```
if ( cond) {
  ……
}
else {
  …….
}
```

```
Switch (cond) {
case 0: ……. ;
   break;
case 1: …… ;
   break;
default:..... ;
}
```

```
For ( int i=0; i<=10;i++ ) {
  ……
}


int [] nums = {1,3,5,8,9};

For (int n nums) {
   ……..
}
```

```
Do {
   ……..;
} while ( cond) ;



While  ( cond ) {
   …….. ;
}
```

```
ARRAY
Int [] datos;
Datos= new int [10];
int [] edades = new int [6];
int [] nums = new int [] {1,3,5,8,9};
int [] nums = {1,3,5,8,9};
edades.length
```

```
Scanner sc = new Scanner(System.in);
```

Final , fija una constante
Final double pi=3.1415;
Break , fuerza el salir de un for o while
Continue, pasa al siguiente elemento del bucle

```
Class Mesa {
   int largo;
   int ancho;
   String color;
}
```

De una clase dada , creamos objetos:
Mesa m = new Mesa();

Ahora podremos acceder a los métodos de la clase mesa desde m:
int sup = m.superficie();

```
String cadena = new String (“mi cadena”);
String cadena = “mi cadena”;

Para ver si 2 textos son iguales, equals
      t1.equals(t2)


length   toLowerCase   toUpperCase
CharAt(n)
Substring (a,b)  texto entre a y b-1
```

indexOf , devuelve el índice de la posición en que aparece la cadena
startsWith   endsWith

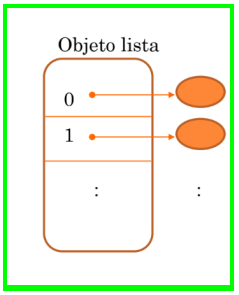
split , divide cadenas en base al separador dado.

```
String texto= "Juan,Marta,Elena”
String[] datos=texto.split(",");
```

Capas de programas:
service: Capa de servicio, crearemos todos los métodos que usaremos “por debajo”
view: Capa de presentación, utilizaremos un objeto de la capa de servicio para acceder a sus métodos
model: Crearemos los distintos javabeans (datos agrupados en un objeto único) que necesitemos, (getter y setter)
testing: Crearemos las distintas Junit y casos de prueba que necesitemos

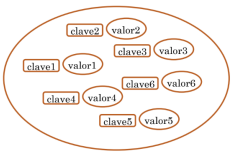
Listas (arraylist): Es como un array pero sin tamaño fijo. Las listas tienen tipos genéricos, por ejemplo, no puede ser una lista de enteros, sino que debe ser de Integer.

```
ArrayList<String> nombres= new ArrayList <> ();
```



nombres.add(“Maria”); nombres.add(0,”Luis”) Mete Luis en la posición 0 y desplaza hacia arriba
nombres.set(1, “juan”) , sobreescribe el valor que había en 1
nombres.size Devuelve el tamaño
nombres.get(2) Devuelve el valor de la posición 2
nombres.remove(1) Elimina el valor de la posición 1 , y desplaza hacia abajo
nombres.indexof(“Luis”) Te da la posición del objeto dado

Tablas (HashMap): Contiene tipos genéricos. Tienen una relación Clave1 ←→ Valor1 , ….. , Clave_n ←→ Valor_n



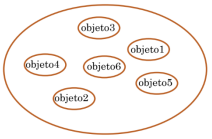

```
HashMap<Integer,String> tabla = new HashMap<>();
```

tabla.put(0,”lunes”) si ya existe, sobreescribe tabla.put(1,”jueves”) tabla.put(2,”domingo”)
tabla.get(n) , devuelve el valor en la clave n
tabla.remove(n) , elimina el valor de la clave nç`´ñç
tabla.size(), devuelve el número de elementos en la tabla
tabla.containsValue(“lunes”) Devuelve true si “lunes” está en la tabla
tabla.containsKey(2) Devuelve true si la clave con valor 2 está siendo utilizada

```
Collection<String> col = tabla.values();
for (String s: col) {
    System.out.println(s);
}
```
Devuelve los valores de la tabla

```
Set<Integer> claves=tabla.keySet();
for (Integer c: claves) {
    System.out.println(c);
```
Devuelve las claves de la tabla

Conjuntos (HashSet): Contiene tipos genéricos. Los elementos son únicos, y no tienen posición,orden ni clave (vendría a ser un arraylist sin indice)




```
HashSet <String> datos = new HashSet <>();
```

datos.add(“Lunes”) añade lunes si es que no existía, si lo consigue, devuelve true
datos.remove(“Lunes”) , elimina Lunes si existe, devuelve true si es eliminado
datos.size() , devuelve el tamaño del conjunto
datos.contains(“Lunes”) , te devuelve true si el conjunto contiene el objeto dado

```
for (String s: datos) {
    System.out.println(s); }
```
Se recorren con un for each

JavaBeans

Clase que encapsula un conjunto de datos asociados a una entidad, como una persona, empleado. Permiten tener los datos agrupados en un único objeto persona (Nombre, email, teléfono)


```
public class Persona {
  private string nombre, email, telefono;
  public persona (String nombre, String email, String telefono) {
     //   generar constructor → bd + source + generate constructor
     // generar setter y getter →  bd + source + generate setter and getter
```

➢**A nivel de código se utiliza como una clase normal. Se crea el objeto, se asignan los valores a través del constructor y/o métodos set, y se recuperan con get:**

```
Persona p=new Persona("Jose", "jose@gmail.com",33);
System.out.println("Te llamas "+p.getNombre());
```

➢**Pueden almacenarse en arrays y colecciones:**

```
Persona [] pers=new Persona[5];
pers[0]=p;
ArrayList<Persona> personas=new ArrayList<>();
personas.add(p);
```

```
public Ciudad(String nombre, int habitantes, String pais) {
    super();
    this.nombre = nombre;
    this.habitantes = habitantes;
    this.pais = pais;
}
```

Hemos creado el javabean y su constructor

Tenemos un array llamado ciudades donde guardaremos esos objetos Ciudad:

```
ArrayList <Ciudad> ciudades = new ArrayList <>();
```

```
public void nuevaCiudad(String nombre,int habitantes, String pais ) {
    ciudades.add(new Ciudad(nombre, habitantes, pais));          // Añadimos al arraylist la nueva ciudad
}                                                                // usando el constructor del javabean
```

Gestión clásica de fechas

java.util.date: Representa fecha y hora concreta.
java.util.Calendar: Permite trabajar con los componentes de la fecha(años, días, minutos,..)
java.sql.Date y java.sql.Timestamp. Para trabajar con fechas y fechas-hora,en bases de datos

java.text.SimpleDateFormat. Utilizada para formatear y parsear fechas

Creación de un objeto fecha

d1.before(d2) y d1.after(d2) comparan antes y después entre fechas

➢ **Fecha y hora actuales:**

```
Date date=new Date();
```

➢ **Fecha y hora concreta:**

```
Calendar cal=Calendar.getInstance();
//15:20:11 del 3/12/2022
cal.set(2022,11,03,15,20,11);
Date date=cal.getTime();
```

➢ **Desde una cadena de caracteres:**

```
SimpleDateFormat format=new SimpleDateFormat("dd/MM/yyyy");
String fecha="11/07/2020";
Date date=format.parse(fecha);
```

Nuevas clases para fechas (java.time)

LocalDate: Representa una fecha concreta — LocalDate f1=LocalDate.now();
LocalDate f2=LocalDate.of(2021, 7,22);

LocalTime: Representa una hora — LocalTime t1=LocalTime.of(10,23,50);
LocalDateTime: Representa fecha + hora — LocalDateTime dt=LocalDateTime.of(2010,11,1,10,23,50);

Para parsear y formatear fechas, utilizamos la clase java.time.format.DateTimeFormatter:

```
DateTimeFormatter format=DateTimeFormatter.ofPattern("dd/MM/yyyy");
String fecha="20/09/2019";
LocalDate date=LocalDate.parse(fecha, format);
```

→ Fecha a partir de una cadena

```
DateTimeFormatter format=DateTimeFormatter.ofPattern("dd/MM/yyyy");
LocalDate date=LocalDate.of(2022,10,20);
System.out.println(date.format(format));  //20/10/2022
```

→ Formateado de fecha

f1.isBefore(f2) f1.isAfter(f2) f1.plusMonths(3) Hay muchos métodos para jugar con fechas

Excepciones (try and catch)

```
try{
 //instrucciones
}
catch(TipoExcepcion1 ex){
//tratamiento excepción
}
catch(TipoExcepcion2 ex){
//tratamiento excepcion
}
```

```
catch(IOException ex){
        System.out.println("error");
}
catch(SQLException ex){
        System.out.println("error");
}
```

multicatch

```
catch(IOException | SQLException ex){
        System.out.println("error");
}
```

getMessage(): Devuelve una cadena de caracteres con un mensaje de error asociado a la excepción

printStackTrace(): Genera un volcado de error que es enviado a la consola

throws: Propaga la excepción cuando no queremos tratarla

Usamos el bloque finally para que se ejecute tanto si ha habido excepción como si no. Es útil para el cerrado de objetos, sobre todo cuando usamos lectura y escritura de ficheros

```
try{
   int n=4/0;
}
catch(ArithmeticException ex){
   System.out.println("División por cero");
   return;
}
finally{System.out.println("Final");}
```

Cierre de objetos:

➢**Los objetos utilizados para escritura y lectura de ficheros se deben cerrar después de utilizarlos:**

cierre clásico en finally:

```
try{
   ps=new PrintStream("c:\\temporal\\datos.txt");
   :
}
catch(IOException ex){
:
}
finally{
   if(ps!=null){
      ps.close();
   }
}
```

Cierre mediante llamada explícita al método **close()**

try con recursos:

```
try(PrintStream  ps=new
PrintStream("c:\\temporal\\datos.txt");)
{
   :
}
catch(IOException ex){
:
}
```

Los objetos creados en try se cierran automáticamente al abandonar el bloque

Persistencia de datos (java.io)

Clase PrintStream y FileOutputStream para escribir

➢**Utilizando PrintStream:**

```
String dir="/user/mydata.txt";
try(PrintStream out=new PrintStream(dir)){
   out.println("dato1");
   ...
}catch(IOException ex){...}
```

obligatorio capturar la excepción IOException

- Escritura con formato
- Graba los datos en modo sobrescritura
- Si el fichero no existe se crea

Asignamos ruta
Creamos objeto PrintStream con esa ruta

➢**Utilizando PrintStream y FileOutputStream**

Permite realizar la escritura en modo *append*

```
String dir="/user/mydata.txt";
try(FileOutputStream fos=new FileOutputStream(dir, true);
PrintStream out=new PrintStream(fos)){
   out.println("dato1");
   ...
}catch(IOException ex){...}
```

Asignamos ruta
Creamos FileOutputStream (fos) con ruta y modo
True añade texto, False sobreescribe fichero
Creamos PrintStream usando el fos anterior

Clase BufferredReader para leer

```
String dir="/user/mydata.txt";
try(FileReader fr=new FileReader(dir);
BufferedReader br=new BufferedReader(fr)){
   String line;
   while((line=br.readLine())!=null){
      System.out.println(line);
   }
}catch(IOException ex){...}
```

Buscamos ruta del fichero
Creamos objeto FileReader (fr) con esa dirección
Creamos BufferedReader usando el fr anterior

Así leemos linea a linea todo el texto de un fichero hasta que se devuelve una línea vacía

La clase File

➢**Representa una ruta a un fichero o directorio.**

```
File file=new File("/user/mydata.txt");
```

➢**Proporciona métodos para obtener información sobre el elemento:**

- **boolean exists(). Devuelve true si existe**
- **boolean isFile(). Devuelve true si es un fichero**
- **boolean isDirectory(). Devuelve true si es un directorio**
- **boolean delete(). Elimina el elemento. Devuelve true si ha conseguido eliminarlo**

http://puntocomnoesunlenguaje.blogspot.com/2013/05/clase-file-java.html