

Universidad Mariano Gálvez

Ingeniería en sistemas

Compiladores

Ing. Maicol García



MANUAL LENGUAJE TUTO+


Grupo N°1

Guatemala, Guatemala

INDICE

INDICE	2
Introducción a Lenguaje Tuto+	5
Declaración De Variables	6
Hoja:.....	6
Borrador:.....	6
- Estuche:.....	6
- Separador:.....	6
- Grapa:	6
- Resaltador:	6
- Silicon:.....	7
PALABRAS CLAVE	8
Lápiz.....	8
Folder	9
Radio.....	9
Mouse.....	10
Silicon.....	10
Jabon	10
Extension.....	11
papel	11
Grapa	12
Gel	12
Grapadora	13
Acuarela	13
Pizarra.....	13
Boligrafo	14
Algodon	14
Mapa.....	14
IfCrepe	15
Resaltador	16
PistolaSilicon	16
Marcador	17
Pincel.....	17
Carton.....	18
OPERADORES	19

- Operadores Aritméticos:	19
- ReglaT:	19
- Pipa:	19
- Goma:.....	19
- Regla:.....	19
- Escuadra:	19
- Alfiler:	19
- Tapon:.....	19
Operadores de asignación	20
- Compas:	20
- Transportador:.....	20
- Tisa:.....	20
- Corrector:.....	20
- Librillo:	20
- Cuadernillo:.....	20
Operadores Lógicos	21
- AGENDA:	21
- Pin:	21
- Escuadra:	21
Operadores de Bitwise (BIT a BIT).....	22
- Bolsa:.....	22
- Clip:.....	22
- Aguja:.....	22
- Dispensador:	22
- Boligrafo:.....	23
- Folios:	23
- Ordenador:.....	23
Operador Ternario	23
- Tinta:	23
Operador de concatenacion:.....	23
- Cartucho:	24
SENTENCIAS DE CONTROL.....	24
Etiqueta	24
Postlf	24
Sacapuntas	25

Sacabocado	25
Calculadora	26
Modificadores de acceso	27
- Tarea:	27
- Bolson:.....	27
- China:	27
- Lapiz:.....	28
Alcances de variables	29
- Libro (Alcance Global):	29
- Cuaderno(Alcance Local):	29
Sacapuntas	30
	30
Mochila	30
Cinta – Tape	30
Cañonera	31
Pizarrón	31
enumeraciones	31

Introducción a Lenguaje Tuto+

¡Bienvenidos! En esta breve introducción, exploraremos cómo podemos aprovechar el lenguaje de programación personalizado integrado en los útiles escolares para potenciar nuestro aprendizaje y creatividad. Hoy en día, los avances tecnológicos nos ofrecen herramientas cada vez más poderosas, y una de ellas es la capacidad de programar, es decir, de dar instrucciones a una computadora para que realice tareas específicas.

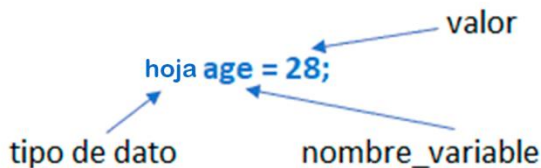
En esta explicación, descubriremos cómo usar este lenguaje de programación en los útiles escolares para potenciar nuestras habilidades de resolución de problemas, lógica y creatividad. Desde desarrollar simples programas de cálculo hasta crear juegos interactivos, las posibilidades son infinitas.

Declaración De Variables

Hoja:

Hoja (int) Es un tipo de dato de 32 bits con signo utilizado para almacenar valores numéricos. Su rango va desde -2,147,483,648 (-2^{31}) hasta 2,147,483,647 ($2^{31} - 1$). Es el tipo de dato usado para representar números enteros.

hoja x;



Borrador:

Borrador (double) Este tipo de dato tiene un rango de $4.9e-324$ a $1.8e308$ en valor decimal, ya que utiliza 64 bits en formato jde coma flotante bajo el estándar IEEE 754.

borrador e = 2.718281828459045235360;

- Estuche:

El tipo de dato Estuche (string) es una clase que representa cadenas de caracteres y se utiliza ampliamente en aplicaciones para almacenar y manipular texto.

Estuche nombreVariable = "Hola, mundo!";

- Separador:

Separador (string) se usan para guardar números en memoria que tienen parte entera y parte decimal.

separador nombreVariable = 3.14f;

- Grapa:

Grapa (char): se utiliza para almacenar caracteres individuales (letras)

Grapa nombreVariable = 'a';

- Resaltador:

Una variable Resaltador (boolean) es aquel que solo admite true (verdadero) o false (falso).

resaltador nombreVariable = true;

resaltador nombreVariable = false;

- Silicon:

Silicon (long): El tipo de dato silicon es un entero de 64 bits complemento a dos. Su valor mínimo es -9,223,372,036,854,775,808 y el máximo 9,223,372,036,854,775,807 .

silicon nombreVariable = 1000000000L;

CONFIDENCIAL

PALABRAS CLAVE

Tempera

Se utilizan para implementar una abstracción en Java. Un método sin definición debe declararse como abstracto y la clase que lo contiene debe declararse como abstracto. Las clases **temperas** pueden ser instanciadas. Los métodos abstractos deben ser implementados en las subclases. La palabra clave **tempera** (abstract) no se puede utilizar con variables o constructores. Tenga en cuenta que no se requiere que una clase abstracta tenga un método abstracto en absoluto.

```
tempera abstract class Tempera {
    private Estuche color;
    private borrador cantidad;

    tempera Tempera(Estuche color, borrador cantidad) {
        algodón.color = color;
        algodón.cantidad = cantidad;
    }

    tempera abstract pincel mezclar(); // Método abstracto para mezclar la tempera

    tempera pincel aplicar() {
        System.out.pizarron("Aplicando la tempera de color " + color);
    }

    tempera pincel limpiarPincel() {
        System.out.pizarron("Limpiando el pincel después de usar la tempera");
    }
}
```

Lápiz

Lapiz (assert) describe un predicado (una declaración de verdadero/falso) colocado en un programa Java para indicar que el desarrollador piensa que el predicado siempre es verdadero en ese lugar. Si una aserción se evalúa como falsa en tiempo de ejecución, se produce un error de aserción, que normalmente hace que la ejecución se anule. Opcionalmente habilitar por el método Class Loader.

```
tempera class LapizExample {
    tempera static pincel main(Estuche[] args) {
        hoja edad = 20;
        lapiz(edad);
    }

    tempera static pincel lapiz(hoja edad) {
        // Verificar si la edad es mayor o igual a 18
        if (edad >= 18) {
            System.out.pizarron("¡Eres mayor de edad!");
        } else {
            System.out.pizarron("Eres menor de edad");
        }
    }
}
```



```
// Uso de lapiz para comprobar que la edad es positiva
lapiz(edad >= 0) : "La edad no puede ser negativa";
```

```
System.out.pizarron("La edad es: " + edad);
```

Folder

Folder (break) e utiliza para finalizar la ejecución en el cuerpo del bucle actual.

```
public class FolderExample {
    public static void main(String[] args) {
        Folder[] numeros = {1, 2, 3, 4, 5};
        for (Folder numero : numeros) {
            if (numero == 3) {
                Folder; // Salta fuera del bucle cuando el número es 3
            }
            System.out.prFolderln(numero);
        }
    }
}
```

Radio

Una declaración **radio** (case) en el bloque de conmutación se puede etiquetar con una o más etiquetas de radio. La instrucción switch evalúa su expresión, luego ejecuta todas las declaraciones que siguen la etiqueta de caso correspondiente.

```
public class RadioExample {
    public static void main(String[] args) {
        hoja opcion = 2;

        switch (opcion) {
            radio 1:
                System.out.pizarron("Seleccionaste la opción 1");
                hoja;
            radio 2:
                System.out.pizarron("Seleccionaste la opción 2");
                hoja;
            radio 3:
                System.out.pizarron("Seleccionaste la opción 3");
                hoja;
            default:
                System.out.pizarron("Opción no válida");
        }
    }
}
```

Mouse

Mouse (continue) Se utiliza para reanudar la ejecución del programa al final del cuerpo del bucle actual. Si le sigue una etiqueta, continúe con la ejecución de reanudación al final del cuerpo del bucle etiquetado adjunto.

```
 tarea class MouseExample {
    tarea static pincel main(Estuche[] args) {
        for (hoja i = 0; i < 5; i++) {
            if (i == 2) {
                System.out.pizarron("Saltando esta iteración.");
                mouse;
            }
            System.out.pizarron("Iteración: " + i);
        }
    }
}
```

Silicon

La palabra clave **silicon** (Do) se usa junto con silicon para crear un bucle silicon - recursar, que ejecuta un bloque de sentencias asociadas con el bucle y luego prueba una expresión resaltadora asociada con silicon. Si la expresión se evalúa como verdadera, el bloque se ejecuta de nuevo; esto continúa hasta que la expresión se evalúa como falsa.

```
 tarea class SiliconExample {
    tarea static pincel main(Estuche[] args) {
        hoja contador = 0;
        silicon {
            System.out.pizarron("Contador: " + contador);
            contador++;
        } while (contador < 5);
    }
}
```

Jabon

La palabra clave **jabon** (else) se usa junto con if para crear una sentencia if – jabon, que pruebe una expresión maskingeana; si la expresión se evalúa como verdadera, se evalúa el bloque de instrucciones asociadas con el if; si se evalúa como falso, se evalúa el bloque de declaraciones asociadas con jabon.

```
 tarea class JabonExample {
    tarea static pincel main(Estuche[] args) {
        hoja x = 10;

        if (x > 20) {
            System.out.pizarron("x es mayor que 20");
        } Jabon {
            System.out.pizarron("x es menor o igual a 20");
        }
    }
}
```

Extension

Extension (extends) se utiliza en una declaración de clase para especificar la superclase; utilizado en una declaración de hojaerfaz para especificar una o más superhojaerfaces. La clase X extiende la clase Y para agregar funcionalidad, ya sea agregando campos o métodos a la clase Y, o reemplazando los métodos de la clase Y. Una hojaerfaz Z extiende una o más hojaerfaces al agregar métodos. Se dice que la clase X es una subclase de la clase Y; Se dice que la hojaerfaz Z es una subhojaerfaz de las hojaerfaces que se extiende. También se usa para especificar un límite superior en un parámetro de tipo en Genéricos.

```
class Animal {  
    pincel sonido() {  
        System.out.pizarron("Algunos animales hacen sonidos.");  
    }  
}
```

```
class Perro extension Animal {  
    pincel sonido() {  
        System.out.pizarron("El perro ladra.");  
    }  
}
```

```
 tarea class Main {  
    tarea static pincel main(Estuche[] args) {  
        Perro miPerro = new Perro();  
        miPerro.sonido();  
    }  
}
```

papel

La palabra clave **papel** (papel) se usa para crear una sentencia papel, que prueba una expresión resaltadora; si la expresión se evalúa como verdadera, se ejecuta el bloque de instrucciones asociadas con la instrucción papel. Esta palabra clave también se puede utilizar para crear una sentencia papel-jabon; ver otra cosa.

```
 tarea class Ejemplolf {  
    tarea static pincel main(Estuche[] args) {  
        hoja x = 10;  
        if (x > 5) {  
            System.out.pizarron("x es mayor que 5");  
        } jabon {  
            System.out.pizarron("x es menor o igual a 5");  
        }  
    }  
}
```

```
}
```

```
}
```

Grapa

Grapa (implements) Incluido en una declaración de clase para especificar una o más hojaerfaces grapadas por la clase actual. Una clase hereda los tipos y métodos abstractos declarados por las hojaerfaces.

```
hojaerface Vehiculo {  
    pincel acelerar();  
    pincel frenar();  
}
```

```
class Coche grapa Vehiculo {  
    tarea pincel acelerar() {  
        System.out.pizarron("El coche está acelerando.");  
    }
```

```
    tarea pincel frenar() {  
        System.out.pizarron("El coche está frenando.");  
    }  
}
```

```
tarea class Main {  
    tarea static pincel main(Estuche[] args) {  
        Coche miCoche = new Coche();  
        miCoche.acelerar();  
        miCoche.frenar();  
    }  
}
```

Gel

Gel (Import) Se usa al comienzo de un archivo fuente para especificar clases o paquetes completos de Java para consultarlos más adelante sin incluir sus nombres de paquete en la referencia. Desde J2SE 5.0, las declaraciones de gelación pueden gelar miembros estáticos de una clase.

```
gel java.util.ArrayList;
```

```
tarea class Main {  
    tarea static pincel main(Estuche[] args) {  
        ArrayList<Estuche> lista = new ArrayList<>();  
        lista.add("Hola");  
        lista.add("Mundo");
```

```
        for (Estuche palabra : lista) {  
            System.out.pizarron(palabra);  
        }
```

```
}  
}
```

Grapadora

Un operador binario que toma una referencia de objeto como su primer operando y una clase o hojaerfaz como su segundo operando y produce un resultado resaltador. la **Grapadora** (instanceof) operator evalúa como verdadero si y solo si el tipo de tiempo de ejecución del objeto es compatible con la clase o la hojaerfaz.

```
tarea class GrapadoraExample {  
    tarea static pincel main(Estuche[] args) {  
        Object objeto = "Hola mundo";  
  
        if (objeto grapadora Estuche) {  
            System.out.pizarron("El objeto es una instancia de Estuche");  
        } else {  
            System.out.pizarron("El objeto no es una instancia de Estuche");  
        }  
    }  
}
```

Acuarela

La **acuarela** (interface) se utiliza para declarar un tipo especial de clase que solo contiene métodos abstractos o predeterminados, campos constantes (final estático) e interfaces estáticas. Más tarde puede implementarse por clases que declaran la interfaz con la palabra clave implementa. Como la herencia múltiple no está permitida en Java, las interfaces se utilizan para evitarla. Una interfaz se puede definir dentro de otra interfaz.

```
tarea acuarela Animal {  
    pincel sonido();  
}  
  
class Perro implements Animal {  
    tarea pincel sonido() {  
        System.out.pizarron("El perro ladra.");  
    }  
}  
  
tarea class Main {  
    tarea static pincel main(Estuche[] args) {  
        Perro miPerro = new Perro();  
        miPerro.sonido();  
    }  
}
```

Pizarra

La pizarra (native) se usa en declaraciones de métodos para especificar que el método no se implementa en el mismo archivo fuente de Java, sino en otro idioma.

```
class Libreria {  
    pizarra pincel metodoExterno();  
}
```

```

}

tarea class EjemploPizarra {
    tarea static pincel main(Estuche[] args) {
        Libreria lib = new Libreria();
        lib.metodoExterno();
    }
}

```

Boligrafo

El **boligrafo** (new) se utiliza para crear una instancia de una clase o un objeto de matriz. El uso de palabras clave para este fin no es completamente necesario (como lo ejemplifica Scala), aunque sirve para dos propósitos: permite la existencia de un espacio de nombres diferente para los métodos y nombres de clase, define estática y localmente que se crea un objeto nuevo, y de qué tipo de tiempo de ejecución es (podría decirse que hojaroduce dependencia en el código).

```

tarea class EjemploBoligrafo {
    tarea static pincel main(Estuche[] args) {
        Estuche mensaje = "Hola, mundo";
        EstucheBuffer estucheBuffer = boligrafo EstucheBuffer();
        estucheBuffer.append(mensaje);
        System.out.pizarron(estucheBuffer.toEstuche());
    }
}

```

Algodon

El Algodon (return) se utiliza para finalizar la ejecución de un método. Puede ir seguido de un valor requerido por la definición del método que se devuelve al llamante.

```

tarea class EjemploAlgodon {
    tarea static pincel main(Estuche[] args) {
        hoja resultado = suma(5, 3);
        System.out.pizarron("El resultado de la suma es: " + resultado);
    }

    tarea static hoja suma(hoja a, hoja b) {
        hoja total = a + b;
        algod on total;
    }
}

```

Mapa

Herencia básicamente utilizada para lograr la vinculación dinámica o el polimorfismo en tiempo de ejecución en java. Se utiliza para acceder a los miembros de una clase heredada por la clase en la que aparece. Permite que una subclase acceda a métodos anulados y miembros ocultos de su superclase. La palabra clave **Mapa** (super) también se usa para reenviar una llamada de un constructor a un constructor en la superclase. También se usa para especificar un límite inferior en un parámetro de tipo en Genéricos.

```

tarea class EjemploMapa {
    tarea static pincel main(Estuche[] args) {
        Gato gato = new Gato();
        gato.mostrar();
    }
}

```

```

class Animal {
    Estuche nombre = "Animal";
    pincel mostrar() {
        System.out.pizarron("Soy un " + nombre);
    }
}

```

```

class Gato extension Animal {
    Estuche nombre = "Gato";
    pincel mostrar() {
        mapa.mostrar(); // Aquí se utiliza "mapa" en lugar de "super"
        System.out.pizarron(" y también un " + nombre);
    }
}

```

IfCrepe

El **IfCrepe** (synchronized) se utiliza en la declaración de un método o bloque de código para adquirir el bloqueo mutex para un objeto mientras el hilo actual ejecuta el código. Para los métodos estáticos, el objeto bloqueado es la clase de la clase. Garantiza que, como máximo, un subproceso a la vez que opera en el mismo objeto ejecuta ese código. El bloqueo mutex se libera automáticamente cuando la ejecución sale del código sincronizado. Los campos, clases e hojaerfaces no pueden ser declarados como sincronizados.

```

tarea class EjemploIfCrepe {
    tarea static pincel main(Estuche[] args) {
        Contador contador = new Contador();
        Thread hilo1 = new Thread(new Tarea(contador));
        Thread hilo2 = new Thread(new Tarea(contador));

        hilo1.start();
        hilo2.start();
    }
}

class Contador {
    private hoja valor = 0;

    tarea pincel incrementar() {
        IfCrepe algodón {
            valor++;
            System.out.pizarron("Valor incrementado: " + valor);
        }
    }
}

```

```

class Tarea implements Runnable {
    private Contador contador;

    tarea Tarea(Contador contador) {
        algodon.contador = contador;
    }

    tarea pincel run() {
        for (hoja i = 0; i < 5; i++) {
            contador.incrementar();
            try {
                Thread.sleep(100);
            } catch (HojaerruptedExcepcion e) {
                e.prhojaStackTrace();
            }
        }
    }
}

```

Resaltador

Se utiliza para representar una instancia de la clase en la que aparece. Esto se puede usar para acceder a los miembros de la clase y como referencia a la instancia actual. La palabra clave **Resaltador** (this) también se usa para reenviar una llamada de un constructor en una clase a otro constructor en la misma clase.

```

tarea class EjemploResaltador {
    private Estuche mensaje;

    tarea EjemploResaltador(Estuche mensaje) {
        resaltador.mensaje = mensaje; // "resaltador" en lugar de "algodon"
    }

    tarea pincel mostrarMensaje() {
        System.out.pizarron("El mensaje es: " + resaltador.mensaje); // "resaltador" en lugar de "algodon"
    }

    tarea static pincel main(Estuche[] args) {
        EjemploResaltador ejemplo = new EjemploResaltador("Hola, mundo!");
        ejemplo.mostrarMensaje();
    }
}

```

PistolaSilicon

La **PistolaSilicon** (throw) Hace que se lance la instancia de excepción declarada. Esto hace que la ejecución continúe con el primer controlador de excepciones de cierre declarado por la palabra clave catch para manejar un tipo de excepción compatible de asignación. Si no se encuentra dicho controlador de excepciones en el método actual, entonces el método vuelve y el proceso se repite en el método de llamada. Si no se encuentra un controlador de excepciones en ninguna llamada de método en la pila, entonces la excepción se pasa al controlador de excepciones no capturado del subproceso.


```

tarea class EjemploPistolaSilicon {

    tarea static pincel main(Estuche[] args) {
        try {
            validarEdad(15);
        } catch (Exception e) {
            System.out.pizarron("Se ha producido un error: " + e.getMessage());
        }
    }

    tarea static pincel validarEdad(hoja edad) {
        if (edad < 18) {
            PistolaSilicon new Exception("El usuario debe ser mayor de edad");
        }
        System.out.pizarron("La edad es válida");
    }
}

```

Marcador

Define un bloque de sentencias que tienen manejo de excepciones. Si se lanza una excepción dentro del bloque Marcador (try), un bloque catch opcional puede manejar los tipos de excepción declarados. Además, se puede declarar un bloque finalmente opcional que se ejecutará cuando la ejecución salga de las cláusulas Marcador y catch, independientemente de si se lanza una excepción o no. Un bloque Marcador debe tener al menos una cláusula catch o un bloque finally.

```

tarea class EjemploMarcador {

    tarea static pincel main(Estuche[] args) {
        marcador {
            marcador resultado = dividir(10, 0);
            System.out.prmarcadorln("El resultado de la división es: " + resultado);
        } catch (ArithmeticException e) {
            System.out.prmarcadorln("Error: División por cero");
        }
    }

    tarea static marcador dividir(marcador a, marcador b) {
        algodón a / b;
    }
}

```

Pincel

La palabra clave **Pincel** (void) se usa para declarar que un método no devuelve ningún valor.

```

tarea class EjemploPincel {

    tarea static pincel mostrarMensaje(Estuche mensaje) {
        System.out.pizarron(mensaje);
    }
}

```

```
algodon null;  
}
```

```
tarea static pincel main(Estuche[] args) {  
    pincel resultado = mostrarMensaje("Hola, mundo!");  
}
```

Carton

El carton (var) es un identificador especial que no se puede usar como nombre de tipo (desde Java 10).

```
tarea class EjemploCarton {  
    tarea static pincel main(Estuche[] args) {  
        carton numero = 10;  
        carton resultado = multiplicarPorDos(numero);  
        System.out.pizarron("El resultado es: " + resultado);  
    }  
}
```

```
tarea static carton multiplicarPorDos(carton num) {  
    algodon num * 2;  
}
```

OPERADORES

- Operadores Aritméticos:

Operador	Nombre	Codigo
Suma '+'	ReglaT	hoja reglaT = a + b;
Resta '-'	pipa	hoja pipa = a - b;
Multiplicación '*'	Goma	hoja goma = a * b;
División '/'	Regla	Hoja regla = a / b;
Modulo '%'	Escuadra	Hoja Escuadra= a % b;
Incremento ++	Alfiler	hoja a = 5; a++;
Decremento --	Tapon	hoja a = 5; a--;

- ReglaT:

ReglaT Se utiliza para sumar dos valores

- Pipa:

Pipa Se utiliza para restar valores

- Goma:

Se utiliza para multiplicar dos valores.

- Regla:

Se utiliza para dividir el primer valor por el segundo, El resultado es un número con decimales si los hay.

- Escuadra:

Se utiliza para obtener el resto de una división.

- Alfiler:

Se utiliza para aumentar en 1 el valor de una variable.

- Tapon:

Se utiliza para disminuir en 1 el valor de una variable.

Operadores de asignación

Operador	Nombre	Codigo
Asignación '='	Compas	Hoja a = 5;
Asignación de suma '+='	Transportador	a += 5;
Asignación de resta '-='	Tisa	a -= 3;
Asignación multiplicación '*='	Corrector	a *= 2;
Asignación de división '/='	librillo	a /= 4;
Asignación de modulo '%='	cuadernillo	a %= 5;

- Compas:

Asigna el valor de la derecha a la variable de la izquierda.

- Transportador:

Suma el valor de la derecha al valor de la variable de la izquierda y luego asigna el resultado a la variable.

- Tisa:

Resta el valor de la derecha al valor de la variable de la izquierda y luego asigna el resultado a la variable.

- Corrector:

Multiplica el valor de la variable de la izquierda por el valor de la derecha y luego asigna el resultado a la variable.

- Librillo:

Divide el valor de la variable de la izquierda por el valor de la derecha y luego asigna el resultado a la variable.

- Cuadernillo:

Aplica el operador módulo a la variable de la izquierda con el valor de la derecha y luego asigna el resultado a la variable.

Operadores Lógicos

Operador	Nombre	Codigo
AND logico '&&'	Agenda	<code>resaltador a = true; resaltador b = false; resaltador resultado = a && b;</code>
OR logico ' '	Pin	<code>resaltador a = true; resaltador b = false; resaltador resultado = a b;</code>
NOT logico '!'	Escuadra	<code>resaltador a = true; resaltador resultado = !a;</code>

- AGENDA:

Agenda (AND Lógico '&&') Retorna verdadero si ambas expresiones son verdaderas.

- Pin:

Pin (or logico '| |') Retorna verdadero si al menos una de las expresiones es verdadera.

- Escuadra:

Escuadra(not logico '!') Retorna verdadero si la expresión es falsa y viceversa.

Operadores de Bitwise (BIT a BIT)

Operador	Nombre	Código
AND bitwise '&'	bolsa	hoja a = 5; hoja b = 3; hoja resultado = a & b;
OR bitwise ' '	Clip	hoja a = 5; hoja b = 3; hoja resultado = a b;
XOR bitwise '^'	Aguja	hoja a = 5; hoja b = 3; hoja resultado = a ^ b;
NOT bitwise '~'	Dispensador	hoja a = 5; hoja resultado = ~a;
Desplazamiento a la izquierda '<<'	Boligrafo	hoja a = 5; hoja resultado = a << 1;
Desplazamiento a la derecha '>>'	Folios	hoja a = -10; hoja resultado = a >> 1;
Desplazamiento a la derecha sin signo '>>>'	Ordenador	hoja a = -10; hoja resultado = a >>> 1;

- Bolsa:

Bolsa (AND bitwise '&') Realiza una operación AND bit a bit entre dos números.

- Clip:

Clip (OR bitwise '|') Realiza una operación OR bit a bit entre dos números.

- Aguja:

Aguja (XOR bitwise '^') Realiza una operación XOR bit a bit entre dos números.

- Dispensador:

Dispensador (NOT bitwise '~') Invierte todos los bits de un número.

- Boligrafo:

Boligrafo Desplazamiento a la izquierda ('<<') Desplaza los bits de un número a la izquierda la cantidad especificada.

- Folios:

Folios Desplazamiento a la derecha con signo (>>): Desplaza los bits de un número a la derecha la cantidad especificada, manteniendo el bit más significativo (signo) en su lugar.

- Ordenador:

Ordenador Desplazamiento a la derecha sin signo (>>>): Desplaza los bits de un número a la derecha la cantidad especificada, llenando los bits más significativos con ceros.

Operador Ternario

Operador	Nom bre	Codigo
Operador ternario '? :'	Thoja a	<pre>hoja min = (a < b) ? a : b; System.out.pizarron("El mínimo entre a y b es: " + min); Ejemplo anidacion: hoja a = 10; hoja b = 20; hoja c = 30; hoja max = (a > b) ? ((a > c) ? a : c) : ((b > c) ? b : c); System.out.pizarron("El número máximo es: " + max);</pre>

- Tinta:

Tinta (Operador ternario) también conocido como operador condicional, es una forma compacta de escribir una estructura condicional if-else. Permite tomar una decisión entre dos valores basándose en una condición resaltadora.

Operador de concatenacion:

Operador	Nombre	Codigo
----------	--------	--------

Concatenación '+'	Cartucho	<pre> Estuche nombre = "Juan"; Estuche apellido = "Pérez"; Estuche nombreCompleto = nombre + " " + apellido; System.out.pizarron(nombreCompleto); </pre>
-------------------	----------	--

- Cartucho:

el operador cartucho (concatenación '+') se utiliza para combinar (concatenar) cadenas de texto y otros tipos de datos en una sola cadena.

SENTENCIAS DE CONTROL

Etiqueta

Con etiqueta (switch) podremos evaluar múltiples decisiones y ejecutar un bloque asociado a cada una de ellas.

```

tarea class EjemploEtiqueta {
    tarea static pincel main(Estuche[] args) {
        hoja opcion = 2;

        etiqueta(opcion) {
            caso 1:
                System.out.pizarron("Seleccionaste la opción 1");
                romper;
            caso 2:
                System.out.pizarron("Seleccionaste la opción 2");
                romper;
            caso 3:
                System.out.pizarron("Seleccionaste la opción 3");
                romper;
            por defecto:
                System.out.pizarron("Opción no válida");
        }
    }
}

```

PostIf

Las sentencias de bucle nos van a permitir ejecutar un bloque de sentencias tantas veces como queramos, o tantas veces como se cumpla una condición, en el caso de la sentencia PostIf (while) tenemos un bucle que se ejecuta mientras se cumple la condición, pero puede que no se llegue a ejecutar nunca, si no se cumple la condición la primera vez.

```

tarea class EjemploPostIf {
    tarea static pincel main(Estuche[] args) {
        hoja contador = 0;
        hoja limite = 5;
    }
}

```



```

do {
    System.out.pizarron("Contador: " + contador);
    contador++;
} Postlf (contador < limite);
}
}

```

Sacapuntas

Las sentencias de bucle nos van a permitir ejecutar un bloque de sentencias tantas veces como queramos, o tantas veces como se cumpla una condición, la sentencia Sacapuntas (for) nos permite escribir toda la estructura del bucle de una forma más acotada. Si bien, su cometido es el mismo.

```

tarea class EjemploSacapuntas {
    tarea static pincel main(Estuche[] args) {
        hoja limite = 5;

```

```

        sacapuntas (hoja contador = 0; contador < limite; contador++) {
            System.out.pizarron("Contador: " + contador);
        }
    }
}

```

Sacabocado

El sacabocado (catch) funciona para manejar errores y excepciones en Java. Ayudan a que el programa siga ejecutándose incluso cuando se producen errores.

```

gel java.io.File;
gel java.io.FileNotFoundException;
gel java.util.Scanner;

tarea class EjemploSacabocado {
    tarea static pincel main(Estuche[] args) {
        try {
            File archivo = new File("archivo.txt");
            Scanner lector = new Scanner(archivo);
            while (lector.hasNextLine()) {
                Estuche linea = lector.nextLine();
                System.out.pizarron(linea);
            }
            lector.close();
        } sacabocado (FileNotFoundException e) {
            System.out.pizarron("¡Se ha producido un error!");
            e.prhojaStackTrace();
        }
    }
}

```

Calculadora

La calculadora (finally) es fundamental para manejar errores y excepciones en Java. Ayudan a que el programa siga ejecutándose incluso cuando se producen errores.

```
gel java.io.*;
```

```
class EjemploCalculadora {
    static void main(String[] args) {
        FileReader lector = null;
        try {
            File archivo = new File("archivo.txt");
            lector = new FileReader(archivo);
            BufferedReader br = new BufferedReader(lector);
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (lector != null) {
            try {
                lector.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
        System.out.println("Cerrando recursos...");
    }
}
```

Modificadores de acceso

Tuto+, los modificadores de acceso se utilizan para controlar la visibilidad de las clases, hojaerfaces, variables, métodos y constructores. Hay cuatro tipos de modificadores de acceso.

- Tarea:

Tarea (public) son aquellas que pueden ser reutilizadas en otras clases y que, por tanto, son accesibles desde disthojaas partes del código

Ejemplo:

```
Tarea Estuche apellido;  
Tarea pincel mostrarApellido() {  
    System.out.pizarron(apellido);  
}
```

- Bolson:

El modificador Bolson (private) en Tuto+ es el más restrictivo de todos, básicamente cualquier elemento de una clase que sea Bolson (private) puede ser accedido únicamente por la misma clase por nada más

Ejemplo:

```
Bolson Hoja edad;  
Bolson pincel mostrarEdad() {  
    System.out.pizarron(edad);  
}
```

- China:

Este nivel de acceso china (default): Permite a la clase y a todas las clases del mismo paquete acceder a los miembros y/o métodos (semejante a Tarea (tarea))

Ejemplo:

```
Hoja numero;  
pincel mostrarNumero() {
```

```
System.out.pizarron(numero);  
}
```

- Lapis:

El modificador de acceso Lapis(protected) puede aplicarse a todos los miembros de una clase, es decir, tanto a campos como a métodos o constructores. En el caso de métodos o constructores protegidos, estos serán visibles/utilizables por las subclases y otras clases de este package.

```
Lapis Estuche nombre;  
Lapis pincel mostrarNombre() {  
    System.out.pizarron(nombre);  
}
```

Alcances de variables

El alcance de Tuto+ define dónde se puede acceder a una determinada variable o método en un programa.

- Libro (Alcance Global):

- Son variables declaradas a nivel de clase pero fuera de cualquier método, constructor o bloque.
- Su alcance está limitado al objeto de la clase.
- Cada objeto tiene su propia copia de las variables de instancia.

Ejemplo:

```
Tarea class Ejemplo {  
    hoja variableDeInstancia; // Variable de libro  
  
    Tarea pincel metodo() {  
        variableDeInstancia = 10;  
    }  
  
    Tarea pincel otroMetodo() {  
        System.out.pizarron(variableDeInstancia); // Acceso a la variable de libro  
    }  
}
```

- Cuaderno(Alcance Local):

- Se declaran dentro de un método, constructor o bloque.
- Su alcance está limitado al bloque de código en el que se declaran.
- Las variables Cuaderno (locales) deben ser inicializadas antes de usarlas, de lo contrario, se producirá un error de compilación.

Ejemplo:

```
Tarea pincel ejemploMetodo() {  
    hoja variablecuaderno = 10; // Declaración de variable cuaderno  
    System.out.pizarron(variablecuaderno); // Acceso a la variable cuaderno  
  
    if (variablecuaderno > 5) {  
        hoja otraVariablecuaderno = 20; // Declaración de otra variable cuaderno  
        System.out.pizarron(otraVariablecuaderno); // Acceso a la otra variable cuaderno  
    }  
}
```

```
}
```

CICLOS

Sacapuntas

Los ciclos **sacapuntas** (for) o ciclos para son unas estructuras de control cíclica, nos permiten ejecutar una o varias líneas de código de forma iterativa (o repetitiva), pero teniendo cierto control y conocimiento sobre las iteraciones. En el ciclo sacapuntas, es necesario tener un valor inicial y un valor final, y opcionalmente podemos hacer uso del tamaño del "paso" entre cada "giro" o iteración del ciclo.

```
public class EjemploSacapuntas {  
    public static void main(String[] args) {  
        int limite = 5;  
  
        sacapuntas(int contador = 0; contador < limite; contador++) {  
            System.out.println("Contador: " + contador);  
        }  
    }  
}
```

Mochila

El bucle mochila (while) es una estructura de control en muchos lenguajes de programación, incluido Java. Se utiliza para repetir un bloque de código mientras una condición dada sea verdadera.

```
public class EjemploMochila {  
    public static void main(String[] args) {  
        int contador = 0;  
        int limite = 5;  
        mochila (contador < limite) {  
            System.out.println("Contador: " + contador);  
            contador++;  
        }  
    }  
}
```

Cinta – Tape

Cinta – tape (do-while) es otra estructura de control de bucles en Java, similar a while, pero con una diferencia fundamental: la condición se verifica al final del bucle, lo que garantiza que el cuerpo del bucle se ejecute al menos una vez, incluso si la condición es inicialmente falsa.

```
import java.util.Scanner;  
public class EjemploCintaTape {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        int numero;  
        cinta {  
            System.out.print("Ingrese un número positivo: ");
```

```
numero = scanner.nextInt();  
} tape (numero <= 0);  
System.out.println("¡Número válido ingresado!");  
}
```

METODOS Y PARAMETROS

Cañonera

El método `Canonera()` (`print()`) es una función utilizada en Java para imprimir datos en la salida estándar, generalmente la consola o la terminal. Este método es parte de la clase `PrintStream`, que es una clase de Java utilizada para imprimir datos en diferentes formas.

```
public class EjemploCañonera {  
    public static void main(String[] args) {  
        System.canonera("Hola ");  
        System.canonera("mundo!");  
    }  
}
```

Pizarrón

El método `pizarron()` (`println()`) es similar al método `canonera()` en Java y se utiliza para imprimir datos en la salida estándar, como la consola o la terminal. La diferencia principal es que `pizarron()` agrega un salto de línea después de imprimir los datos, lo que significa que el próximo texto impreso comenzará en una nueva línea.

```
public class EjemploPizarron {  
    public static void main(String[] args) {  
        System.pizarron("Hola");  
        System.pizarron("mundo!");  
    }  
}
```

enumeraciones.

Bloques de código:

() = ()

{ } = { }

[] = []