TEMA 1 Elementos de un Programa Informático

Crear nuestro Primer Programa

Los pasos a realizar son:

1. Abrir el entorno de netbeans
2. Seleccionar la opción de crear un proyecto nuevo con la categoría "Java with Maven" y proyecto "Java Aplication"
3. Una vez seleccionado daremos nombre al proyecto y asignaremos un directorio donde se guardará el proyecto
4. Y se nos genera un código

Variables en Java

Las variables son contenedores que sirven para almacenar los datos que utiliza un programa. Tienen un nombre (identificador) que solo contiene letras y números y el caracter de subrayado

```
tipo nombreVariable;
```

Ejemplos

```
int dias; // días es un número entero, sin decimales
boolean decision; //solo puede ser true o false
```

Se puede hacer que una variable tome un valor inicial

```
int dias=365;
```

Y se puede declarar más de una variable a la vez del mismo tipo

```
int dias=365, anio=23, semanas=12;
```

Tipos de Datos

Tipo	Tamaño	Dato
Byte	8	Entero

Tipo	Tamaño	Dato
Short	16	Entero
Int	32	Entero
Long	64	Entero
Float	32	Coma Flotante
Double	64	Coma Flotante
Boolean	16	Booleanos
Char	8	Carácter

Caracteres Especiales

Caracter	Significado
\b	Retroceso
\t	Tabulador
\n	Nueva Línea
\r	Retorno de Carro
"	Dobles comillas
'	Comillas Simples
\	Barra Inclinada

Conversión entre tipos (Casting)

```
int a;
byte b;
b=(byte) a;
```

Otro ejemplo

```
byte n1=100, n2=100, n3;
n3=(byte)(n1 * n2 / 100);
```

Operadores

Operador	Descripción
+	Suma dos operandos

Operador	Descripción
-	Resta dos operandos
*	Multiplica dos operandos
/	Divide dos operandos
%	Calcula el resto de dividir el Operador1 entre el operador2

Incrementales o unarios

Operador	Uso	Descripción
++	x++	Incrementa x en 1. Evalúa antes
++	++x	Incrementa x en 1. Evalúa después
--	x--	Decrementa x en 1. Evalúa antes
--	--x	Decrementa x en 1. Evalúa después

```
int x=5, y=5
System.out.println(++x); //Imprime 6
System.out.println(x);   //Imprime 6
System.out.println(y++)  //Imprime 5
System.out.println(y)    //Imprime 6
```

Relacionales

Operador	Uso	Devuelve Verdadero
>	x>y	x es mayor que y
>=	x>=y	x es mayor o igual que y
<	x<y	x es menor que y
<=	x<=y	x es menor o igual que y
==	x==y	x es igual a y
!=	x!=y	x es distinto de y

Lógicos

Operador	Uso	Devuelve verdadero
&& (and)	Condicion1 && Condicion2	Condicion1 y Condicion2 verdaderas
"		(or)"
!(not)	!Condicion	Condicion falsa

Asignación

Operador	Uso	Descripcion
=	x=y	Asigna a x el valor de y
+=	x+=valor	Equivalente a x=x+valor
-=	x-=valor	Equivalente a x=x-valor
*=	x*=valor	Equivalente a x=x*valor
/=	x/=valor	Equivalente a x=x/=valor

Ternario

Este operador devuelve un valor que se selecciona de dos posibles. Puede tomar dos valores verdadero o falso

```
expresioncondicional ? valor1 : valor2
```

En caso de que se devuelva verdadero, devuelve valor1, y cuando resulte false, devuelve valor2

```
int num=11;
int valorAbs=num>0?num:-num;

System.out.println("El valor absoluto de "+num+" es: "+valorAbs);
```

Constantes

Es una variable de sólo lectura. Es un valor que no puede variar (por lo tanto no es una variable). Las constantes deberían en mayúsculas.

```
final int MAXIMO=999999;
```

Escritura por pantalla

Orden Printf

Utiliza códigos de conversión para indicar de qué tipo es el contenido a mostrar. Se caracterizan porque llevan delante el simbolo %.

- %c: Escribe un carácter
- %s: Escribe una cadena de texto
- %d: Escribe un entero

- %f: Escribe un número en punto flotante
- %e: Escribe un núemro en punto flotante en notación científica

```
System.out.printf("%,.2f\n",12345.1684);
```

Lectura por Teclado

Se hace de tres formas distintas:

1. Visualizando una caja por pantalla

- Hay que incluir el paquete javax.swing y utilizar la clase JOptionPane:

```
import javax.swing.JOptionPane;

String texto;
int num;

texto= JOptionPane.showInoutDialog("Escribe un número");
num=Integer.parseInt(texto);
System.out.println("Has introducido el número "+num);

//Mostrando el resultado en un cuadro de diálogo
JOptionPane.showMessageDialog(null,"Has introducio el número "+num);

//Mensaje de advertencia
JOptionPane.showMessageDialog(null,
"Advertencia","Peligro",JOptionPane.Warning_Message);

//Mensaje de Error
JOptionPane.showMessageDialog(null,"Error","Fatal
Error",JOptionPane.Error_Message);

//Mensaje de pregunta
JOptionPane.showMessageDialog(null,"Pregunta","Mensaje de
Pregunta",JOptionPane.Question_Message);
```

2. Usando la clase System

Con esta opción, se va a capturar el contenido de System.in, mediante el uso de dos clases que se encuentran dentro del paquete java.io(import java.io.*)

- InputStreamReader: Captura los bytes del buffer y los convierte a caracteres
- BufferedReader: Clase que proporciona un método que permite leer hasta el final de la línea

```
InputStreamReader isr= new InputStreamReader(System.in);
BufferedReader br= new BufferedReader(isr);
```

```
System.out.println("Introduce un número:m ");
String cad= br.readLine();

//Conversión de la variable cad a el tipo de dato que quiero
int num1=Integer.parseInt;
System.out.println(num1);
```

3. Usando la clase Scanner

```
int edad;
String nombre,apellido;
Scanner teclado=new Scanner(System.in);

System.out.println("Introduce tu nombre: ");
nombre=teclado.nextLine();

System.out.println("Introduce tus apellidos: ");
apellido=teclado.nextLine();

System.out.println("Introduce tu edad: ");
edad=teclado.nextInt();

System.out.println("Nombre: "+nombre+"\nApellidos: "+apellido+"\nEdad: "+edad);
```

Enumerados

Son una forma de declarar una variable con un conjunto restringido de valores

```
enum TamanoDeCafe{PEQUEÑO,MEDIANO,GRANDE};
```

O creando una clase aparte

```
public class Ejemplo_Enumerados {
    public enum nivel{
        BAJO,MEDIO,ALTO;
    }
```

TEMA 2 UTILIZACIÓN DE OBJETOS Y DESARROLLO DE CLASES

Java y la programación orientada a objetos(POO)

Con la POO los problemas se dividen en objetos. Cada uno de ellos funcionan de forma totalmente independiente.

Un objeto es un elemento del programa que integra sus propios datos y su propio funcionamiento. Una clase es lo que define a un tipo de objeto. Al definir una clase lo que se hace es indicar como funciona un determinado tipo de objetos.

Elementos de la POO: clases y objetos

Una clase es un tipo al cual pertenecen **objetos** o **instancias de la clase**.

- Atributos: Define sus propiedades (datos que almacena el objeto)
- Métodos: Define el código de sus métodos (comportamiento)

Ejemplo

```
public class Pez {

    //Definimos atributos con su tipo y su nombre
    private String color;
    private String tipo;
    private int tamano;

    //contructores ->Permiten instanciar la clase y su nombre es el mismo
    //contructor por defecto
    public Pez() {

    }

    //constructor un parámtero del tipo int parámetros
    public Pez(int _tamano) {
        tamano = _tamano;
    }

    //constructor de tres parámetros
    public Pez(String miColor, String miTipo, int miTamano) {
        color = miColor;
        tipo = miTipo;
        tamano = miTamano;
    }

    //Definimos los métodos
    public void respirar() {
        //contenido del método
    }

    public void nadar() {
        //contenido del método
    }

    public void alimentarse() {
        //contenido del método
```

```
    }

    //Métodos get y set para obtener y modificar los atributos de la clase
    public String getColor() {
        return color;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getTipo() {
        return tipo;
    }

    public void setTipo(String tipo) {
        this.tipo = tipo;
    }

    public int getTamano() {
        return tamano;
    }

    public void setTamano(int tamano) {
        this.tamano = tamano;
    }


}
```

Uso de clases y objetos

Una vez tenemos creada la clase, vamos a proceder a crear un objetos. A la creación de un objeto se denomina instanciación del objeto, o crear una instancia de la clase

Para utilizar un objeto primero se debe definir una variable que lo referenciará

```
Clase variable
```

Luego se crea el objeto

```
variable=new Constructor(lista de parámetros);
```

Más ejemplos de creación de objetos de la clase Pez

```

```

```
Pez carpa=new Pez();
Pez grande=new Pez(50);
Pez cometa= new Pez("Naranja","Cometa",8);
```

Palabra reservada this

Su utilidad es resolver ambigüedades cuando existen atributos con el mismo identificador que alguna variable local o parámetro.

```
public class Pez
{
    private int tamano;
    public Pez(int tamano)
    {
        this.tamano=tamano;
    }
}
```

Métodos get y set

Sirve para obtener o modificar los atributos de una clase

```
 //Método para obtener el radio del circulo
   public double getRadio() {
       return radio;
   }

   //Método para asignar el valor de radio
   public void setRadio(double radio) {
       this.radio = radio;
   }
```

Operador "."

Se utiliza para acceder a los miembros de una clase

```
//Introduzco la radio por teclado para c3
System.out.println("Introduce un radio para c3: ");
radio = teclado.nextDouble();
c3 = new Circulo(radio);

//Cambiamos el radio de c1 a 5
c1.setRadio(5);
System.out.printf("El radio de c1 es: %.2f", c1.getRadio());
```

Modificadores de Acceso

	La Misma Clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
public	x	x	x	x
protected	x	x	x	
default	x	x		
private	x			

Definir Métodos

Un **método** es una llamada a una operación de un determinado objeto. La mayoría de métodos devuelven un resultado (gracias a un return). Si el método no devuelve ningún resultado se indica como tipo de datos a devolver el tipo void.

Para construir un método necesitamos:

1. Sus especificadores de alcance o visibilidad
2. El tipo de datos o de objeto que devuelve
3. El identificador del método
4. Los parámetros
5. El cuerpo del método

```
 public boolean esEquilatero() {
        boolean equilatero = false;
        if (lado1 == lado2 && lado2 == lado3) {
            equilatero = true;
        }
        return equilatero;
    }
```

Constructores

Es un método que se invoca cuando se crea un objeto y que sirve para iniciar los atributos del objeto

- ***Constructor por defecto***

```
public Rectangulo() {

    }
```

- ***Constructor por parámetros***

```
public Alimento(String nombre, int grasas, int hidratos, boolean origenAnimal) {
        this.nombre = nombre;
        this.grasas = grasas;
        this.hidratos = hidratos;
        this.origenAnimal = origenAnimal;
    }
```

- ***Constructor Copia***

```
public class Noria
{
    private double radio;

public Noria(){
    this.radio = 55;
}
public Noria(double radio){
    this.radio = radio;
}
public Noria(Noria n){
    this.radio=n.radio;
}
}
```

Métodos recursivos

Es una técnica de escritura de métodos o funciones, pensada para problemas complejos. La idea parte de que un método puede invocarse a sí mismo.

```
public class Matematicas
{
        public double factorial(int n)
    {
        if(n<=1){
            return 1;
        }
        else{
            return n*factorial(n-1);
        }
    }
}
```

CADENAS DE TEXTO EN JAVA

Introducción

La clase String nos permitirá almacenar cadenas de caracteres. Es dinámica, por lo tanto le podemos asignar diferentes valores(cadenas)con diferente tamaño.

Creación

Podemos tratarla como un dato primitivo. Una vez asignado un valor este es inmodificable. Cada vez que se asigna un nuevo valor(nueva cadena), provoca una nueva instanciación interna pero transparentes para nosotros.

```
public class usoString{
    public static void main(String[] args){
        String modulo="programación";
        String ciclo= new String();
        ciclo="Desarrollo de Aplicaciones Web";
    }
}
```

Asignación de valores

Se representa con comillas dobles. Las dobles comillas no forman parte de la cadena. Si queremos que la propia " sea un carácter en sí, y no el final de la cadena, hay que antepones la secuencia escape.

```
System.out.println("*****ASIGNACION DE VALORES*****");
String introduccion;
introduccion="Trataremos inicialmente los 'String' para el manejo de cadenas. Son
muy \"IMPORTANTES\"");
```

Extracción de un carácter individual

Con el método charAt(posicion), podemos acceder al carácter que ocupa dicha posición.

```
 for (int i = 0; i < frase.length(); i++) {
            if (frase.charAt(i) == 'a' || frase.charAt(i) == 'e' ||
frase.charAt(i) == 'i' || frase.charAt(i) == 'o' || frase.charAt(i) == 'u') {
               frase_convertida+="o";
            }else{
                frase_convertida+=frase.charAt(i);
            }
        }
```

Obtener la longitud de la cadena

Nos proporciona la longitud de la cadena, incluyendo espacios en blanco ocupados dentro de la cadena. Para ello se hace el uso del método lenght().

```
public static void main(String[] args) {
        String nombre = "Adrián", primerApellido = "Tresgallo", segundoApellido =
"Arozamena";
        //Concateno los dos apellidos con mi nombre con nombre.concat
        nombre = nombre.concat(" " + primerApellido + " " + segundoApellido);
        //Muestro mi nombre completo y la longitu
        System.out.println("Mi nombre es: " + nombre + "\nLongitud: " +
nombre.length());
    }
```

Descomposición de una cadena

Es un método que permite extraer una subcadena, indicando la posición inicial y final

```
System.out.println("*****Extraccion de una subcadena*****");
System.out.println("Subcadena de la cadena ciclo de las cuatro primeras caracteres
que podemos nombrar como 0, 1, 2, 3 (el número de caracteres es cuatro). El
carácter final indicado en la llamada al método, el número 4, queda excluido del
substring.: "+ciclo.substring(0,4);
```

Igualdad de dos cadenas

Si queremos averigüar si dos cadenas son iguales, distingue si está en mayúsculas o minúsculas. Si necesitamos preguntar sin distinguir mayúsculas de mnúsculas, utilizamos equalsIgnoreCase().

```
System.out.println("*****COMPARACIÓN ENTRE CADENAS*****);
String texto1="Buenos días";
String texto2=new String("Buenos días");
System.out.println("¿Las cadenas son iguales?: "+texto1.equals(texto2));
System.out.println("¿Las cadenas son iguales?: " + texto1==texto2); // El
resultado de esta instrucción puede ser incorrecto.
```

Otras funciones

- startsWith(subcadena) y endsWith(subcadena): Para comprobar si una cadena comienza o termina con la subcadena determinada.

```
public static void main(String[] args)
{
    String cdn="Hola, me llamo adrian";
    String msj = "Valor devuelto: ";

    System.out.println(msj + cdn.startsWith("Hola"));
    System.out.println(msj + cdn.startsWith("llamo"));
```

```
    System.out.println(msj + cdn.endsWith("adrian",15));
}
```

- trim(): Elimina los espacios en blanco de una cadena que tenga por delante y por detrás. No elimina los espacios intermedios

```
String sCadena = " Esto Es Una Cadena " ;
System.out.println(sCadena.trim());  //Devuelve "Esto Es Una Cadena"
```

- toUpperCase() y toLowerCase(): permite cambiar todos los caracteres por mayúsculas o minúsculas

```
 public static void main(String[] args)
{
    String cdn="Hola, me llamo adrian";
    String msj = "Valor devuelto: ";

    System.out.println(msj + cdn.toUpperCase());
    System.out.println(msj + cdn.toLowerCase();
}
```

- indexOf(cadenaABuscar): permite buscar una cadena dentro de otra

```
public static void main(String[] args)
{
    String cdn="Hola, me llamo adrian";
    String msj = "Valor devuelto: ";

    System.out.println(msj + cdn.indexOf("llamo"));
    System.out.println(msj + cdn.indexOf("Hola",0);
}
```

- replace(cadenaABuscar,cadenaSustituta):permite reemplazar una cadena por otra

```
public static void main(String[] args)
{
    String cdn="Hola, me llamo adrian";
    String msj = "Valor devuelto: ";

    System.out.println(msj + cdn.replace("me","no me"));
}
```

- lastIndexOf(String cad): Retorna la posición de la ultima ocurrencia de la cadena

- lastIndexOf(String cadena, int ini): Retorna la posición de la ultima ocurrencia de la cadena dada como parámetro buscando en retroceso a partir de la posición dada como parámetro

---

Tema 3 Estructuras De Control

IF - Sentencia Condicional Simple

Se trata de una sentencia que, tras evaluar una expresión lógica, ejecuta una serie de instrucciones en caso de que la expresión lógica sea verdadera.

```
if(expresión lógica){
    instrucciones
    .....
}

//Ejemplo
if(nota>=5){
    System.out.println("Aprobado");
    aprobados++;
}
```

También se puede crear el if sin llaves

```
if(expresión logica) sentencia;
//Ejemplo
if(notas<5)
    System.out.println("Suspenso");
```

IF - Sentencia Condicional Compuesta

```
if(expresión lógica){
    instrucciones
    ....
}else{
    instrucciones
    .....
}
//Ejemplo

if(nota>=5){
    System.out.println("Aprobado");
    aprobados+++;
}else{
    System.out.println("Suspensos");
```

```
    suspensos++;
}
```

IF - Anidación

Dentro de una sentencia if se puede colocar otra sentencia if. Permite crear programas donde se valoren expresiones complejas.

```
if(x==1){
    instrucciones
    ...
}else{
    if(x==2){
        instrucciones
        .....
    }else{
        if(x==3){
            instrucciones
            ...
        }
    }
}
```

Una forma más elegible de escribir sería con la instruccion if-else-if

```
if(x==1){
    instrucciones
    ...
}else if(x==2){
    instrucciones
    ...
}else if(x==3){
    instrucciones
    ....
}
```

Switch

Se llama estructura condicional compleja porque permite evaluar varios valores a la vez. Sirve como sustituta de algunas expresiones del tipo if-else-if

```
switch(expresiónEntera){
    case valor1:
        instrucciones del valor 1
        break;
    case valor2:
```

```
        instrucciones del valor2
        break;
    ...
    default:
        /*instrucciones que se ejecutan si la expresión
        no toma ningun de los valores anteriores*/
}
```

Otra sintaxis de la expresión es:

```
switch (expresiónEntera){
    case valor1-> instrucciones del valor1;
    case valor2-> instrucciones del valor2;
    ....
    default-> instrucciones que se ejecutan si la expresión no toma ninguno de los
valores anteriores
}
```

While - Sentencia repetitiva mientras

Permite crear bucles. Es un conjunto de sentencias que se repiten mientras se cumpla una determinada condición.

```
while(expresión lógica){
    sentencias que se ejecutan si la condición es true;
}
```

El programa se ejecuta siguiendo:

1. Se evalúa la expresión lógica
2. Si la expresión es verdadera ejecuta las sentencias, sino el programa abandona la sentencia
3. Tras ejecutar, volvemos al paso 1;

```
int i=1;
while (i<=100){
    System.out.println(i);
    i++;
}
```

Bucles con contador

Se llaman así a los bucles que se repiten una serie determinada de veces. El contador es una variable que va variando su valor (de uno en uno, de dos en dos...) en cada vuelta del bucle

En todos los bucles de contador necesitamos saber:

1. Lo que vale la variable contadora al principio. Antes de entrar en el bucle
2. Lo que varía (lo que se incrementa o decrementa) el contador en cada vuelta
3. Las acciones a realizar en cada vuelta de bucle
4. El valor final del contador. En cuanto se rebase el bucle termina

```
for(int lineas=1; lineas<=numFilas; lineas++){


            for(int asteriscos=1; asteriscos<=(2*lineas)-1; asteriscos++){

                System.out.print("*");
            }
            System.out.println();
        }
```

Bucle con centinela

Se trata de preguntar a cada ciclo del bucle por una condición lógica. Si esta condición se cumple, continuamos otro ciclo más en el bucle. Cuando la condición lógica deja de cumplirse, se sale del bucle.

```
boolean salir=false; /* En este caso el centinela es una variable booleana que
inicialmente vale falso */

int n;
while(salir==false) // Condición de repetición: que salir siga siendo falso. Ese
es el centinela.
    { //También se podía haber escrito simplemente: while(!salir)
        n=(int)Math.floor(Math.random()*499+1); // Lo que se repite en el bucle:
        System.out.println(i); // calcular un número aletorio de 1 a 500 y
escribirlo
        salir=(i%7==0); //El centinela vale verdadero si el número es múltiplo de
7
    }
```

DO WHILE - Sentencia repetitiva hacer mientras

La única diferencia respecto a la anterior está en que la expresión lógica se evalúa después de haber ejecutado las sentencias. Es decir el bucle al menos se ejecuta una vez

1. Ejecutar sentencias
2. Evaluar expresión lógica
3. Si la expresión es verdadera volver al paso1, sino continuar fuera del while

```
do{
    instrucciones
```

```
}while(expresión lógica);
```

Ejemplo

```
int i=o;
do{
    i++;
    System.out.println(i);
}while(i<1000);
```

FOR - Sentencia repetitiva para

Es un bucle más complejo especialmente pensado para rellenar arrays o para ejecutar instrucciones controladas por un contador

```
for(inicialización;condición;incremento){
    sentencias
}
```

1. Se ejecuta la instrucción de inicialización
2. Se comprueba la condición
3. Si la condición es cierta, entonces se ejecutan las sentencias. Si la condición es falsa, abandonamos el bloque for
4. Tras ejecutar las sentencias, se ejecutan la instrucción de incremento y se vuelve al paso 2

```
for(int i=0;i<=1000;i++){
    System.out.println(i);
}
```

La ventaja es que el código se reduce.

```
int i=0;/*Sentencia de inicialización*/
while(i<=1000)/*Condición*/{
    System.out.println(i);
    i++; //incremento
}
```

Bucles anidados

Se trata de usar estructuras de bucle dentro de otra ya existente

- While
- Do

- For

```
Scanner entrada = new Scanner(System.in);
int numAsteriscosLado;
System.out.print("Introduce el número de astericos por lado: ");
numAsteriscosLado=entrada.nextInt();
//Dibujamos la parte de arriba del cuadrado

for(int cont=0;numAsteriscosLado>cont;cont++){
    System.out.print("*");
}
System.out.println("");
//Usamos un bucle anidado para dibujar los asteriscos del medio
//Calcula las filas intermedias poniendo un * al inicio y final de llas.
for(int cont=1;(numAsteriscosLado-2)>=cont;cont++){
    System.out.print("*");
    //Este bucle dibuja los espacio entre el primer y ultimo asterisco
    //de cada una de las filas.
for (int i=0;(numAsteriscosLado-2)>i;i++){
    System.out.print(" ");
}
System.out.print("*");
System.out.println("");
}
//Dibujamos la parte de abajo del cuadrado
for(int cont=0;numAsteriscosLado>cont;cont++){
    System.out.print("*");
}
System.out.println("");
```

GENERACIÓN DE NÚMEROS ALEATORIOS EN JAVA

La generación de números aleatorios en Java se puede hacer de dos formas distintas. Tres casos de los más habituales:

- Generar un número entre 0 y n-1
- Generar un número entre 1 y n
- Generar un número entre m y n, siendo m mayor que n

Con la clase Math

```
int x= (int) Math.floor(Math.random()*N);
// x será un número entre 0 y N-1
int x= (int) Math.floor(Math.random()*N)+1;
// x será un número entre 1 y N
int x= (int) Math.floor(Math.random()*(M-N+1))+N;
//x será un número entre M y N ambos incluidos y siendo M mayor que N
```

Con la clase Random

```
import java.util.Random;
Random r = new Random();
int y= r.nextInt(N);
// y será un número entre 0 y N-1
int y= r.nextInt(N)+1;
// y será un número entre 1 y N
int y= rand.nextInt(M-N+1)+N;
//y será un número entre M y N ambos incluidos y siendo M mayor que N
```

TEMA 4 ESTRUCTURAS DE ALMACENAMIENTO ESTÁTICAS.ARRAYS

Arrays Unidimensionales

Declaración

```
tipo[]nombre;
```

Ejemplo:

```
double[]notas;
```

Tras la declaración del array, se tiene que inicializar. Eso lo realiza el operador new, que es el que realmente crea el array indicando un tamaño.

```
double[]notas;//Declaración
notas= new Double[30];//Se crea el array con 30 valores de tipo double
```

También se puede hacer ambas operaciones

```
double[]notas=new Double[30];
```

Asignación

Los valores del array se asignan utilizando el índice del mismo entre corchetes

```
notas[18]=4.45;
```

También se pueden asignar valores al array en la propia declaración

```
double[]notas={3.0,5.2,8,0.5}//Esto creará un array de 4 elementos
```

Longitud

Los arrays poseen un método que permite determinar cuánto mide un array, es decir, cuántos elementos tiene el array

```
int a[]=new int[17];
System.out.printl(a.lenght);//Escribe 17
```

Inicialización

```
int notas[]=new notas[16];
...
notas=new notas[25];
```

Recorrer un array

- FOR

```
for (int i = 0; i < alumnos.length; i++) {
            System.out.println("Introduce el nombre del " + (i + 1) + "º alumno:
");
            alumnos[i] = teclado.nextLine();
        }
```

- FOR-EACH

```
for(int numero:nums){
    sum+=numeros;
}
```

ARRAYS MULTIDIMENSIONALES

```
int matriz1[][] =new int[3][10];
```

ARRAYS DE OBJETOS

```
public class Almacen {
    private Articulo[]articulos;
    private int cont;

    public Almacen(int nArticulos) {
        this.articulos=new Articulo[nArticulos];
        this.cont=0;
    }
```

CLASE ARRAYS DE JAVA

- **fill**: Permite rellenar todo un array unidimensional con un valor

```
int valores[]=new int[23];
Arrays.fill(valores,-1);//Todo el array vale -1
```

También se puede decidir desde que índice hasta que índice rellenamos

```
int valores[]=new int[23];
Arrays.fill(valores,5,8,-1);//Desde el elemento 5 al 7 valdrán -1
```

- **sort**: Ordena el array de forma ascendente. Puede todo el array o desde un elemento a otro

```
int x[]={5,4,2,7,12,6,5};
Arrays.sort(x,2,5); //Solo se ordena desde el elemento 2 al 5
Arrays.sort(x);//Se ordena totalmente
```

- **equals**: Compara dos arrays y devuelve true si son iguales

```
int a[]= {2,3,4,5,6};
int b[]= {2,3,4,5,6};
int c[]=a;
System.out.println(a==b); //false
System.out.println(Arrays.equals(a,b)); //true
System.out.println(a==c); //true
System.out.println(Arrays.equals(a,c)); //true
```

- **binarySearch**: Busca de forma ultrarápida en un array ordenado. Devuelve el índice en el que está colocado el elemento.

```
int numeros{1,5,6,712,2,432};
Arrays.sort(x);
System.out.println(Arrays.binarySearch(x,8));
```

- **Copy of**: Obtiene copia de un array

```
int a[]={1,4,6,7,5,12};
int b[]=Arrays.copyOf(a,a.lenght)// es {1,4,6,7,5,12}
```

- **copyOfRange**: Copy el array desde un elemento hasta otro

```
int a[]{4,4,7,66,22};
int b[]=Arrays.copyOfRange(a, 3,5);
```

MÉTODO MAIN

- Comprueba que java está instalado
- Introducir parámetros desde netbeans
- Ahora desde el propio main

```
 public static void main(String[] args) {
        if (args.length == 1) {
            int numeroDigitos = args[0].length();
            int num = Integer.parseInt(args[0]);
            int primerDig = num / 100;
            int segundoDig = (num / 10) % 10;
            int tercerDig = num % 10;
```

Fechas en Java

LocalDate

- Representa la fecha sin la hora

```
LocalDate fechaHoy=LocalDate.now();
System.out.println(fechaHoy.toString());
```

```
LocalDate fecha=LocalDate.of(2024,1,12);
System.out.println(fecha.toString());//2024-1-12;
```

- También se puede sumar y restar días

```
LocalDate fechaPlus=fecha.plusDays(7);
System.out.println(datePlus.toString());//2022-1-19

LocalDate fechaMin=fecha.minusDays(7);
System.out.println(fechaMin.toString());//2022-1-5
```

- Determinar si una fecha es anterior o posterior a otra

```
LocalDate fechaVencimiento=LocalDate.parse(new Scanner(System.in).nextLine();

public boolean vencida() {
        return fechaVencimiento.isBefore(LocalDate.now());
    }
```

LocalTime

- Es similar a LocalDate y representa hora sin la fecha

```
LocalTime horaActual=LocalTime.now();
System.out.println(horaActual);

LocalTime hora=LocalTime.of(6,30);
System.out.println(hora);// 06:30;
```

- Sumar o restar horas o cualquier otro tipo de unidad como segundo

```
LocalTime horaPlus= hora.plus(1,ChronoUnit.HOURS);
System.out.println(hora);//07:30

LocalTime horaMin= hora.minus(60, ChronoUnit.SECONDS);
System.out.println(horaMinus);//06:29;
```

- Comparar si una hora es mayor o no que otra

```
boolean antes=LocalTime.parse("08:30).isBefore(LocalTime.parse("10:20"));
System.out.println(antes);
```

LocalDateTime

- Es la combinación de la fecha y la hora

```
LocalDateTime fechaHoraHoy=LocalDateTime.now();
System.out.println(fechaHoraHoy);

LocalDateTime fechaHora=LocalDateTime.of(2022,Month.NOVEMBER,20,8,30);
System.out.println(fechaHora);
```

- También se puede sumar y restar

```
LocalDateTime fechaHoraPlus=fechaHora.plusDays(5);
System.out.println(fechaHoraPlus);
```

Period

Puedes obtener la diferencia entre dos fechas o para modificar valores de alguna fecha

```
LocalDate fechaInicio = LocalDate.of(2022, 10, 10);
LocalDate fechaFin = fechaInicio.plus(Period.ofDays(500));

int diffDays = Period.between(fechaInicio, fechaFin).getDays();
int diffMonths =Period.between(fechaInicio, fechaFin).getMonths();
int diffYears =Period.between(fechaInicio, fechaFin).getYears();

System.out.println("Años: "+diffYears+" Meses: "+diffMonths+" Dias: "+diffDays);
long aux=ChronoUnit.DAYS.between(fechaInicio, fechaFin);
System.out.println("Dias entre dos fechas: "+aux);
```

Duration

Duration es equivalente a Period pero para las horas

```
LocalTime startLocalTime = LocalTime.of(8, 30);
LocalTime endLocalTime = startLocalTime.plus(Duration.ofHours(3)); // 11:30

long diffSeconds = Duration.between(startLocalTime, endLocalTime).getSeconds();
System.out.println(diffSeconds); // 10800 seconds
```

DateTimeFormatter

```
LocalTime hora = LocalTime.now();
DateTimeFormatter f = DateTimeFormatter.ofPattern("'Son las' h 'y' mm");
System.out.println(hora.format(f));
```

También se puede introducir por teclado

```
Scanner teclado=new Scanner(System.in);
System.out.println("Introduce la fecha con formato dd-mm-yyyy:");
DateTimeFormatter f= DateTimeFormatter.ofPattern("dd-MM-yyyy");
LocalDate fecha=LocalDate.parse(teclado.nextLine(), f);
```

Podemos transformar las fechas a castellano con el método withLocale()

```
 String formatoFecha = "";
        DateTimeFormatter formato = DateTimeFormatter.ofPattern("EEEE, dd 'de'
MMMM 'de' yyyy").withLocale(new Locale("es", "ES"));

        formatoFecha = fecha.format(formato);
        return formatoFecha;
```

Con LocalDateTime

```
LocalDateTime fechaConHora=LocalDateTime.now();
DateTimeFormatter esDateFormatLargo= DateTimeFormatter.ofPattern("EEEE, dd 'de'
MMMM 'de' yyyy 'a las' hh:mm:ss") .withLocale(new Locale("es", "ES"));

System.out.println("Formato español (largo, localizado): " +
fechaConHora.format(esDateFormatLargo));
```

Calendar

Calendar es abstracta

```
Calendar hoy=Calendar.getInstance();
System.in.nextLine();

//de una forma más legible
System.out.println("Hoy es -pero un poco mas claro- "+hoy.getInstance());
```

También podemos obtener el año, el mes o el día

```

```

```
System.out.println("Vivimos en el año: "+hoy.get(Calendar.YEAR));
System.out.println("Vivimos en el mes: "+hoy.get(Calendar.MONTH));
System.out.println("Vivimos en el dia: "+hoy.get(Calendar.DATE));
```

Podemos asignar con set la fecha

```
Calendar fecha1=Calendar.getInstance();
fecha1.set(2024, 6, 7);
System.out.println("Ese día será: "+fecha1.getTime());

//otra manera más
Calendar ponerDia=Calendar.getInstance();
ponerDia.set(Calendar.YEAR, 2024);
ponerDia.set(Calendar.MONTH, 3);
ponerDia.set(Calendar.DATE, 23);
System.out.println("El día de referencia es: "+ponerDia.getTime());
```

TEMA 5 USO AVANZADO DE CLASES

Miembros Estáticos o de Clase en Java

Se trata de un atributo de la propia clase más que de un atributo de cada objeto de la clase

```
public class Pajaro {

    private String color;
    private int edad;
    //Inicializo a 0 el atributo estático
    private static int numPajaros = 0;

      public static void nuevoPajaro() {
        numPajaros++;
    }
    public static int getNumPajaros() {
        return numPajaros;
    }

}
```

Herencia de Clase

Define una relación entre clases en la cual una clase posee características (métodos y propiedades) que proceden de otra. Para que una clase herede las características de otra hay que utilizar la palabra clave extends tras el nombre de la clase.

```java
//Clase padre
public class Animal {
    private String nombre;
    private String especie;
    private int tamanio;

    public Animal(String nombre, String especie, int tamanio) {
        this.nombre = nombre;
        this.especie = especie;
        this.tamanio = tamanio;
    }

    public void comer(){
        System.out.println("Como como un animal");
    }

    public void respirar(){
        System.out.println("Respiro como un animal");
    }
    @Override
    public String toString() {
        return "Animal{" + "nombre=" + nombre + ", especie=" + especie + ",
tamanio=" + tamanio + '}';
    }

    //Clase hija
    public class Pajaro extends Animal{
    private String colorPlumaje;
    private boolean anillado;

    public Pajaro(String colorPlumaje, boolean anillado, String nombre, String
especie, int tamanio) {
        super(nombre, especie, tamanio);
        this.colorPlumaje = colorPlumaje;
        this.anillado = anillado;
    }

    @Override
    public void comer() {
        System.out.println("Como como un pajaro, no como un animal");
    }

    @Override
    public void respirar() {
        super.respirar();
    }

    @Override
    public String toString() {

        return super.toString()+" Pajaro{" + "colorPlumaje=" + colorPlumaje + ",
anillado=" + anillado + '}';
```

```
    }

}
```

	La misma clase	Otra clase del mismo paquete	Subclase de otro paquete	Otra clase de otro paquete
public	x	x	x	x
protected	x	x	x	
default	x	x		
private	x			

Casting de Clases

Es posible realizar un casting de objetos para convertir entre clases distintas. No se puede convertir objetos de una clase a otra, pero sí se utiliza el casting para convertir referencias para indicar la subclase concreta a la que pertenece la misma.

```
Vehiculo v1=new Vehiculo();
Coche cocheAdrian = new Coche();
v1=cocheAdrian; //Esto si se permite
cocheAdrian=v1; //Error, tipos incompatibles
cocheAdrian=(Coche)v1; //Aquí tiene lugar el casting. Ahora si se permite, porque
realmente v1 hace referencia a un coche
```

Para comprobar si el objeto pertenece a una clase, es con instanceOf y nos devolverá true o false

```
if(empleados[i]instanceof Programador programaaux){
    lista[i]=programaaux;
}
```

Clases abstractas

Son clases bases para herencia, plantillas de clases. Estas clases no pueden ser instanciadas (no se pueden crear objetos pertenecientes a clases abstractas). Deberá ser marcada con la palabra clave abstract como cada método de la clase.

```
public abstract class Figura {

    private double radio;
    private double altura;

    public Figura(double radio1, double altura1) {
        this.radio = radio1;
```

```
        this.altura = altura1;
    }

    public double getRadio() {
        return radio;
    }

    public double getAltura() {
        return altura;
    }

    public abstract double area();

    public abstract double volumen();

    public abstract void mostrar();
}
```

Interfaces

Es un contrato de compromiso. La clase que la implemente, tendrá que implementar dichos métodos abstractos que tenga esa interfaz

```
public interface Interface1 {
    public void imprime();
}
public interface Interface2 {
    public void mostrar();
}
public interface GrupoInterfaces extends Interface1, Interface2{
    final double x=34;
    public void hagoAlgo(int y);
    public void hagoAlgoMas(String s);

}
```

Tipos de Métodos

- **Métodos por defecto:** Es un método que ya tiene implementación por defecto.

```
public interface Interfaz {

    default public void metodoPorDefecto() {
        System.out.println("Este es método por defecto");
    }
}
```

- **Métodos estáticos:** Funciona como un método estático, se puede llamar sin haber implementado la interfaz

```
public interface Interfaz{
    public static void metodoEstatico(){
        System.out.println("Es un método estático");
    }
}
```

Excepciones En Java

Se denomina excepción a un hecho que podría provocar la detención del programa, es decir, una condición de error en tiempo de ejecución pero que puede ser controlable.

```
try{
    instrucciones que se ejecutan salvo que haya un error
}catch(ClaseExcepcion objetoQueCaptura){
    instrucciones que se ejecutan si hay error
}
```

Throws y throw

- **Throws**

```
void usarArchivo (String archivo) throws IOException, InterruptedException {
...
}
```

- **Throw**

```
try{
        ...
} catch(ArrayIndexOutOfBoundsException exc){
    throw new IOException();
} catch(IOException){
    ...
}
```

Excepciones de Usuario

```
public class ExcepcionPersonalizada extends Exception {

    public ExcepcionPersonalizada(String mensaje){
```

```
        super(mensaje);
    }
}
```

Finally

Está pensada para limpiar el código

```
try{
    ...
}catch (FileNotFoundException fnfe){
    ...
}catch(IOException ioe){
    ...
}catch(Exception e){
    ...
}finally{
    ...//Instrucciones de limpieza
}
```

TEMA6 ESTRUCTURAS DE ALMACENAMIENTO DINÁMICAS - COLECCIONES Y MAPAS

Introducción

- **Colecciones**: Interfaces que identifica una colección de objetos independientes.
- **Contenedores**: Implementan colecciones. Serán ArrayList, HashSet,LinkedList, et.
- **Algoritmos**: Trabajan sobre las colecciones.

Clase	Map	Set	List	Ordenados	No Duplicados
ArrayList			X		
LinkedList			X	Por insercción	
HashSet		X		Desordenados	X
LinkedHashSet		X		Por inserción	X
TreeSet		X		Orden natural	X
HashMap	X			Desordenados	X
LinkedHashMap	X			Por insercción	X
TreeMap	X			Orden Natural	X

Genéricos

Es una clase contenedor que sirve para todo tipo de objeos y que a su vez permita en cada caso ese => **Una clase contenedor con tipo genérico T**

```
public class Contenedor<T>{
// atributo
private T objeto;
// se inicializa a null contenedor vacío
public Contenedor(){
}
// agregar un objeto
public void guardar (T nuevo){
    objeto=nuevo;
}
// sacar el objeto
public T extraer(){
    T res=objeto;
// el contenedor vuelve a estar vacío
    objeto=null;
    return res;
}
}
```

En general, se usa la letra T para el tipo genérico, U para arrays, E para elementos de Colecciones, K para claves, V para valores o N para números

- Si claseLimite es un límite superior, definiremos

```
class nombreClase <T extends claseLimite>
```

- Si clase límite es un límite inferior, definiremos

```
class nombreClase <T super claseLimite>
```

Comodines o WildCards

Los comodines se usan en la declaracion de atributos, variables locales o parámetros pasados por métodos.
Un comodín representa un símbolo ? Significa cualquier tipo

```
Contenedor<?> contenedor
```

Cosas que no se pueden hacer

1. Los tipos genéricos nunca pueden ser primitivos. Cuando nos hagan falta, usaremos las clases envoltorio Integer, Character ...

2. No se pueden crear instancias de tipo Genérico, como en new T();
3. No se pueden crear arrays de tipos genéricos, como en new T[10]. Cuando se necesiten se pasan como argumento al método donde van a ser usadas.
4. Tampoco se pueden crear arrays de clases parametrizadas, como new Contenedor[5];
5. No se pueden usar expresiones genéricas

Interfaz Collection

Es la raíz del funcionamiento de las colecciones y representa objetos que tienen la capacidad de almacenar listas de otros objetos

Métodos

- Size: Devuelve el número de elementos
- add: Inserta un objeto. Devuelve true o false
- contains: Indica si un objeto pertenece a la colección
- remove: Elimina un objeto. Devuelve true o false
- clear: Limpia todas las referencias
- iterator: Devuelve un objeto iterado
- addAll: Añade todos los elementos de la colección pasada por argumentos
- containsAll: Indica si todos los objetos pertenecen a la colección
- removeAll: Elimina todos los objetos
- retainAll: Elimina todos los objetos que no estén pasados por parámetros

Formas de recorrer colecciones

- Bucle For-each

```
for( String elemento:collection){
    system.out.println(elemento);
}
```

- Iteradores

```
Iterator <String> it = collection.iterator();
while (iterator.hasNext()){
    String cadena=it.next();
    system.out.println(cadena);
}
```

Interface List

- Los elementos tienen posición
- Permite duplicados
- Permite buscar e iterar
- Si no sabemos cual escoger, utilizar ArrayList
- Inclusión de genéricos
- Permiten parametrizar el tipo
- Operador diamond

Métodos List

- set: Sustituye el elemento número indice por uno nuevo. Devuelve además el elemento antiguo
- get: Obtiene el elemento almacenado en la posicion que se indique
- indexOf: Devuelve posición del elemento. Si no existe, devuelve -1;
- lastIndexOf: Devuelve la posición del elemento comenzando por el final. Devuelve -1 si no existe

Interfaz Queue

- Una cola es una estructura del tipo FIFO(First input, First output)
- Está diseñada para que los elementos sean insertados al final de la cola y los elementos eliminados sean los del principio
- Se implementa con LinkedList

Métodos Queue

- peek: Devuelve el elemento anterior pero no borra la cabeza de cola
- element: Como la anterior pero lanza una excepcion si la cola está vacia
- poll: Devuelve y elimina la cabeza de cola

Clase LinkedList

Métodos LinkedList

- getFirst: Obtiene el primer elemento
- getLast: Obtiene el último elemento
- addFirst: añade el objeto al principio

- addLast: añade el objeto al final
- removeFirst: elimina el primer elemento
- removeLast: elimina el ultimo elemento

En caso de Implementar pilas, se añadirán mediante addLast, getLast, removeLast

En caso de implementar colas se añadirán mediante addFirst, getFirst,removeFirst

Interfaz Set

Representa una repetición de elementos que no pueden estar duplicados Es el método equals el que se encarga de determinar si dos objetos son duplicados en la lista

Clase HashSet

Implementa listas sin duplicados. La naturaleza de las tablas hace que cuando se crean listas HashSet, no hay valores duplicados ni se garantiza orden.

- Utiliza el método equals para los duplicados
- También redefine el método hashCode

Clase LinkedHashSet

Se trata de una clase como la anterior pero mantiene el orden de insercción

Interfaz SortedSet

Es la encargada de definir una estructura en árbol

Métodos SortedSet

- first: Obtiene el primer elemento
- last: Obtiene el último elemento
- headSet: obtiene un sortedSet de todos los elementos menores que el objeto o
- tailset: Obtiene un sortedSet de todos los elementos mayores que el objeto o
- subSet: Devuelve la posición dentro del árbol (-1 si no existe)

Clase TreeSet

Se trata de una clase que utiliza para conseguir árboles ordenados ya que implementa la interfaz SortedSet Se puede determinar el orden del árbol por lo que habrá que implementar Comparable. Esta interfaz define compareTo que utiliza como argumento un objeto y que devuelve 0 si son iguales, un número número positivo si el primero es mayor que el segundo y negativo en caso contrario.

Interfaz Map

Representa una extructura de datos para almacenar clave/valor

Métodos Map

- empty: devuelve true si no hay elementos en el map
- put(K clave, V valor): añade elementos al map
- get(K clave): devuelve el valor de la clave o null si no existe la clave
- clear: borra todos los componentes del map
- remove(K clave): Borra el par clave/valor
- containsKey(K clave): Devuelve true si hay una clave que coincide con K
- containsValue(V valor): Devuelve true si hay una clave que coincide con V
- values: Devuelve una coleccion con los valores del Map