada y tampoco recibe parámetros");
}
```
En este ejemplo el método no retorna ningún valor, ni recibe parámetros, aun así es necesario poner los paréntesis vacíos.

4.2.1 Métodos Estáticos o de Clase

Se cargan en memoria en tiempo de compilación y no a medida que se ejecutan las líneas de código del programa. Van precedidos del modificador static.
Para invocar a un método estático no se necesita crear un objeto de la clase en la que se define. Si se invoca desde la clase en la que se encuentra definido, basta con escribir su nombre.
Si se le invoca desde una clase distinta, debe anteponerse a su nombre, el de la clase en que se encuentra seguido del operador punto (.)
La sintaxis es:
< NombreClase> .metodoEstatico( );

Suelen emplearse para realizar operaciones comunes a todos los objetos de la clase. No afectan a los estados de los mismos.

Por **ejemplo**, si se necesita un método para contabilizar el número de objetos creados de una clase, se define estático ya que su función, aumentar el valor de una variable entera, se realiza independientemente del objeto empleado para invocarle.
No es conveniente usar muchos métodos estáticos, pues si bien se aumenta la rapidez de ejecución, se pierde flexibilidad, no se hace un uso efectivo de la memoria y no se trabaja según los principios de la POO.
A veces es necesario crear un método que se utiliza fuera del contexto de cualquier instancia, para ello hay que declarar estos métodos como static. Los métodos estáticos sólo pueden llamar a otros métodos static directamente, y no se pueden referir a *this* o super de ninguna manera.

Las variables también se pueden declarar como static,
y es equivalente a declararlas como variables globales, que son accesibles desde cualquier fragmento de código.

Se puede declarar un bloque static que se ejecuta una sola vez si se necesitan realizar cálculos para inializar las variables static.

Ejemplo:

```
class Estatica{
static int a=3,b;
static{ System.out.println("Bloque static inicializado");
b=a*4;
}
static void metodo2(int x){
System.out.println("x= "+x);
System.out.println("a= "+a);
System.out.println("b= "+b);
} public static void main(String[] args){ metodo2(42);
}
}
```

En el ejemplo la clase que tiene dos variables static, un bloque de inicialización static y un método static. La salida del programa es:

```
Bloque static inicializado
x = 42
a = 3
b = 12
```

5. OBJETOS

Un objeto consta de :

- una estructura interna (los atributos) y de
- una interfaz que permite acceder y manipular dicha estructura (los métodos).

Para construir un objeto de una clase cualquiera hay que llamar a un método de iniciación, el constructor. Para ello se utiliza el operador **new**. La sintaxis es la siguiente:
Nombre-clase nombre-objeto=new Nombre-clase(<valores>);
donde *Nombre-clase* es el nombre de la clase de la cual se quiere crear el objeto, *nombre-objeto* es el nombre que el programador quiere dar a ese objeto y <*valores*> son los valores con los que se inicializa el objeto. Dichos valores son opcionales.
Ej: Circunferencia circ = new Circunferencia();
Se crea el objeto circ, de la clase Circunferencia, con los valores predeterminados.
Cuando se crea un objeto, Java hace lo siguiente:

- Asignar memoria al objeto por medio del operador **new**.
- Llamar al constructor de la clase para inicializar los atributos de ese objeto con los valores iniciales o con los valores predeterminados por el sistema: los atributos numéricos a cero, los alfanuméricos a nulos y las referencias a objetos a null.

Si no hay suficiente memoria para ubicar el objeto, el operador new lanza una excepción *OutOfMemoryError*.
Para acceder desde un método de una clase a un miembro de un objeto de otra clase diferente se utiliza la sintaxis: *objeto.miembro*. Cuando el miembro accedido es un método se entiende que el objeto ha recibido un mensaje, el especificado por el nombre del método y responde ejecutando ese método.

5. 1 Asignación

Una vez el objeto está creado, se tiene una referencia a ese objeto. Si se realiza la asignación de un objeto a otro los dos harán referencia al mismo objeto.

Ej: Circunferencia c1 = new Circunferencia(0,0,15);
Circunferencia c2 = c1;
c1.ponRadio(25);
System.out.println(c2.radio);

En el ejemplo la variable c1 apunta a un objeto de la clase Circunferencia. Debido a la asignación, la variable c2 apunta al mismo objeto. Después se modifica el valor del radio del objeto apuntado por c1. Y por último, se visualiza c2 que será el valor 25.

5.2 Igualdad

Creamos dos objetos iguales (con los mismos valores de sus variables miembro), al preguntar si las variables que apuntan a esos objetos son iguales nos devolverá false, pues aunque tengan el mismo valor no son el mismo objeto.

Ej: Circunferencia c1 = new Circunferencia(0,0,15);
Circunferencia c2 = new Circunferencia(0,0,15);
if (c1==c2) **/* Esto es falso*/**

6 CONSTRUCTOR

Un Constructor es un método especial en Java empleado para inicializar valores en instancias de objetos.
A través de este tipo de métodos es posible generar diversos tipos de instancias para la clase en cuestión.
Los métodos constructores tienen las siguientes características:

- Se llaman igual que la clase.
- No devuelve nada, ni siquiera void.
- Puede haber varios constructores, que deberán distinguirse por el tipo de valores que reciba.
- De entre los que existan, sólo uno se ejecutará al crear el objeto.
- El código de un constructor, generalmente, suele ser inicializaciones de variables y objetos,

para conseguir que el objeto sea creado con dichos valores iniciales.

Si no se define ningún constructor el compilador crea uno por defecto sin parámetros que, al ejecutarse, inicializa el valor de cada atributo de la nueva instancia a 0, false o null, dependiendo de si el atributo es numérico, alfanumérico o una referencia a otro objeto respectivamente, pero dicho constructor desaparece en el mismo momento en que se defina otro constructor, por lo que si se quiere tener el de por defecto habrá que definirlo.

La declaración de constructores sigue la siguiente sintaxis:
<modificadordeVisibilidad> NombredelaClase (<argumentos>) {
< declaraciones>
}
donde *<modificadorvisibilidad>* es el tipo de modificador de acceso del constructor, *<Nombredelaclase>* es el nombre del constructor y debe coincidir con el de la clase y *<argumentos>* son las variables que recibe el constructor y contienen los valores con los que se inicializaran los atributos.

Ej: class Circunferencia{
private double x,y, radio;
public Circunferencia(){ }
public Circunferencia(double cx, double cy, double r){
x=cx; y=cy; radio=r;
}
……..
}
En este ejemplo existen 2 constructores, el primero que es el de por defecto y el segundo que inicializa los atributos x, y y radio con los valores cx, cy y r, respectivamente.

7 **REFERENCIA THIS**

Java incluye un valor de referencia especial llamado *this*, que se utiliza dentro de cualquier método para referirse al objeto actual. El valor *this* se refiere al objeto sobre el que ha sido llamado el método actual.
Se puede utilizar *this* siempre que se requiera una referencia a un objeto del tipo de una clase actual. Si hay dos objetos que utilicen el mismo código, seleccionados a través de otras instancias, cada uno tiene su propio valor único de *this*.
Normalmente, dentro del cuerpo de un método de un objeto se puede referir directamente a las variables miembros del objeto. Sin embargo, algunas veces no se querrá tener ambigüedad sobre el nombre de la variable miembro y uno de los argumentos del método que tengan el mismo nombre.
Ej: class Circunferencia{
private double x,y, radio;
public Circunferencia(double x, double y, double radio){
this.x=x; this.y=y; this.radio=radio;
}
En el ejemplo el constructor de la clase inicializa las variables con los argumentos pasados al constructor.
Se debe utilizar *this* en este constructor para evitar la ambigüedad entre el los argumentos y las variables miembro.
También se puede utilizar *this* para llamar a uno de los métodos del objeto actual. Esto sólo es necesario si existe alguna ambigüedad con el nombre del método y se utiliza para intentar hacer el código más claro.

8 ARRAYS

Conceptos básicos

Un arreglo unidimensional es un grupo de valores (llamados elementos o componentes) que son del mismo tipo. Los arreglos son objetos, por lo que se consideran como tipos de referencia.

Los elementos de un arreglo pueden ser tipos primitivos o de referencias (incluyendo arreglos). Para hacer referencia a un elemento específico en un arreglo se debe especificar el nombre del arreglo y el número de la posición del elemento en el arreglo.
El número de posición del elemento se conoce formalmente como el índice o subíndice del elemento en el arreglo. Todos los elementos de un arreglo deben ser del mismo tipo.

Declaración y creación de arreglos Unidimensionales

Para crear un arreglo, hay que crear una variable de arreglo del tipo deseado.
La sintaxis general de un arreglo unidimensional es:
Tipo nombre-de-variable[ ] ;

Los objetos arreglo ocupan espacio en memoria. Todos los objetos deben de crearse con la palabra clave **new**.
El programador especifica el tipo de cada elemento y el número de elementos que se requieren para el arreglo.

La siguiente declaración crea 12 elementos para el arreglo de enteros de nombre c:
int c[ ] ; / / declara la variable arreglo
c = new int [12] ; / / crea el arreglo

Esta tarea también puede realizarse en un paso:
int c[ ] = new int[12] ;

Al crear un arreglo cada uno de sus elementos recibe un valor predeterminado de cero para los elementos numéricos de tipos primitivos, falso para los elementos boléanos y nulo para las referencias (cualquier tipo no primitivo).

Al declararse un arreglo, su tipo y los corchetes pueden combinarse al principio de las declaraciones para indicar que todos los identificadores en la declaración son referencias a arreglos.

Ejemplo: double[ ] arreglo1, arreglo2;
arreglo1 = new double[10] , arreglo2 = new double[20] ;

Un programa puede declarar arreglos de cualquier tipo.

- Todo elemento de un arreglo de tipo primitivo es una variable del tipo declarado del arreglo.
- En un arreglo que sea de tipo de referencia, cada elemento del arreglo es una referencia a un objeto del tipo declarado de ese arreglo.
- Cada uno de los elementos de un arreglo String es una referencia a un objeto String.

Un programa puede hacer referencia a cualquiera de estos elementos mediante una expresión de acceso a un arreglo que incluye el nombre del arreglo, seguido por el índice del elemento especifico encerrado entre ([]) corchetes.
El primer elemento en cualquier arreglo tiene el índice cero (lo que se denomina como elemento cero). Por lo tanto, el primer elemento del arreglo c es c[0], el segundo elemento es c[1], el séptimo elemento del arreglo es c[6] y, en general, el i-esimo elemento del arreglo c es[c-i].
Los nombres de los arreglos siguen las mismas convenciones, que los demás nombres de las variables.
Un índice debe ser un entero positivo o una expresión entera que pueda promoverse a un **int**. Si un programa utiliza una expresión como índice, el programa evalúa la expresión para determinar el índice.
Ej.: Supóngase que la variable a es 5 y que b es 6, entonces la instrucción: C [a + b] += 2; suma 2 al elemento c[11] del arreglo.
El nombre del arreglo con subíndice es una expresión de acceso al arreglo. Dichas expresiones pueden utilizarse en el lado izquierdo de una asignación, para colocar un nuevo valor en un elemento del arreglo.

Operaciones
Para realizar cualquier operación con un arreglo, normalmente, se usa un for que comenzara en 0 y terminara en n-1, donde n es el número de elementos del array.

```
Ej: int c[ ] = new int[12] ;
for (int i=0; i<12;i++)
c[i]= (int) (Math.random()*10)+1; // Crea de manera aleatoria un array de 12 componentes
for (int i=0; i<12;i++)
System.out.print(c[i]+" "); // Escribe en pantalla el contenido de las componentes
```

3.6.3 Arreglos Bidimensionales

Se define como un conjunto de datos del mismo tipo organizados en filas y en columnas, es decir, en una matriz o tabla.

Los arreglos con dos dimensiones se utilizan a menudo para representar tablas de valores, que constan de información ordenada en filas y columnas.

El primer índice indica la fila y el segundo la columna. Para identificar un elemento de una tabla, se deben especificar los dos índices.

```
Ej:  int[ ] [ ] M = {{1,2,3,4},{5,6,7,8}{9,10,11,12}};
     for( int i=0;i<3;i++){
        for(int j=0;j<4;j++)
           System.out.print(M[i][j]+ " " );
        System.out.println();
     }
```

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

PRACTICA ARRAY

Un array, o arreglo es una colección finita de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.
Ej: Se quieren guardar las notas de los 20 alumnos de una clase.
Gráficamente el array se puede representar de la siguiente forma:
Arraynotas:

8.50	6.35	5.75	8.50	· · ·	3.75	6.00	7.40
notas[0]	notas[1]	notas[2]	notas[3]	· · ·	notas[17]	notas[18]	notas[19]

Para acceder a cada elemento del array se utiliza el nombre del array y un índice que indica la posición que ocupa el elemento dentro del array. El índice se escribe entre corchetes.

El primer elemento del array ocupa la posición 0, el segundo la posición 1, etc. En un array de N elementos el último ocupará la posición N-1.
En el ejemplo, notas[0] contiene la nota del primer alumno y notas[19] contiene la del último
Los índices deben ser enteros no negativos.

Crear arrays unidimensionales
Para crear un array se deben realizar dos operaciones:
1. Declaración
2. Instanciación

1. Declarar de un array: En la declaración se crea la referencia al array.
La referencia es el nombre del array en el programa. Se debe indicar el nombre del array y el tipo de datos que contendrá.

De forma general un array unidimensional se puede declarar de cualquiera de estas dos formas:
tipo [] nombreArray; o tipo nombreArray[];

tipo: indica el tipo de datos que contendrá. Un array puede contener elementos de tipo básico o referencias a objetos.
nombreArray: es la referencia al array.

Ej: int [] ventas; //array de tipo int llamado ventas
double [] temperaturas; //array de tipo double llamado temperaturas
String [] nombres; //array de tipo String llamado nombres

2. Instanciación

Instanciar un array: Mediante la instanciación se reserva un bloque de memoria para almacenar todos los elementos del array.
La dirección, donde comienza el bloque de memoria, donde se almacenará el array se asigna al nombre.

De forma general:

nombreArray = new tipo[tamaño];

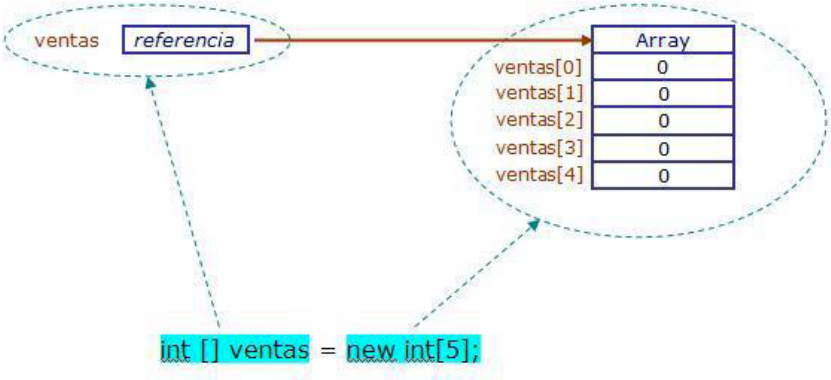
nombreArray: es el nombre creado en la declaración.
tipo: indica el tipo de datos que contiene.
tamaño: es el número de elementos del array. Debe ser una expresión entera positiva. El tamaño no se puede modificar durante la ejecución del programa.
new: operador para crear objetos. Mediante new se asigna la memoria necesaria para ubicar el objeto. Java implementa los arrays como objetos.

Ejemplo:
ventas = new int[5]; //se reserva memoria para 5 enteros.

Lo normal es que la declaración y la instanciación se hagan en una sola instrucción: *tipo [] nombreArray = new tipo[tamaño];*
Ejemplo:
int [] ventas = new int[5];

El tamaño del array también se puede indicar durante la ejecución del programa, es decir, en tiempo de ejecución se puede pedir por teclado el tamaño del array y crearlo.

Ejemplo:
Scanner teclado = new Scanner(System.in);
System.out.print("Número de elementos del array: ");
int numeroElementos = teclado.nextInt();
int [] ventas = new int[numeroElementos];
Si no hay memoria suficiente para crear el array, new lanza una excepción java.lang.OutOfMemoryError.



Inicializar arrays unidimensionales
Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial.

Los valores iniciales por defecto para un array en java son:
- 0 para arrays numéricos
- '\u0000' (carácter nulo) para arrays de caracteres
- false para arrays booleanos
- null para arrays de String y de referencias a objetos.

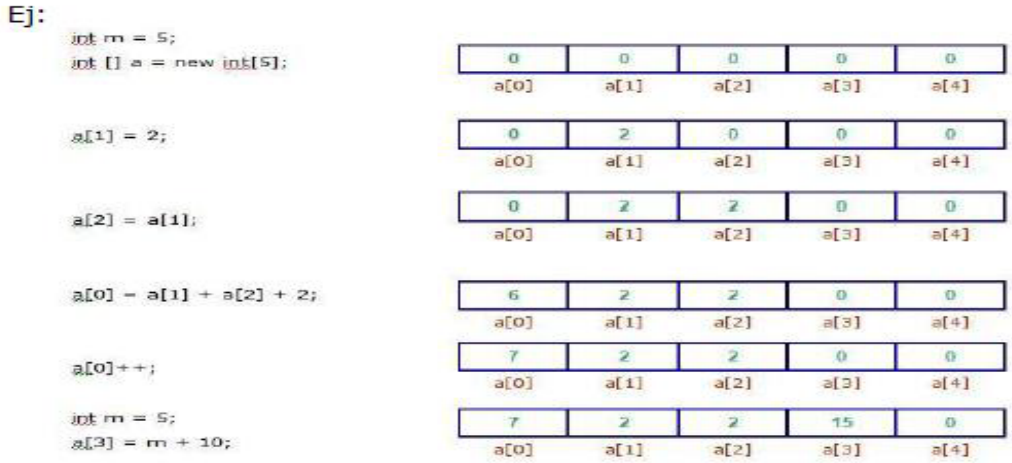
También se pueden dar otros valores iniciales al array cuando se crea. Los valores iniciales se escriben entre llaves separados por comas y deben aparecer en el orden en que serán asignados a los elementos del array. El número de valores determina el tamaño del array.

Ej: double [] notas = {6.7, 7.5, 5.3, 8.75, 3.6, 6.5};
boolean [] resultados = {true,false,true,false};
String [] dias = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes","Sábado", "Domingo"};

Acceder a los elementos de un array

Para acceder a cada elemento del array se utiliza el nombre del array y el índice que indica la posición que ocupa el elemento dentro del array. El índice se escribe entre corchetes. Se puede utilizar como índice un valor entero, una variable de tipo entero o una expresión de tipo entero.
Un elemento de un array se puede utilizar igual que cualquier otra variable.
Se puede hacer con ellos las mismas operaciones que se pueden hacer con el resto de variables (incremento, decremento, operaciones aritméticas, comparaciones, etc)

Ej:



OJO!!! Si se intenta acceder a un elemento que está fuera de los límites del array (índice negativo o con un índice mayor que el último elemento del array) ,el compilador no avisa del error. El error se producirá durante la ejecución.
En ese caso se lanza una excepción Java.lang.ArrayIndexOutOfBoundsException.

Se puede saber el número de elementos del array mediante el atributo length. Se puede utilizar length para comprobar el rango del array y evitar errores de acceso.
Ej: Para asignar un valor a un elemento del array que se leen por teclado:

```
Scanner teclado = new Scanner(System.in);
int i, valor; int [] a = new int[10];
System.out.print("Posición: ");
i = teclado.nextInt();
System.out.print("Valor: ");
valor = teclado.nextInt();
if (i>=0 && i < a.length)
a[i] = valor;
```

Recorrer un array unidimensional
Para recorrer un array se utiliza una instrucción iterativa (normalmente una instrucción for, aunque también puede hacerse con while o do..while) usando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

```
Ej: double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0};
for (int i = 0; i< 7; i++)
System.out.print(notas[i] + " ");
```

Para evitar errores de acceso al array es recomendable utilizar length para recorrer el array completo.

```
Ej: double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0};
for (int i = 0; i< notas.length; i++)
System.out.print(notas[i] + " ");
```

Ej: Programa que lee por teclado la nota de los alumnos de una clase y calcula la nota media del grupo. También muestra los alumnos con notas superiores a la media. El número de alumnos se lee por teclado.

```
import java.util.*;
public class Ejemplo {
public static void main(String[] args) {
Scanner teclado= new Scanner(System.in);
int numAlum, i;
double suma = 0, media;
do {
System.out.print("Número de alumnos de la clase: ");
numAlum = teclado.nextInt();

} while (numAlum <= 0);
double[] notas = new double[numAlum];
for (i = 0; i < notas.length; i++) {
System.out.print("Alumno " + (i + 1) + " Nota final: ");
notas[i] = teclado.nextDouble();
}
for (i = 0; i < notas.length; i++)
suma = suma + notas[i];
media = suma / notas.length;
System.out.printf("Nota media del curso: %.2f %n", media);
System.out.println("Listado de notas superiores a la media:
");
for (i = 0; i < notas.length; i++)
if (notas[i] > media)
System.out.println("Alumno numero " + (i + 1)+ " Nota
final: " + notas[i]);
}
}
```

Arrays de caracteres en Java
Un array de caracteres en Java se crea de forma similar a un array de cualquier otro tipo de datos.
Ejemplo: Array de 8 caracteres llamado cadena. Por efecto se inicializa con el carácter nulo.
char [] cadena = new char[8];

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

Ejemplo: Array de 5 caracteres llamado vocales. Se asignan valores iniciales: a, e, i, o, u char [] vocales = {'a', 'e', 'i', 'o', 'u'};

a	e	i	o	u
vocales[0]	vocales [1]	vocales [2]	vocales [3]	vocales [4]

A diferencia de los demás arrays, se puede mostrar el contenido completo de un array de caracteres mediante una sola instrucción **print** o **printf**.
Ej: System.out.println(cadena); //Muestra 8 caracteres nulos (en blanco)
System.out.println(vocales); //Muestra aeiou

El atributo length de un array de caracteres contiene el tamaño del array independientemente de que sean caracteres nulos u otros caracteres.
Ejemplo: char [] cadena = new char[8];

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

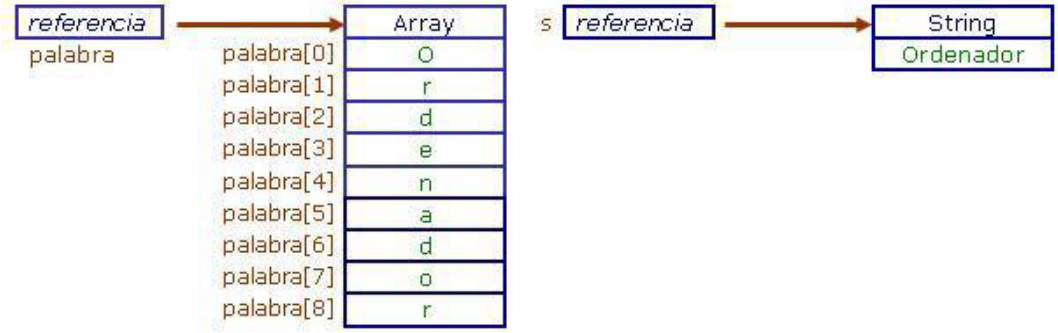
System.out.println(cadena.length); // Muestra: 8
cadena[0] ='m';
cadena[1] ='n';

m	n	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

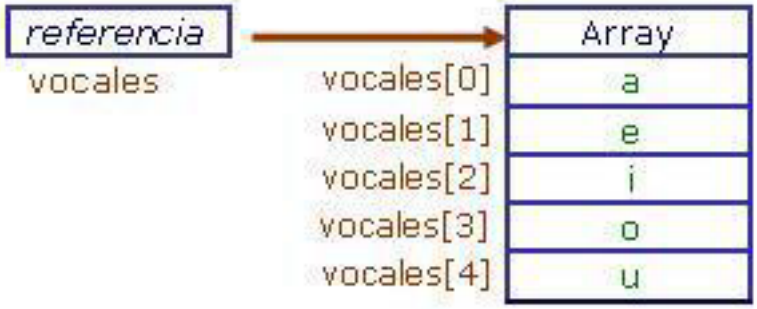
System.out.print(cadena);
System.out.print(cadena);
System.out.println(".");

Muestra: mnbbbbbbmnbbbbbb. //b representa los espacios en blanco
Se puede asignar un String a un array de caracteres mediante el método **toCharArray()** de la clase String.
Ej: **String s = "Ordenador";**

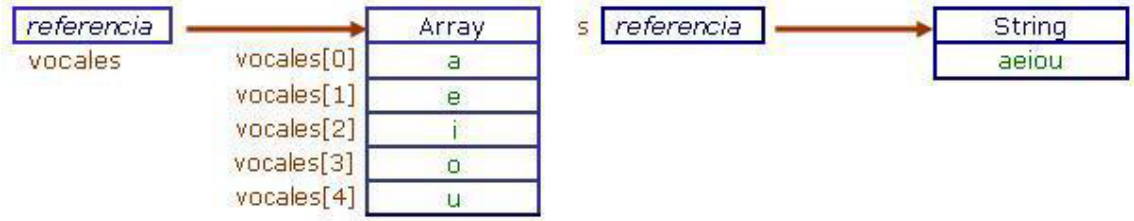
char [] palabra = s.toCharArray();
Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a palabra



Se puede crear un String a partir de un array de caracteres.
Ej: char [] vocales = {'a', 'e', 'i', 'o', 'u'};
String s = new String(vocales);



Se crea un nuevo String con el contenido del array vocales y se asigna la dirección de memoria a s.



Recorrer un array de caracteres unidimensional
Se puede recorrer de forma completa utilizando una instrucción iterativa, normalmente un for.
Ej: char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
for(int i = 0; i<s.length; i++)
System.out.print(s[i]+ " "); //Muestra todos los caracteres del array,
// incluidos los nulos.
Si los caracteres no nulos se encuentran al principio del array se puede recorrer utilizando un while, mientras que no encontremos un carácter nulo.
Ejemplo: char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
int i = 0;
while(s[i]!='\0'){
System.out.print(s[i]); // Muestra los caracteres del array hasta que i++;
// encuentra el primer nulo.
}

Arrays Bidimensionales (Matrices)

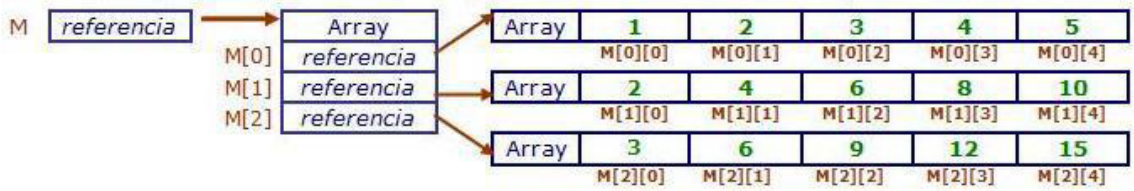
Un array puede tener más de una dimensión. El caso más general son los arrays bidimensionales también llamados matrices o tablas. La dimensión de un array la determina el número de índices necesarios para acceder a sus elementos.
Los arrays unidimensionales porque solo utilizan un índice para acceder a cada elemento.

Una matriz necesita dos índices para acceder a sus elementos. Gráficamente se puede representar una matriz como una tabla de n filas y m columnas cuyos elementos son todos del mismo tipo.

La siguiente figura representa un array M de 3 filas y 5 columnas:

	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

Pero en realidad una matriz en Java es un array de arrays.
Gráficamente podemos representar la disposición real en memoria del array anterior así:

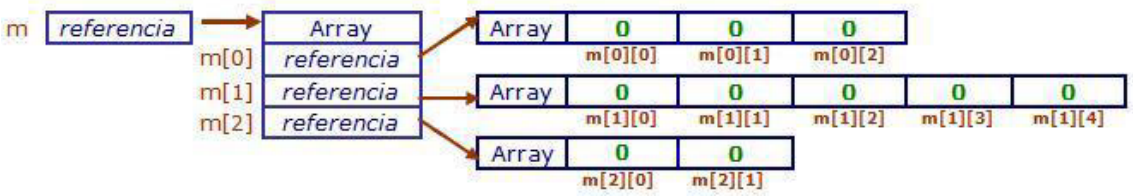


La longitud del array M (M.length) es 3.
La longitud de cada fila del array (M[i].length) es 5.
Para acceder a cada elemento de la matriz se utilizan dos índices. El primero indica la fila y el segundo la columna.

matrices en JAVA

Se crean de forma similar a los arrays unidimensionales, añadiendo un índice.
Ejemplo: Matriz de datos de tipo int llamado ventas de 4 filas y 6 columnas.
int [][] ventas = new int[4][6];
Matriz de datos double llamado temperaturas de 3 filas y 4 columnas.
double [][] temperaturas = new double[3][4];
En Java se pueden crear arrays irregulares en los que el número de elementos de cada fila es variable. Solo es obligatorio indicar el número de filas.
Ejemplo: int [][] m = new int[3][]; //crea una matriz m de 3 filas.
A cada fila se le puede asignar un número distinto de columnas:
m[0] = new int[3];
m[1] = new int[5];
m[2] = new int[2];

Gráficamente podemos representar la disposición real en memoria del array anterior así:



Inicializar matrices

Una matriz es un objeto, por tanto, cuando se crea, a sus elementos se les da automáticamente un valor inicial, al igual que a los array unidimensionales.
- 0 para arrays numéricos
- '\u0000' (carácter nulo) para arrays de caracteres
- false para arrays booleanos
- null para arrays de String y de referencias a objetos.

También es posible dar otros valores iniciales a la matriz cuando se crea.
Los valores iniciales se escriben entre llaves separados por comas.
Los valores que se le asignen a cada fila aparecerán a su vez entre llaves separados por comas.
El número de valores determina el tamaño de la matriz.
Ej: int [][] numeros = {{6,7,5}, {3, 8, 4}, {1,0,2}, {9,5,2}};
Se crea la matriz numeros de tipo int, de 4 filas y 3 columnas, y se le da esos valores iniciales.

Recorrer matrices

Para recorrer una matriz se anidan dos bucles for. En general para recorrer un array multidimensional se anidad tantas instrucciones for como dimensiones tenga el array.
Ej: Programa que lee por teclado números enteros y los guarda en una matriz de 5 filas y 4 columnas. A continuación muestra los valores leídos, el mayor y el menor y las posiciones que ocupan.

```
import java.util.*;
public class Bidimensional {
public static void main(String[] args) {
final int FILAS = 5, COLUMNAS = 4;
Scanner teclado = new Scanner(System.in);
int i, j, mayor, menor;
int filaMayor, filaMenor, colMayor, colMenor;
int[][] A = new int[FILAS][COLUMNAS];
System.out.println("Lectura de elementos de la matriz: ");
```

```
for (i = 0; i < FILAS; i++)
for (j = 0; j < COLUMNAS; j++) {
System.out.print("A[" + i + "][" + j + "]= ");
A[i][j] = teclado.nextInt();
}

System.out.println("valores introducidos:");
for (i = 0; i < A.length; i++) {
for (j = 0; j < A[i].length; j++)
System.out.print(A[i][j] + " ");
System.out.println();
}
mayor = A[0][0]; //se toma el primero como mayor y menor
menor = A[0][0];
filaMayor = filaMenor = colMayor = colMenor = 0;
for (i = 0; i < A.length; i++) {
for (j = 0; j < A[i].length; j++) {
if (A[i][j] > mayor) {
mayor = A[i][j];
filaMayor = i;
colMayor = j;
} else if (A[i][j] < menor) {
menor = A[i][j];
filaMenor = i;
colMenor = j;
}
}
}

System.out.print("Elemento mayor: " + mayor);
System.out.println(" Fila: "+ filaMayor + " Columna: " + colMayor);
System.out.print("Elemento menor: " + menor);
System.out.println(" Fila: "+ filaMenor + " Columna: " + colMenor);
}
}
```

Se usa siempre length para obtener el número de columnas que tiene cada fila.

```
for (i = 0; i < a.length; i++) { //número de filas
for (j = 0; j < a[i].length; j++) //número de columnas de cada fila
System.out.print(a[i][j] + " ");
System.out.println();
}
```

ArrayList

La ***clase ArrayList*** permite el almacenamiento de datos en memoria de forma similar a los arrays convencionales, pero con la gran ventaja de que el número
de elementos que puede almacenar es dinámico. La cantidad de elementos que puede almacenar un array está limitado a la dimensión que se declara a la hora
de crearlo o inicializarlo. Los ArrayList, en cambio, pueden almacenar un número variable de elementos sin estar limitados por un número prefijado.

Declaración de un objeto ArrayList

La declaración genérica de un ArrayList se hace con un formato similar al
siguiente:
ArrayList nombreDeLista;
De esta manera no se indica el tipo de datos que va a contener, cosa que suele ser recomendable, para que así se empleen las operaciones y métodos adecuados para el tipo de datos manejado.

Para especificar el tipo de datos que va a contener la lista se debe indicar entre los caracteres '<' y '>' la clase de los objetos que se almacenarán**: ArrayList<nombreClase> nombreDeLista;**

En caso de almacenar datos de un tipo básico como char, int, double, etc, se debe especificar el nombre de la clase asociada: Character, Integer, Double, etc.
Ej: *ArrayList<String> listaPaises;*
ArrayList<Integer> edades

Creación de un ArrayList

Para crear un ArrayList se puede seguir el siguiente formato:
nombreDeLista = new ArrayList();

Se puede declarar la lista a la vez que se crea:
ArrayList<nombreClase> nombreDeLista = new ArrayList(); Ej:
ArrayList<String> listaPaises = new ArrayList();
La **clase ArrayList** forma parte del paquete **java.util**, por lo que hay que incluir en la parte inicial del código la importación de ese paquete.

Añadir elementos al final de la lista
El método add posibilita añadir elementos. Los elementos que se van añadiendo, se colocan después del último elemento que hubiera en el ArrayList. En primer elemento que se añada se colocará en la posición 0.
Ej: ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España"); //Ocupa la posición 0
listaPaises.add("Francia"); //Ocupa la posición 1
listaPaises.add("Portugal"); //Ocupa la posición 2

Ejemplo: Se pueden crear ArrayList para guardar datos numéricos de igual manera
ArrayList<Integer> edades = new ArrayList();
edades.add(22);
edades.add(31);
edades.add(18);

Añadir elementos en cualquier posición de la lista
También es posible insertar un elemento en una determinada posición desplazando el elemento que se encontraba en esa posición, y todos los siguientes, una posición más. Para ello, se emplea también el *método add* indicando como primer parámetro el número de la posición donde se desea colocar el nuevo elemento.
Ej: ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal
listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal
Si se intenta insertar en una posición que no existe, se produce una excepción (IndexOutOfBoundsException)

Suprimir elementos de la lista
Si se quiere eliminar un determinado elemento de la lista se puede emplear el método remove al que se le puede indicar por parámetro un valor int con la posición a suprimir, o bien, se puede especificar directamente el elemento a eliminar si es encontrado en la lista.
Ej: ArrayList<String> listaPaises = new ArrayList();
listaPaises.add("España");
listaPaises.add("Francia");
listaPaises.add("Portugal"); //El orden es: España, Francia, Portugal
listaPaises.add(1, "Italia"); //Ahora es: España, Italia, Francia, Portugal
listaPaises.remove(2); //Eliminada Francia, queda: España, Italia, Portugal
listaPaises.remove("Portugal"); //Eliminada Portugal, queda: España, Italia

Consulta de un determinado elemento de la lista
El método get permite obtener el elemento almacenado en una determinada posición que es indicada con un parámetro de tipo int. Con el elemento obtenido se podrá realizar cualquiera de las operaciones posibles según el tipo de dato del elemento (asignar el elemento a una variable, incluirlo en una expresión, mostrarlo por pantalla, etc).
Ej: System.out.println(listaPaises.get(3)); // Mostraría: Portug

Modificar un elemento contenido en la lista
Es posible modificar un elemento que previamente ha sido almacenando en la lista utilizando el método set. Como primer parámetro se indica, con un valor int, la posición que ocupa el elemento a modificar, y en el segundo parámetro se especifica el nuevo elemento que ocupará dicha posición sustituyendo al elemento anterior.
Ejemplo: En la lista de países se desea modificar el que ocupa la posición 1 (segundo en la lista) por "Alemania".
listaPaises.set(1, "Alemania");

Buscar un elemento
La clase ArrayList facilita mucho las búsquedas de elementos gracias al método indexOf que retorna, con un valor int, la posición que ocupa el elemento que se indique por parámetro. Si el elemento se encontrara en más de una posición, este método retorna la posición del primero que se encuentre. El método lastIndexOf obtiene la posición del último encontrado.
En caso de que no se encuentre en la lista el elemento buscado, se obtiene el valor -1
Ejemplo: Comprobar si Francia está en la lista, y mostrar su posición.

```
String paisBuscado = "Francia";
int pos = listaPaises.indexOf(paisBuscado);
if(pos!=-1)
System.out.println(paisBuscado + " encontrado en la posición:
"+pos);
else
System.out.println(paisBuscado + " no se ha encontrado");
```

Recorrer el contenido de la lista
Es posible obtener cada uno de los elementos de la lista utilizando un bucle con tantas iteraciones como elementos contenga, de forma similar a la usada con los arrays convencionales. Para obtener el número de elementos de forma automática se puede emplear el método size() que devuelve un valor int con el número de elementos que contiene la lista.
Ejemplo: for(int i=0; i<listaPaises.size(); i++)

System.out.println(listaPaises.get(i));
También se puede emplear otro formato del bucle for (bucle foreach) en el que se va asignando cada elemento de la lista a una variable declarada del mismo tipo que los elementos del ArrayList.
Ejemplo: for(String pais:listaPaises)
System.out.println(pais);
Si el ArrayList contiene objetos de tipos distintos o se desconoce el tipo se pondría.
for(Object o: nombreArray)
System.out.println(o);
También es posible hacerlo utilizando un objeto Iterator. La ventaja de usar un Iterador es que no se necesita indicar el tipo de objetos que contiene el
ArrayList. Iterator tiene como métodos:
hasNext: devuelve true si hay más elementos en el array.
next: devuelve el siguiente objeto contenido en el array.
Ejemplo: ArrayList<Integer> nros = new ArrayList<Integer>();
//se insertan elementos
Iterator it = nros.iterator(); //se crea el iterador it para el ArrayList nros
while(it.hasNext()) //mientras queden elementos
System.out.println(it.next()); //se obtienen y se muestran

Otros métodos
void clear(): Borra todo el contenido de la lista.
Object clone(): Retorna una copia de la lista.
boolean contains(Object elemento): Retorna true si se encuentra el elemento indicado en la lista, y false en caso contrario.
boolean isEmpty(): Retorna true si la lista está vacía.

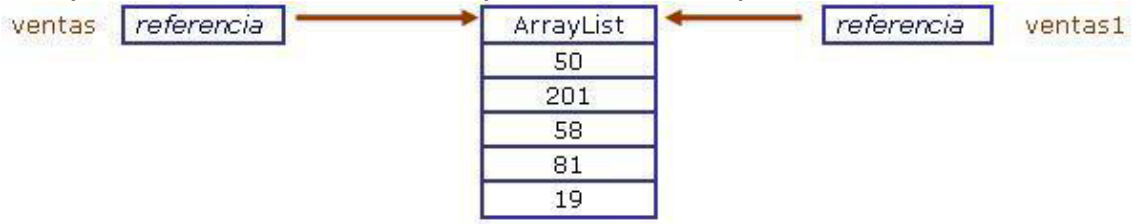
Ejemplos de uso de ArrayList
Ejemplo: ArrayList<String> nombres = new ArrayList<String>();
nombres.add("Ana");
nombres.add("Luisa");
nombres.add("Felipe");
System.out.println(nombres); // [Ana, Luisa, Felipe]
nombres.add(1, "Pablo");
System.out.println(nombres); // [Ana, Pablo, Luisa, Felipe]
nombres.remove(0);
System.out.println(nombres); // [Pablo, Luisa, Felipe]
nombres.set(0,"Alfonso");
System.out.println(nombres); // [Alfonso, Luisa, Felipe]
String s = nombres.get(1);
String ultimo = nombres.get(nombres.size() – 1);
System.out.println(s + " " + ultimo); // Luisa Felipe

Ejemplo: Escribe un programa que lea números enteros y los guarde en un ArrayList hasta que se lea un 0 y muestra los números leídos, su suma y su media. Import java.util.*; public class ArrayList2 { public static void main(String[] args) { Scanner teclado = new Scanner(System.in); ArrayList<Integer> numeros = new ArrayList<Integer>(); int n; do { System.out.println("Introduce números enteros. 0 para acabar: "); System.out.println("Numero: "); n = teclado.nextInt(); if (n != 0) numeros.add(n); }while (n != 0);

System.out.println("Ha introducido: " + numeros.size() + " números:");
System.out.println(numeros); //Muestra el arrayList completo
Iterator it = numeros.iterator();
while(it.hasNext())
System.out.println(it.next());
double suma = 0;
for(Integer i: numeros)
suma = suma + i;
System.out.println("Suma: " + suma);
System.out.println("Media: " + suma/ numerous.size());
}
}

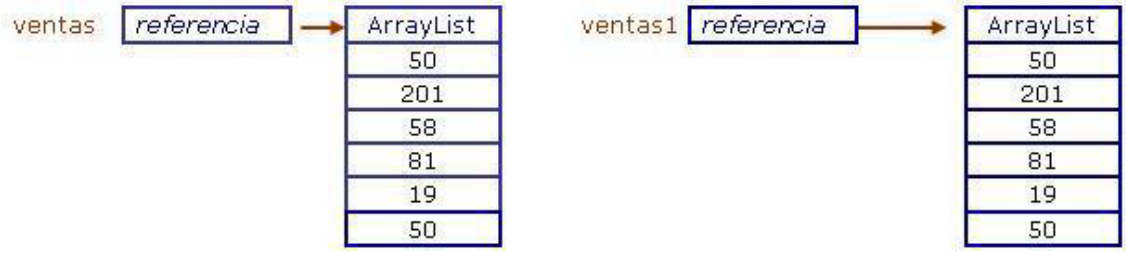
Copiar un ArrayList
El nombre de un ArrayList contiene la referencia al ArrayList, es decir, la dirección de memoria donde se encuentra el ArrayList, igual que sucede con los arrays estáticos.
Ejemplo: Se tiene un ArrayList de enteros llamado ventas.

La instrucción ArrayList<Integer> ventas1 = ventas; no copia el array ventas en el nuevo array ventas1 sino que crea una referencia



De esta forma se tiene dos formas de acceder al mismo

ArrayList: mediante la referencia ventas y mediante la referencia ventas1.
Para hacer una copia se puede hacer de forma manual elemento a elemento o se puede pasar la referencia del ArrayList original al constructor del nuevo.
ArrayList<Integer> ventas1 = new ArrayList<Integer>(ventas);



ArrayList como parámetro de un método
Un ArrayList puede ser usado como parámetro de un método. Además un método también puede devolver un ArrayList mediante la sentencia return.
Ejemplo: Método que recibe un ArrayList de String y lo modifica invirtiendo su contenido.

```
import java.util.*;
public class ArrayList4 { public static void main(String[] args) {
ArrayList<String> nombres = new ArrayList<String>();
nombres.add("Ana");
nombres.add("Luisa");
nombres.add("Felipe");
nombres.add("Pablo");
System.out.println(nombres);
nombres = invertir(nombres);
System.out.println(nombres);
}
public static ArrayList<String> invertir(ArrayList<String> nombres) {
ArrayList<String> resultado = new ArrayList<String>();
for (int i = nombres.size() - 1; i >= 0; i--)
resultado.add(nombres.get(i));
return resultado;
}
}
```