

**Índice**

Introducción a Java ………………………………………………………. 4

Definiciones de programación ………………………………………….. 5

Interfaz de desarrollo ……………………………………………………. 6

Primeros pasos en Java ………………………………………………… 7

Tipos de datos …………………………………………………………… 9

Variables …………………………………………………………………. 10

Operadores ………………………………………………………………. 11

Estructuras de control …………………………………………………… 12

Funciones y procedimientos …………………………………………… 13

Control de errores (**try-catch**, **throw**) ………………………………….. 14

Bibliotecas (**import**) ……………………………………………………… 16

Objetos y Clases (**class**, **static**) ………………………………………… 17

String y Array …………………………………………………………… 20

Interfaz gráfica …………………………………………………………… 22

Imágenes ………………………………………………………….......... 29

Conexión entre interfaces …………………………………………….. 30

Dialogs y otras funciones útiles ………………………………………. 31

Programación orientada a objetos …………………………………….. 33

Encapsulamiento ……………………………………………………….. 34

Herencia …………………………………………………………………. 35

Polimorfismo ……………………………………………………………. 37

Palabra reservada **super** ……………………………………………… 39

Hilos ………………………………………………………………………. 40

Escuchadores (**listener**) ……………………………………………….. 44

Math ………………………………………………………………………. 45

Control de decimales ………………………………………………….. 47

Números aleatorios ……………………………………………………. 48

Vectores y Matrices …………………………………………………… 49

Estructuras de datos (**listas**) …………………………………………… 52

Recursividad ………………………………………………………,…….. 55

Archivos …………………………………………………………………… 56

Bases de datos …………………………………………………………… 58

Introducción a Java

Java es un lenguaje de programación orientado a objetos multiplataforma.

Es un lenguaje de tipo **compilado**, es decir, un lenguaje donde se escribe un código y, antes de poder ejecutarse, debe pasar por un compilador que traduce el código a otro lenguaje el cual, otro programa, lo convierte a lenguaje binario para que pueda ser comprendido por el ordenador.

Dicho compilador se trata de Java Development Kit (JDK), el cual debe ser descargado para comenzar a trabajar con Java.

Es necesario saber que el código de Java se lee de arriba hacia abajo y de derecha a izquierda.

Es importante recordar los principios básicos de la programación:

1. Analizar la situación
2. Resolver el problema
3. Programar

Definiciones de programación

**Algoritmo**

Serie ordenada, finita y precisa de acciones que resuelven un problema.

Siempre se debe obtener el mismo resultado si se tienen los mismos datos de entrada (como una receta de cocina).

**Programa**

Traducción de un algoritmo a un lenguaje de programación determinado capaz de ser ejecutado por una computadora.

Un programa es un algoritmo, pero no todo algoritmo es un programa.

* **Datos de entrada:** Toda información que llega al algoritmo.
* **Datos de salida:** Información que sale del algoritmo.

**Pre-condiciones**

Todas las condiciones que el algoritmo asume que cumplen sobre los datos de entrada.

**Pos-condiciones**

Todas las condiciones que va a cumplir el resultado del algoritmo y sólo se pueden cumplir si se cumplen las pre-condiciones.

**Caja negra**

Se observan sólo los datos de entrada y los datos de salida para comprobar el funcionamiento del algoritmo.

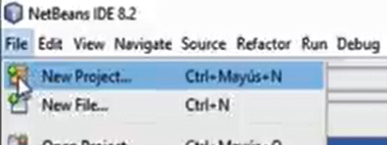
Se centra en qué hace el algoritmo y se preocupa en cumplir las pre-condiciones.

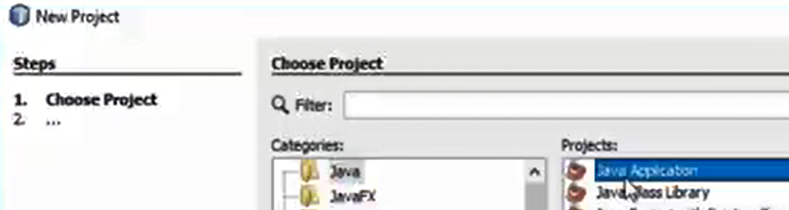
**Caja blanca**

Se observa el algoritmo desde dentro para ver todo su funcionamiento.

Se centra en cómo se hace el algoritmo y se preocupa por cumplir las pos-condiciones.

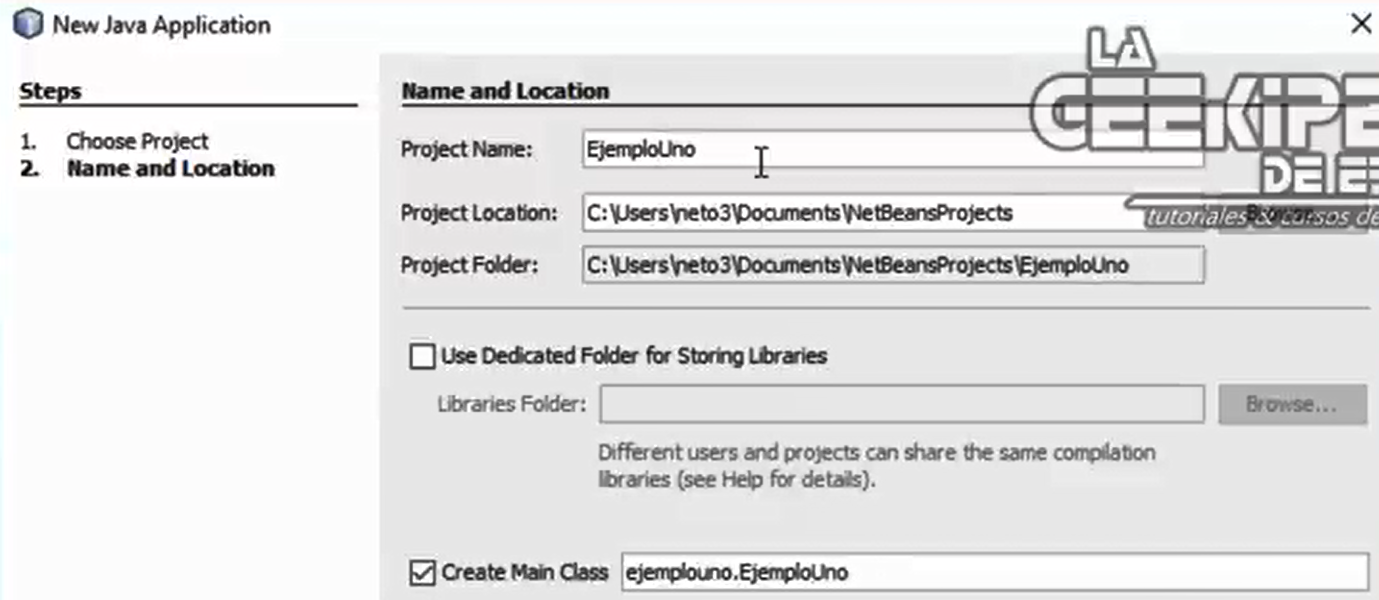
Interfaz de Desarrollo

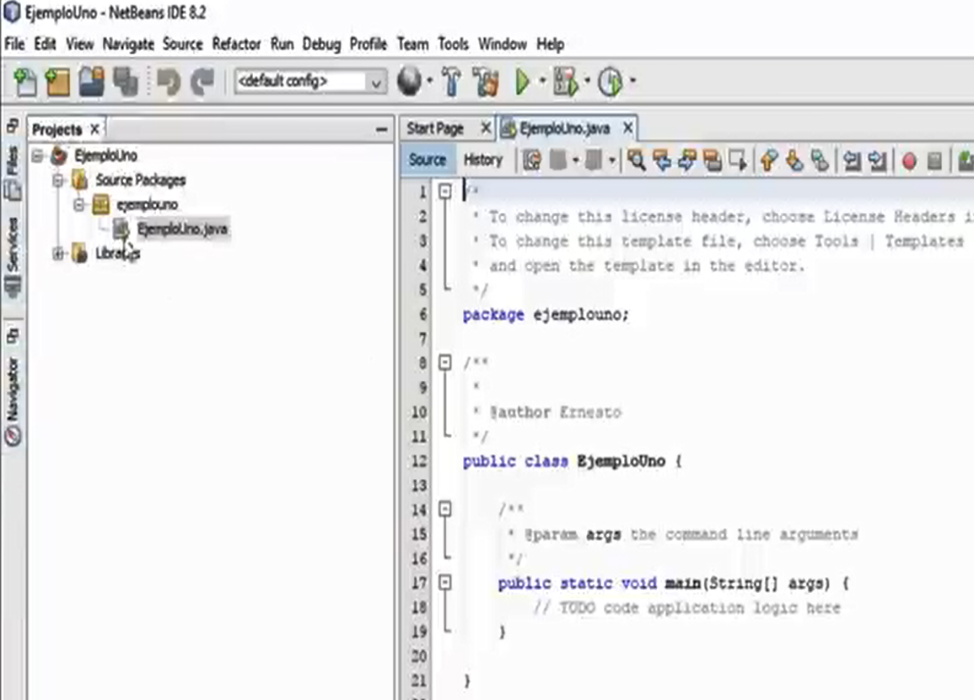
Uno de los Entornos de Desarrollo Integrados (IDE son sus siglas en inglés), es **NETBEANS**.

Para empezar a utilizar Netbeans, se debe empezar creando un nuevo proyecto.

Se debe elegir la opción “Java Aplication”.

Luego se procede a escribir el nombre del proyecto, el cual debe estar escrito sin espacios.



Posteriormente, se creará la primera class del proyecto donde podremos empezar a trabajar.

JavaDocs

Mediante el uso de etiquetas específicas, Netbeans permite documentar el trabajo de forma “automática”.

**@author** establece el nombre del autor del proyecto.

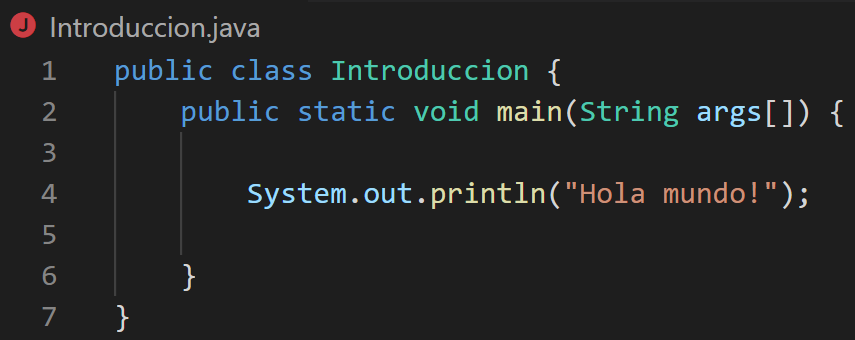
**@param** indica qué datos de entrada fueron enviado por medio de argumentos a un método o función.

Primeros pasos en Java

Para empezar a programar en Java, se debe crear un archivo con extensión **.java**

La primera plantilla que se muestra (nuestra primer class) debe iniciar con el nombre de la class con la primer letra en mayúscula, el cual debe ser el mismo que el del archivo. Si hay más de una palabra, se escribe sin espacio y con mayúscula en cada letra inicial de cada palabra.

A continuación, y dentro de la class, se debe escribir el **método main**, que es donde iniciará la ejecución del código.



Para compilar en Java, se debe abrir el **Simbolo del sistema** y dirigirnos hasta el directorio donde se encuentra el archivo java que creamos.

Mediante el comando **dir** se pueden “ver” los directorios de la carpeta actual.

Con el comando **cd** seguido del nombre de un directorio, se puede acceder a él.

Una vez encontrado nuestro archivo java en el Símbolo del sistema, se debe utilizar el comando **javac** (brindado por JDK) seguido del nombre de nuestro archivo java junto con su extensión.



De esta manera, se creará un archivo **.class**, el cual puede ser ejecutado con el Símbolo del sistema mediante el comando **java** seguido del nombre de nuestro archivo class (sin su extensión).



Empaquetado de un proyecto

Para que un programa creado con Java pueda ser ejecutado sin necesidad de utilizar el símbolo del sistema, es necesario empaquetarlo.

Para ello, se deben colocar todos los archivos .java y .class (junto con la carpeta de imágenes, en caso de que corresponda) en una misma carpeta.

En dicha carpeta, se debe crear un archivo de texto llamado **MANIFEST.MF**

El contenido del archivo manifest debe contener:

|  |
| --- |
| Manifest-Version: 1.0  Created-By: Mi nombre  Main-Class: Nombre\_de\_la\_class\_donde\_está\_el\_main  X-COMMENT: un comentario |

**es necesario dejar esta línea en blanco**

Luego, en el símbolo del sistema, se debe ejecutar la siguiente línea de código (estando ubicados en el directorio con todos los archivos):



Nombre del proyecto Todos los class del proyecto Todas las imágenes

Finalmente, se ejecuta la siguiente línea de código en el símbolo del sistema para crear el archivo .jar ejecutable:



Tipos de datos

Primitivos

|  |  |  |
| --- | --- | --- |
|  | ***nombre*** | ***rango*** |
| **Enteros** | byte | ± 28 / 2 |
| short | ± 216 / 2 |
| int | ± 232 / 2 |
| long | ± 264 / 2 |
| **Decimales** | float | 8 decimales |
| double | 15 decimales |
| **Caracteres** | char | un solo caracter |
| **Lógicos** | bool | **true** *o* **false** |

Objetos

Se caracterizan por ser necesaria su invocación debido a que son clases y, como tales, su inicial es en mayúscula.

* Cadenas de texto **String**
* Arreglos **Array** *vectores y matrices*

Variables

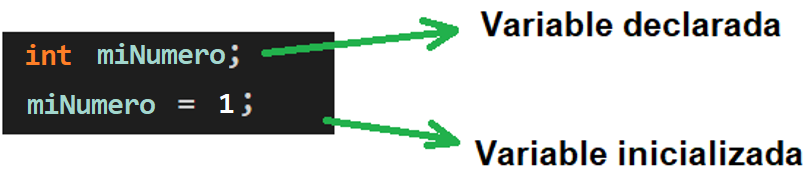
Una variable es un espacio en memoria donde se puede alojar información o datos.

Está conformado por 2 elementos: el tipo de dato y un nombre.

El nombre de una variable tiene ciertas reglas.

1. La letra inicial no puede ser un número ni un símbolo (excepto guion bajo o $).
2. Las constantes deben ser escritas sólo con letras mayúsculas.
3. Deben describir brevemente lo que la variable representa.
4. Si el nombre de la variable tiene más de una palabra, la inicial de las siguientes palabras se escribe en mayúscula (**camel case**).

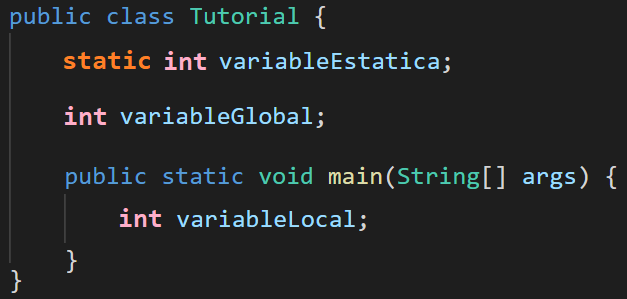
Inicializar variables

Es el proceso de asignar por primera vez un valor a una variable.

Se puede resumir de la siguiente manera:

Scope de una variable

El ámbito de una variable define su alcance de uso. Indica en qué secciones de código una variable estará disponible.

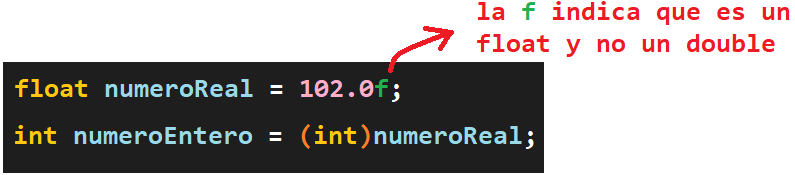
* **Local / Bloque** Sólo puede utilizarse dentro del método o bloque de código en que se encuentra.
* **Global / de instancia** Puede utilizarse en cualquier parte de la class. Incluso puede utilizarse desde fuera de esta class según su modificador de acceso. No debe inicializarse.
* **Estático / de clase** Pertenecen a la propia clase y para acceder a ellas desde otra class no es necesario crear una instancia de clases. Se denominan atributos de clase.

Operadores

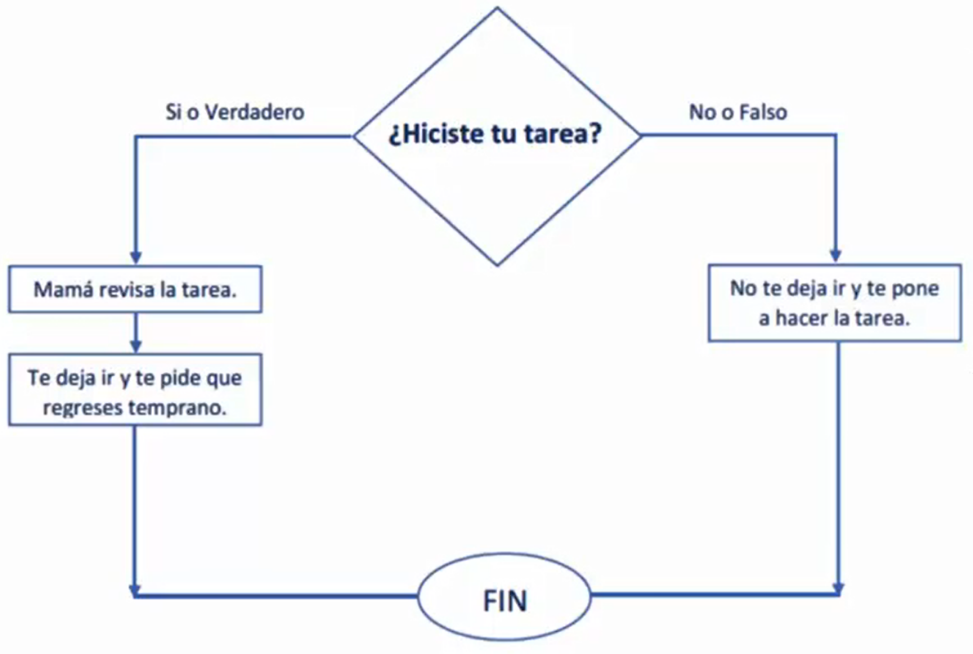
Símbolo que indica que debe ser llevada a cabo una operación especificada sobre cierto número de operador (tipo de dato).

* **Asignación** **=** Asigna un valor a un operando.
* **Aritméticos**
  + **Suma +** Suma dos números o concatena textos.
  + **Sustracción -** Resta dos números.
  + **Multiplicación \*** Multiplica números.
  + **División /** Divide números.
  + **Resto %** Divide números y se obtiene el resto de dicha división.
  + **Exponenciación ^** Aplica potencias.
* **Relacionales** Compara dos valores del mismo tipo y devuelve un valor lógico.
  + **Mayor estricto > true** *o* **false**
  + **Menor estricto <**
  + **Mayor o igual >=**
  + **Menor o igual <=**
  + **Desigualdad !=**
  + **Igualdad ==**
    - Si se desea comprar dos cadenas de texto (string), se debe usar el método **.ecuals**
* **Lógicos**
  + **And &&** Si todo es verdadero, retorna verdadero.
  + **Or ||** Si al menos uno es verdadero, retorna verdadero.
  + **Not !** Convierte lo verdadero en falso y viceversa.

Casting

Conversión de un tipo de dato a otro mediante el uso del tipo de dato entre paréntesis antes de escribir la variable con un tipo de dato diferente.

Estructuras de control



Estructura condicional

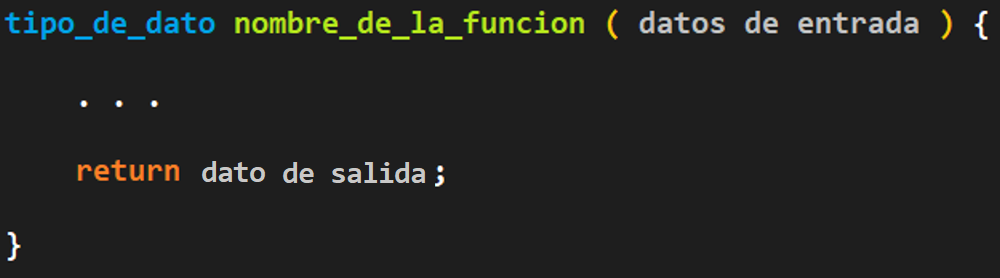
Ejecutan un bloque de código si se cumple una condición (**if**, **if-else**, **switch**).

Iteradores

Ejecuta un bloque de código varias veces (**while**, **do-while**, **for**).

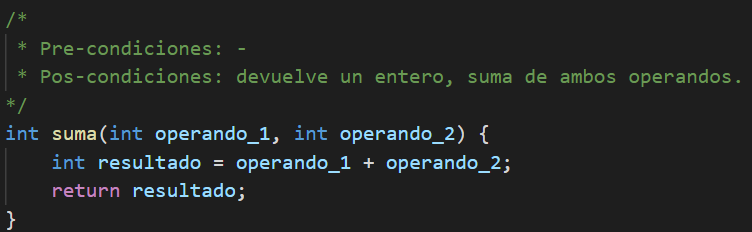
Funciones y procedimientos

Bloque de código el cual se puede utilizar declarándolo fuera del main (modularizar).



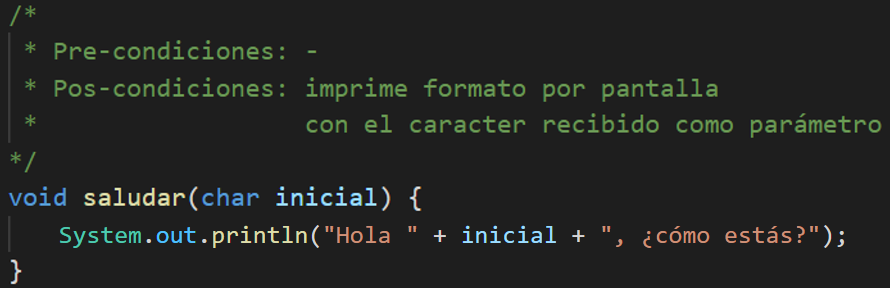
* **Funciones**

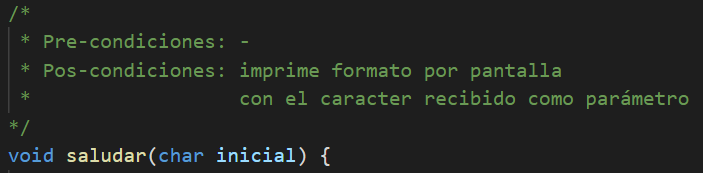
Se utilizan para realizar una acción específica.



* **Procedimientos (void)**

Realizan acciones sin devolver ningún resultado.

Se nombran con verbos.

Ninguna función o procedimiento debe recibir más de 5 o 7 parámetros.

Control de errores

Gestión de la excepción

Las excepciones son el medio para tratar situaciones anómalas que pueden suceder, como por ejemplo:

- Invocar a un método sobre un objeto "null".

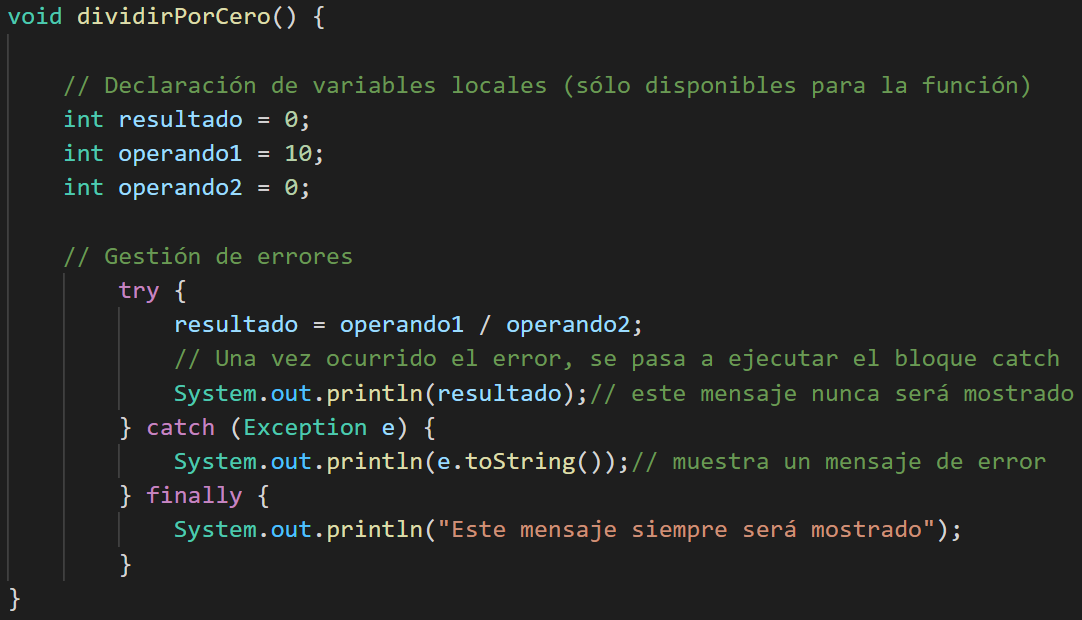
- Intentar abrir un fichero que no existe para leerlo.

- Intentar dividir un numero sobre cero.

Si un código no se ejecuta de la forma prevista, se establece cómo debe responder el programa.

Para ello, se utilizan las palabras reservadas:

* **try** función que se intentará realizar.
* **catch** lo que sucederá si surge un error.
* **finally** lo que sucederá independientemente de si hubo un error o no.

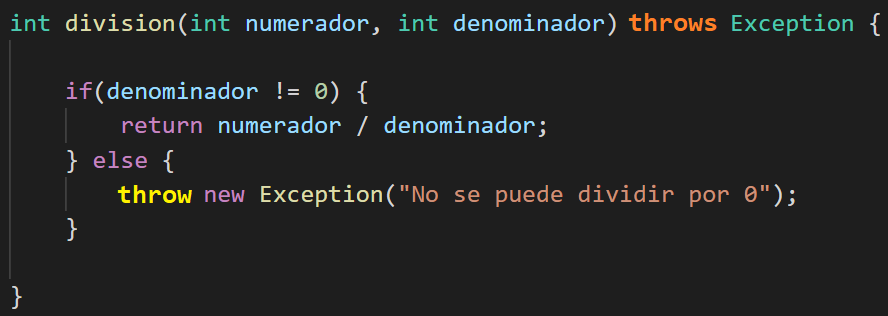


Definición de un error

Se puede definir un tipo de error mediante la palabra reservada **throws** / **throw**.

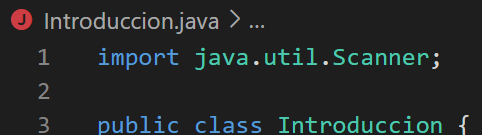
Al crearse una función con este tipo de declaración, forzosamente se deberá utilizar mediante una gestión de errores, es decir, deberá usarse con **try – catch**.

En código, se escribe de la siguiente manera:



Bibliotecas

Una biblioteca es un conjunto de clases que contienen atributos y métodos, es decir, un conjunto de códigos escritos por alguien para facilitar ciertas tareas.

Existen diversas bibliotecas nativas de Java, y para hacer uso de ellas, se necesita utilizar la palabra clave **import** seguida de la ruta donde se encuentra la biblioteca.

Scanner

Es una biblioteca que permite, entre otras cosas, obtener datos ingresados por el usuario en pantalla.

import java.util.Scanner;

public class ObtenerNombre {

    public static void main(String args[]) {

        Scanner input = new Scanner(System.in);

        String nombre = "";

        int edad = 0;

        System.out.println("¿Cuál es tu nombre?");

        nombre = input.nextLine();

        System.out.println("¿Cuál es tu edad?");

        edad = input.nextInt();

        System.out.println("Tu nombre es " + nombre);

        System.out.println("Y tu edad es " + edad);

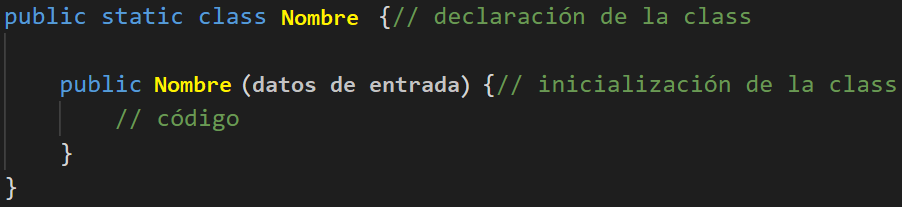
    }

}

Objetos y Clases

Una **class** es un objeto, se escriben con la primera inicial en mayúsculas y su función es, básicamente, la de un algoritmo (resolver un problema).

Para crearse, se deben declarar e inicializar de la siguiente forma:



Para que esta class pueda ser utilizada como una biblioteca, se debe crear el código completo de la class con su respectiva funcionalidad.

Luego, desde la class en la cual quiere ser utilizado como biblioteca, se la debe importar (**import**) para utilizarla y luego realizar una **instancia de clases**.

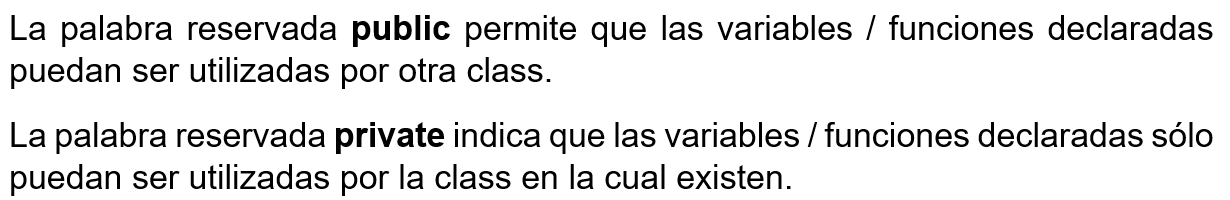


Cuando se empieza a trabajar con más de una class, es necesario conocer la palabra reservada **this**, la cual es una forma de indicar que se está trabajando con la class en la cual está metida dicho **this**.

Atributos de clases

Recordando que las class pueden ser trabajadas como objetos, se le pueden asignar atributos mediante la palabra reservada **static** (variables estáticas, específicas de una class).

Métodos de clases

Son las funciones o procedimientos pertenecientes a una class.

Ejemplo de una class

public class Culculadora {

    // Declaración de variables que no serán accesibles (privadas)

    private int operando1;

    private int operando2;

    // Constructor de la class (donde se inicializa)

    public Culculadora(int operando1, int operando2) {

        this.operando1 = operando1;

        this.operando2 = operando2;

    }

    /\* Pre-condiciones: -

     \* Pos-condiciones: devuelve la sumatoria entre los operandos ingresados

    \*/

    public int suma() {

        return operando1 + operando2;

    }

    /\* Pre-condiciones: -

     \* Pos-condiciones: devuelve la sustracción entre los operandos ingresados

    \*/

    public int resta() {

        return operando1 - operando2;

    }

    /\* Pre-condiciones: -

     \* Pos-condiciones: devuelve el producto de los operandos ingresados

    \*/

    public int multiplicacion() {

        return operando1 \* operando2;

    }

    /\* Pre-condiciones: denominador debe ser distinto de cero

     \* Pos-condiciones: devuelve la división de los números ingresados

    \*/

    public int division() throws Exception {

        if(operando2 != 0) {

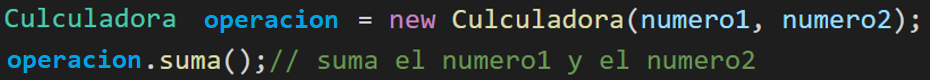
            return operando1 / operando2;

        } else

            throw new Exception("No se puede dividir por 0");

    }

}

Para poder utilizarlo desde otra class, se debe realizar una **instancia de clases**:

Otra manera de utilizar una class como biblioteca sin necesidad de instanciarla, es haciendo uso de la palabra reservada **static**. La cual, hace referencia a los atributos de una class como si fueran las características un objeto.

public class Culculadora {

    // No hay necesidad de declarar variables privadas

    // ni de inicializar la class

    /\*

     \* Pre-condiciones: -

     \* Pos-condiciones: devuelve la sumatoria entre los operandos ingresados

    \*/

    public static int suma(int operando1, int operando2) {

        return operando1 + operando2;

    }

    /\*

     \* Pre-condiciones: -

     \* Pos-condiciones: devuelve la sustracción entre los operandos ingresados

    \*/

    public static int resta(int operando1, int operando2) {

        return operando1 - operando2;

    }

    /\*

     \* Pre-condiciones: -

     \* Pos-condiciones: devuelve el producto de los operandos ingresados

    \*/

    public static int multiplicacion(int operando1, int operando2) {

        return operando1 \* operando2;

    }

    /\*

     \* Pre-condiciones: denominador debe ser distinto de cero

     \* Pos-condiciones: devuelve la división de los números ingresados

    \*/

    public static int division(int operando1, int operando2) throws Exception {

        if(operando2 != 0) {

            return operando1 / operando2;

        } else

            throw new Exception("No se puede dividir por 0");

    }

}

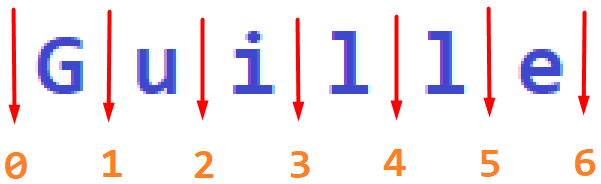
Para hacer uso de las funciones de esta class, se procede sin necesidad de instanciarla:

Existen muchas clases nativas de Java, las cuales incluyen los Strings y muchos objetos más.

Cadenas de texto (Strings)

Las cadenas de texto tienen métodos exclusivos que permiten manipularlos.

* **.substring(desde, hasta)** Permite obtener una parte en específico de un string.

 Para hacer buen uso del método, es necesario saber que una cadena de texto se divide en posiciones, las cuales comienzan en 0 y finalizan luego del último carácter.

Por lo tanto, si se utiliza la siguiente instrucción:

****



* **.lenght** Permite obtener el largo de una cadena de texto.
* **.equals(otroTexto)** Permite comparar dos cadenas de texto.



Arreglos

Estructura de datos que almacena una colección de datos bajo un mismo nombre. Todos deben pertenecer a un mismo tipo de dato.

* Unidimensional (Vectores)

Los elementos se almacenan en posiciones contiguas de memoria, identificados por un índice (índex).

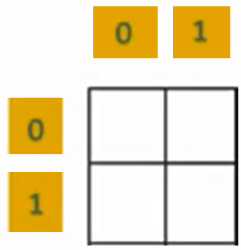


Su representación gráfica es la siguiente:

Para guardar un dato en una de las posiciones, se debe especificar dicha posición entre corchetes.

* Bidimensionales (Matrices)

Los elementos se almacenan en posiciones contiguas de memoria, identificados por dos índices (índex).



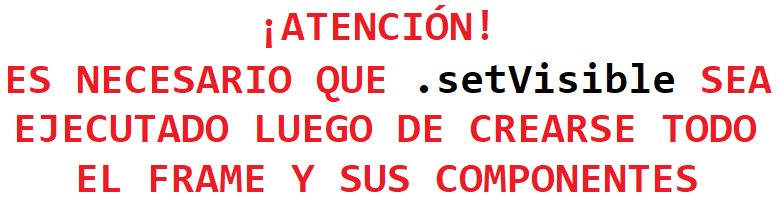
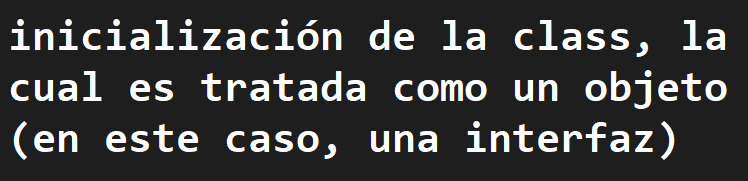
Su representación gráfica como matriz es la siguiente:

Para guardar un dato en una de las posiciones, se debe especificar dicha posición entre corchetes.

Interfaz gráfica

Programa informático que facilita el entorno al usuario al momento de manipular el programa.

Se debe pensar a la clase (class) como un objeto, el cual es, en este caso, una interfaz gráfica con sus propiedades y componentes.

****Para usar la class como interfaz gráfica, se debe hacer uso de la **herencia** de JFrame. Para ello, se debe importar la biblioteca **javax.swing.\***

import javax.swing.\*;

public class MiPrimerFrame extends JFrame {

    // Método main: donde inicia el programa

    public static void main(String[] args) {

        MiPrimerFrame interfaz = new MiPrimerFrame();

        interfaz.initComponents();// Componentes de interfaz

        interfaz.setVisible(true);// Visibiliza interfaz

    }

    // Constructor de igual nombre que la class

    public MiPrimerFrame() {

        setTitle("Tutorial");// Establece el título de la interfaz

        setLayout(null);// permite colocar las coordenadas de la interfaz manualmente

        setBounds(0,0,500,500);// posición X, posición Y, Ancho, Alto

        setLocationRelativeTo(null);// Interfaz en el centro de la pantalla

        setResizable(false);// Establece inmodificable el tamaño de la interfaz de forma manual

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);// Termina el programa al cerrar la interfaz

        //setSize(640,480);// (opcional) cambia el tamaño de la interfaz

    }

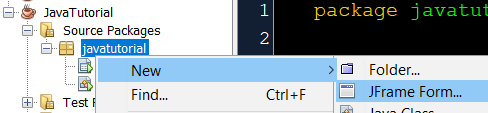
    // Crear componentes de la interfaz

    public void initComponents() {

. . .

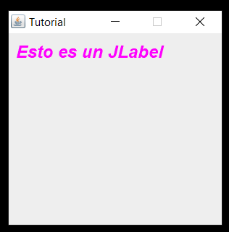
    }

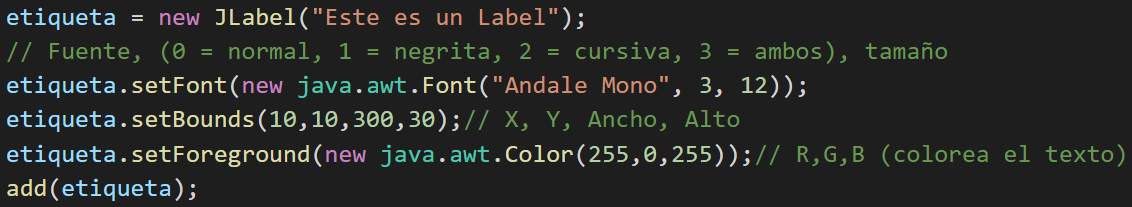
}

Una forma de crear interfaces utilizando el IDE Netbeans es mediante la creación de un JFrame Form.

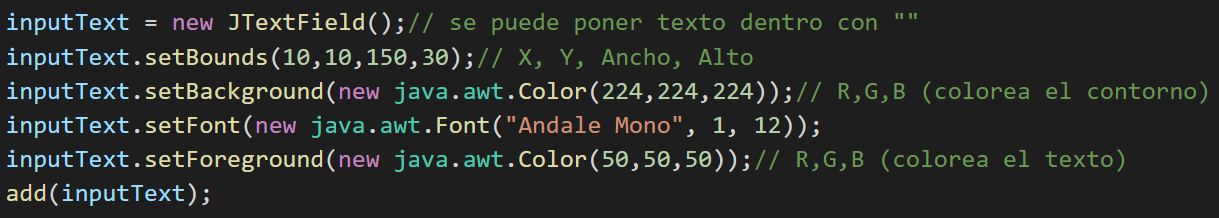
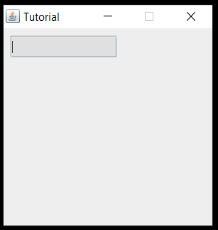
Componentes de la interfaz

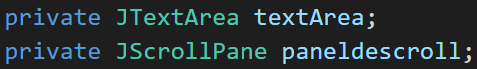
Todos los objetos de la interfaz deben ser declarados justo después del **public class**

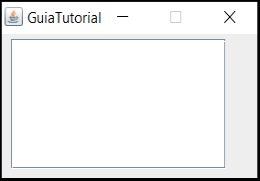
* **JLabel** (TextView): etiqueta



* **JTextField** (EditText): caja de texto *o* input-text



* **JTextArea**



Se puede obtener el texto introducido en una caja de texto mediante el siguiente método:

Y se puede pegar un texto dentro de una caja de texto mediante **.setText** o agregar un texto mediante **.append**

Si se desea quitar los espacios en blanco de un texto, se puede usar **.trim()**

Se puede obtener un valor numérico de un texto mediante el uso del siguiente método:

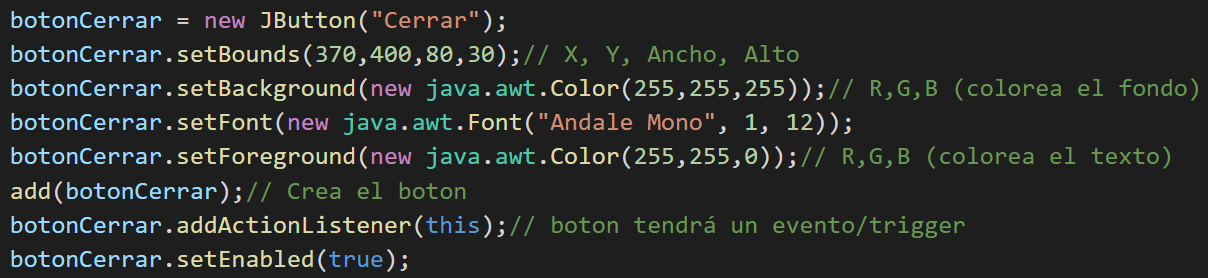
Puede ser necesario usar **try – catch** debido a que, si no hay números, se producirá un error.

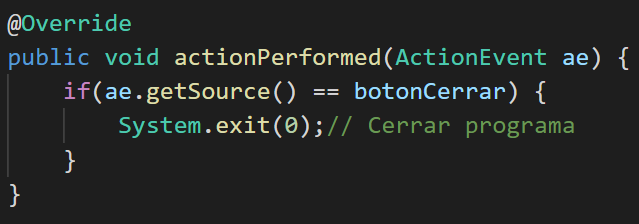


* **JButton**: permite crear botones con eventos.



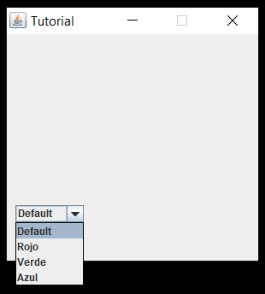


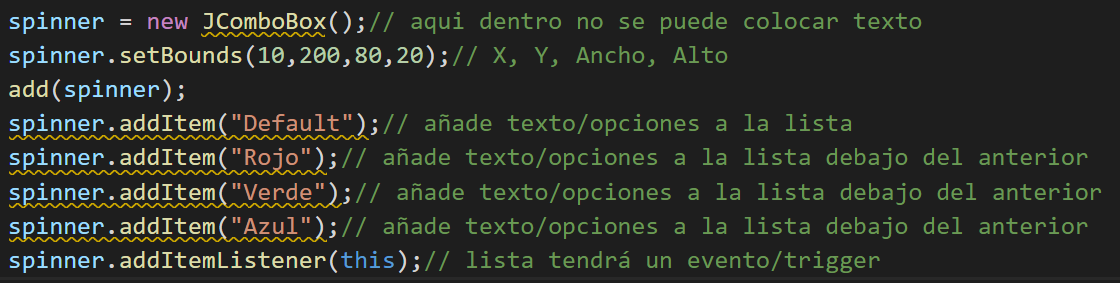


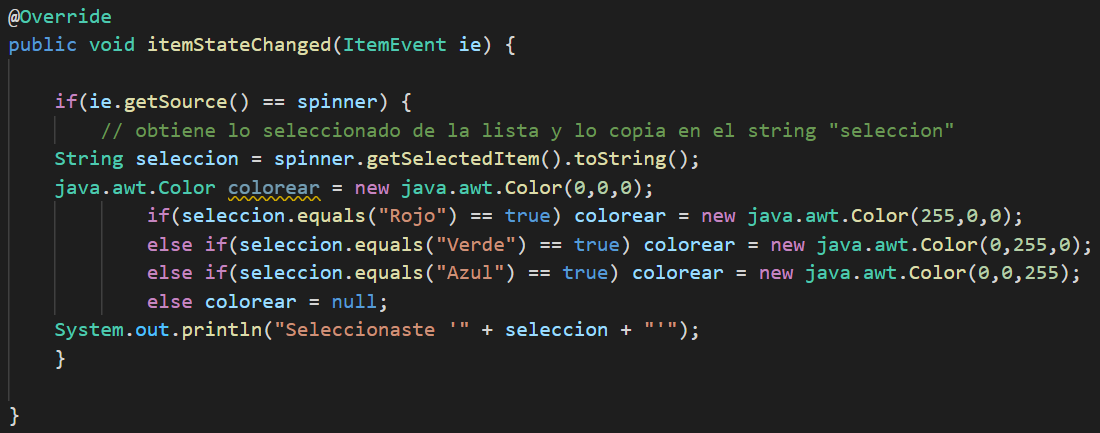


* **JComboBox** (Spinner): lista desplegable







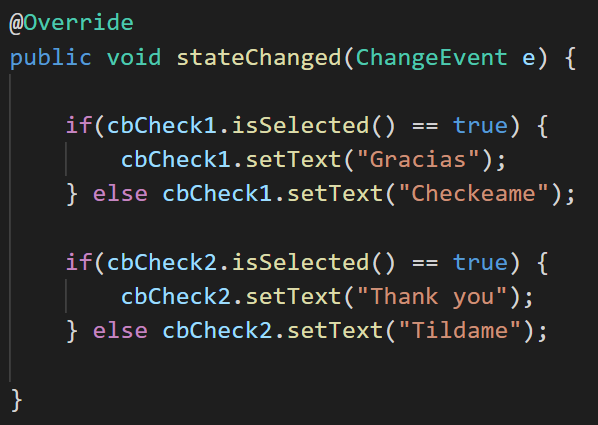
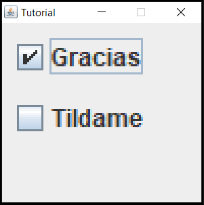
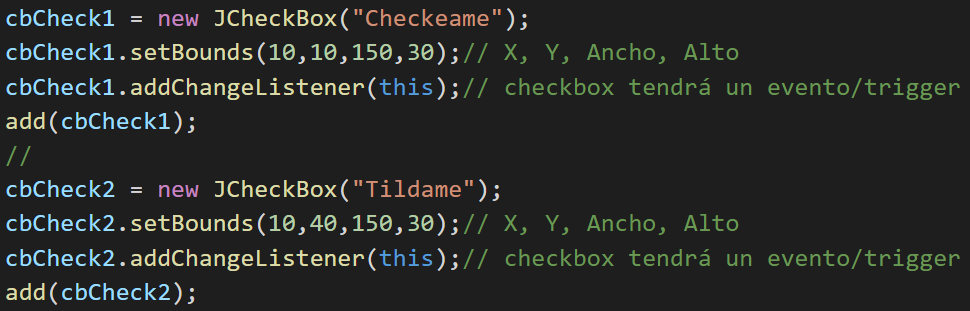


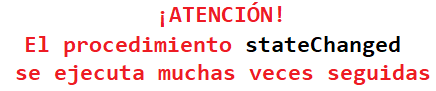


* **JCheckBox**

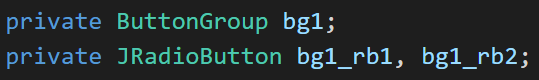




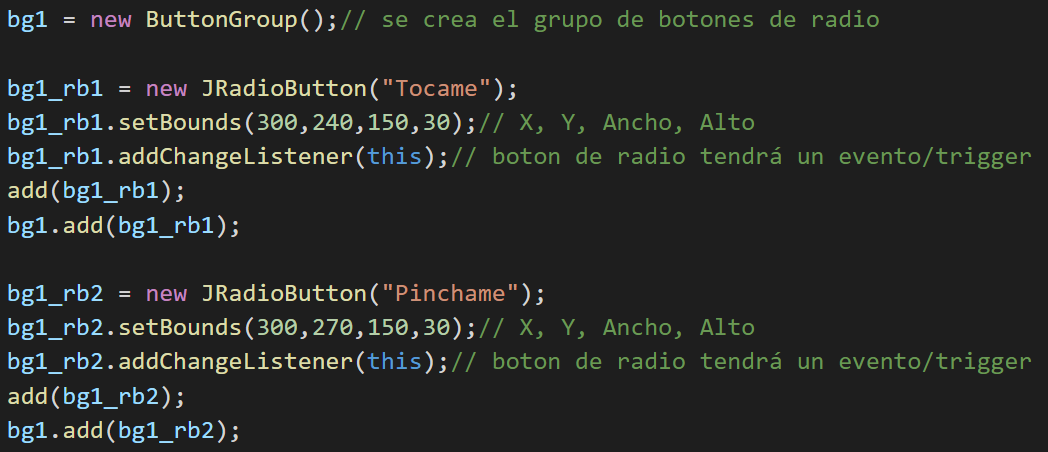
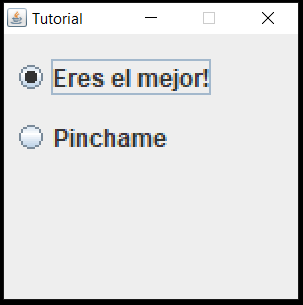
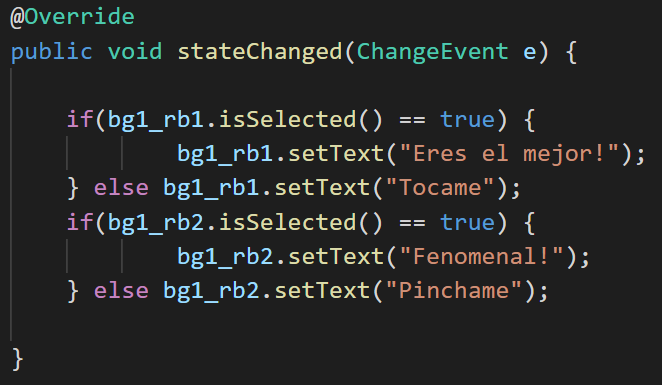
****

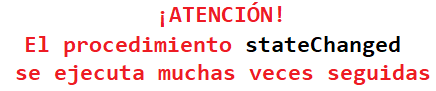


* **JRadioButton**: sólo un radio button puede estar activo a la vez.



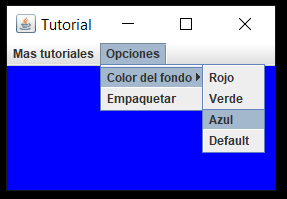






* **JMenu**

import javax.swing.\*;



import java.awt.Color;

import java.awt.Container;

import java.awt.event.\*;// Permite añadir eventos al menú

public class MiPrimerFrame extends JFrame implements ActionListener {

    private JMenuBar barra\_menus;

    // Menú 1:

    private JMenu menu1, menu2;

    private JMenu menu1\_item1;// un submenu

    private JMenuItem

            menu1\_item1\_subitem1,

            menu1\_item1\_subitem2,

            menu1\_item1\_subitem3,

            menu1\_item1\_subitem4;

    private JMenuItem menu1\_item2;

    // Menu 2:

    private JMenu menu2\_item1;// submenu

    private JMenuItem

            menu2\_item1\_subitem1,

            menu2\_item1\_subitem2,

            menu2\_item1\_subitem3,

            menu2\_item1\_subitem4;

    private JMenuItem menu2\_item2,menu2\_item3,menu2\_item4,menu2\_item6;

    private JMenu menu2\_item5;// submenu

    private JMenuItem menu2\_item5\_subitem1;

    // Método main: donde inicia el programa

    public static void main(String[] args) {

        MiPrimerFrame interfaz = new MiPrimerFrame();// Crea la interfaz

        interfaz.initComponents();// Crea los componentes de la interfaz

        interfaz.setVisible(true);// Visibiliza interfaz

    }

    // Constructor que debe tener el mismo nombre de la class

    public MiPrimerFrame() {

        setLayout(null);// permite colocar las coordenadas de la interfaz manualmente

        setBounds(0,0,500,500);// posición X, posición Y, Ancho, Alto

        setLocationRelativeTo(null);// Interfaz en el centro de la pantalla

        setResizable(false);// Establece que no se pueda modificar el tamaño de la interfaz de forma manual

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);// Termina el programa al cerrar la interfaz

        //setSize(640,480);// Cambia el tamaño de la interfaz

    }

    // Crear componentes de la interfaz

    public void initComponents() {

    barra\_menus = new JMenuBar();

    setJMenuBar(barra\_menus);// se aplica automaticamente en la parte superior de la interfaz

    //barra\_menus.setBackground(new java.awt.Color(255,0,0));// R,G,B (colorea la barra de menus)

    //

        menu2 = new JMenu("Mas tutoriales");

    barra\_menus.add(menu2);// añade el menu2 a la barra de menus

        //

            menu2\_item1 = new JMenu("Programacion Orientada a Objetos");

            menu2.add(menu2\_item1);

            //

                menu2\_item1\_subitem1 = new JMenuItem("Main");

        menu2\_item1.add(menu2\_item1\_subitem1);

        menu2\_item1\_subitem1.addActionListener(this);

                menu2\_item1\_subitem2 = new JMenuItem("Herencia (extends)");

        menu2\_item1.add(menu2\_item1\_subitem2);

        menu2\_item1\_subitem2.addActionListener(this);

                menu2\_item1\_subitem3 = new JMenuItem("Polimorfismo (abstract)");

        menu2\_item1.add(menu2\_item1\_subitem3);

        menu2\_item1\_subitem3.addActionListener(this);

                menu2\_item1\_subitem4 = new JMenuItem("Palabra reservada 'super'");

        menu2\_item1.add(menu2\_item1\_subitem4);

        menu2\_item1\_subitem4.addActionListener(this);

            //

            menu2\_item2 = new JMenuItem("File y PrintWriter");

            menu2.add(menu2\_item2);

            menu2\_item2.addActionListener(this);

            menu2\_item3 = new JMenuItem("Ambitos de variables");

            menu2.add(menu2\_item3);

            menu2\_item3.addActionListener(this);

            menu2\_item4 = new JMenuItem("Excepciones (try-catch-finally)");

            menu2.add(menu2\_item4);

            menu2\_item4.addActionListener(this);

            menu2\_item5 = new JMenu("Threads / Hilos");

            menu2.add(menu2\_item5);

            //

                menu2\_item5\_subitem1 = new JMenuItem("Main");

        menu2\_item5.add(menu2\_item5\_subitem1);

        menu2\_item5\_subitem1.addActionListener(this);

            //

            menu2\_item6 = new JMenuItem("Excepciones (try-catch-finally)");

            menu2.add(menu2\_item6);

            menu2\_item6.addActionListener(this);

    //

    menu1 = new JMenu("Opciones");

    barra\_menus.add(menu1);// añade el menu1 a la barra de menus

    //

            menu1\_item1 = new JMenu("Color del fondo");

            menu1.add(menu1\_item1);

            //

                menu1\_item1\_subitem1 = new JMenuItem("Rojo");

        menu1\_item1.add(menu1\_item1\_subitem1);

        menu1\_item1\_subitem1.addActionListener(this);

                menu1\_item1\_subitem2 = new JMenuItem("Verde");

        menu1\_item1.add(menu1\_item1\_subitem2);

        menu1\_item1\_subitem2.addActionListener(this);

        menu1\_item1\_subitem3 = new JMenuItem("Azul");

        menu1\_item1.add(menu1\_item1\_subitem3);

        menu1\_item1\_subitem3.addActionListener(this);

        menu1\_item1\_subitem4 = new JMenuItem("Default");

        menu1\_item1.add(menu1\_item1\_subitem4);

        menu1\_item1\_subitem4.addActionListener(this);

            //

            menu1\_item2 = new JMenuItem("Empaquetar");

            menu1.add(menu1\_item2);

            menu1\_item2.addActionListener(this);

    //

    }

    // Eventos del menú

@Override

    public void actionPerformed(ActionEvent ae) {

        if(ae.getSource() == menu1\_item1\_subitem1) {

            Container fondo = this.getContentPane();// obtiene el fondo de la interfaz, incluso si se modifica el tamaño

        fondo.setBackground(new Color(255,0,0));

        System.out.println("Pintaste el fondo de rojo.");

        }

        if(ae.getSource() == menu1\_item1\_subitem2) {

            Container fondo = this.getContentPane();// obtiene el fondo de la interfaz, incluso si se modifica el tamaño

        fondo.setBackground(new Color(0,255,0));

        System.out.println("Pintaste el fondo de verde.");

        }

        if(ae.getSource() == menu1\_item1\_subitem3) {

            Container fondo = this.getContentPane();// obtiene el fondo de la interfaz, incluso si se modifica el tamaño

        fondo.setBackground(new Color(0,0,255));

        System.out.println("Pintaste el fondo de azul.");

        }

        if(ae.getSource() == menu1\_item1\_subitem4) {

            Container fondo = this.getContentPane();// obtiene el fondo de la interfaz, incluso si se modifica el tamaño

        fondo.setBackground(null);

        System.out.println("Pintaste el fondo de default.");

        }

        if(ae.getSource() == menu1\_item2) {

            Container fondo = this.getContentPane();// obtiene el fondo de la interfaz, incluso si se modifica el tamaño

        fondo.setBackground(new Color(244,244,244));

        System.out.println("Ahora sabes empaquetar");

        }

        if(ae.getSource() == menu2\_item1\_subitem1) {// Programacion Orientada a Objetos (base)

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item1\_subitem2) {// Herencia (extends)

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item1\_subitem3) {// Polimorfismo (abstract)

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item1\_subitem4) {// palabra reservada super

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item2) {// Files y PrintWriter

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item3) {// Ámbitos de variables

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item4) {// Excepciones (try-catch-finally)

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item5\_subitem1) {// Threads / Hilos

            System.out.println("¿Ya viste este tutorial?");

        }

        if(ae.getSource() == menu2\_item6) {// Randomize

            System.out.println("¿Ya viste este tutorial?");

        }

    }

}

Fondo de la interfaz

Se puede modificar el fondo de la interfaz gráfica mediante las siguientes líneas

Container fondo = this.getContentPane();// obtiene el fondo de la interfaz

fondo.setBackground(new Color(244,244,244));

Ícono de la interfaz

Para agregar un ícono en nuestra interfaz gráfica, se procede de la siguiente manera:

1. Establecer una carpeta donde se ubicará el proyecto.

Esta carpeta es donde se ubicarán los archivos .java y los archivos .class

1. Dentro de la carpeta del proyecto, crear una carpeta llamada **images**.
2. Ubicar dentro de la carpeta **images**, un archivo de imagen con extensión .png
3. Una vez establecido todo, se debe utilizar la siguiente línea de código en el procedimiento de creación de los componentes de la interfaz.

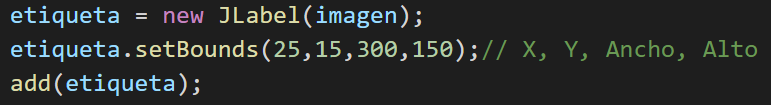


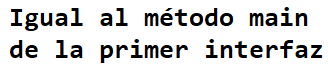
Imágenes

Para agregar una imagen a nuestra interfaz gráfica, se debe utilizar un JLabel.

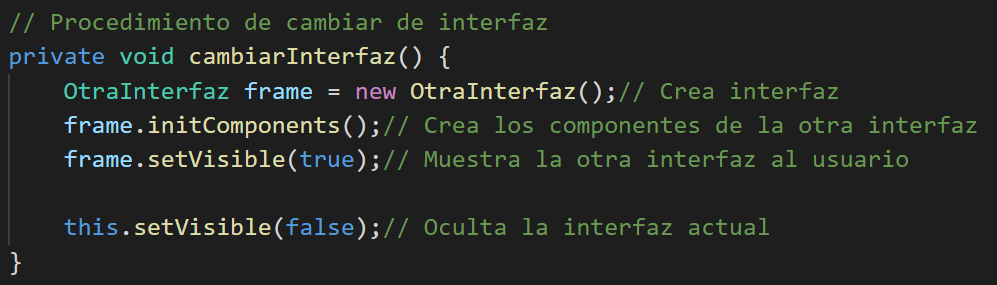
Al igual que para agregar un ícono, se debe crear la carpeta **images** con un archivo de imagen en su interior, por lo tanto, los primeros 3 pasos son idénticos.

4. Crear un ImageIcon:

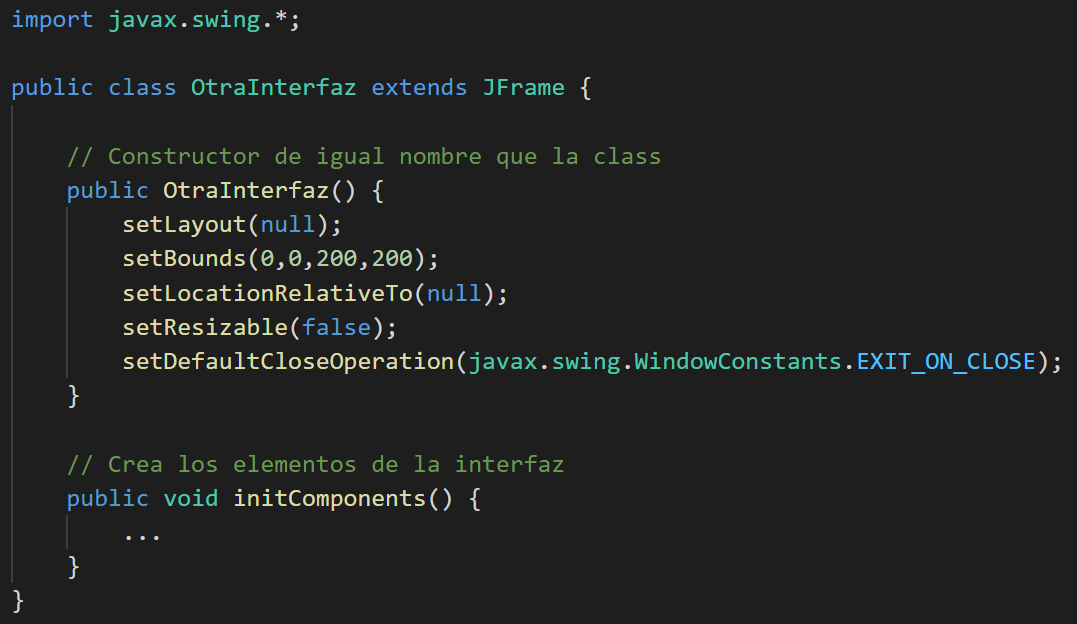
5. Crear una etiqueta (JLabel) e inicializarlo con la imagen.

Conexión entre interfaces gráficas

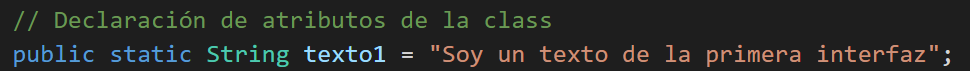
Se puede pasar de una interfaz gráfica a otra mediante el ocultamiento de la interfaz actual y creación de otra interfaz tal cual se hizo con la primera.

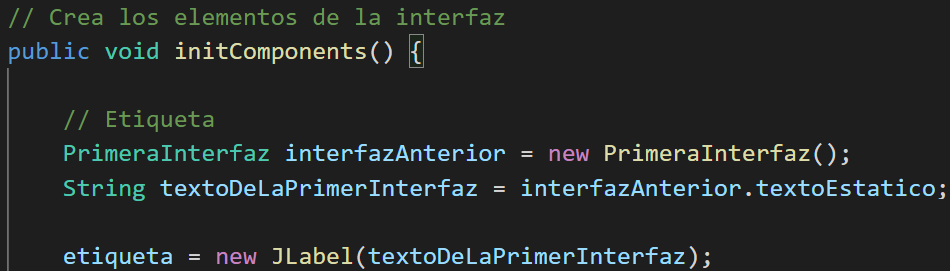


En la class creada con la otra interfaz, no debe existir el método main.



Si se desean pasar datos de una interfaz a otra, se puede realizar una instancia de clases y usar variables estáticas.





Si la interfaz anterior no se utilizará más y desea liberar espacio en memoria, en vez de utilizar **.setVisible** puede utilizar **.disponse()**

Dialogs

Se puede crear una ventana de diálogo simple mediante la línea de código utilizando la biblioteca:

* Información

JOptionPane.showMessageDialog(null, "Ventana de información.","Info", JOptionPane.INFORMATION\_MESSAGE);

* Advertencia

JOptionPane.showMessageDialog(null, "Ventana de advertencia.","Warning", JOptionPane.WARNING\_MESSAGE);

* Error

JOptionPane.showMessageDialog(null, "Ventana de error","Error", JOptionPane.ERROR\_MESSAGE);

Función para establecer un limite de caracteres en una caja de texto

    public static class JTextFieldLimit extends javax.swing.text.PlainDocument {

        private int limit;

        JTextFieldLimit(int limit) {

            super();

            this.limit = limit;

        }

        @Override

        public void insertString(int offset, String str, javax.swing.text.AttributeSet attr)

        throws javax.swing.text.BadLocationException {

            if(str == null) return;

            if((getLength() + str.length()) <= limit) {

                super.insertString(offset, str, attr);

            }

        }

    }

Se aplica utilizando el método:

Funcion para crear una barra de progreso

    public static class ProgressBar extends JFrame {

        private NombreClass frame;

        private javax.swing.JLabel encryptingTitle;

        private javax.swing.JProgressBar pb;

        private String title = "Title";

        public ProgressBar(NombreClass aThis) {

            this.frame = aThis;// obtiene el ID de la class desde la cual se lo llama

            this.setResizable(false);

            t = new javax.swing.JLabel();

            pb = new javax.swing.JProgressBar();

            //setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

            setFocusable(false);

            setFocusableWindowState(false);

            t.setFont(new java.awt.Font("Arial", 0, 12)); // NOI18N

            t.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);

            t.setText("Encrypting...");

            t.setMinimumSize(new java.awt.Dimension(129, 14));

            javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());

            getContentPane().setLayout(layout);

            layout.setHorizontalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()

                .addContainerGap()

                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

                    .addGroup(layout.createSequentialGroup()

                        .addComponent(t, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

                        .addGap(17, 17, 17))

                    .addGroup(layout.createSequentialGroup()

                        .addComponent(pb, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)

                        .addGap(10, 10, 10)))

                .addContainerGap())

            );

            layout.setVerticalGroup(

            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()

                .addContainerGap()

                .addComponent(t, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

                .addGap(18, 18, 18)

                .addComponent(pb, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)

                .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))

            );

            pb.setStringPainted(true);

            pb.setValue(0);

            pb.setString("100%");

            pack();

            this.setTitle(title);

        }

        public void showFrame() {

            this.setVisible(true);

            this.setLocationRelativeTo(null);

            this.setAlwaysOnTop(true);

        }

        public void setPercent(int min, int max) {

            if(min < 0) min = 0;

            int p = min\*100/max;

            pb.setValue(p);

            pb.setString(p+"%");

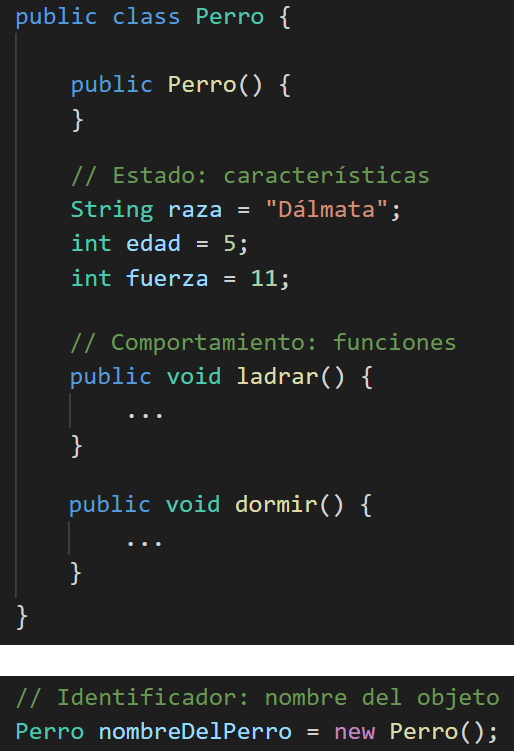
        }

        public void closeFrame() {

            this.setVisible(false);

        }

Programación Orientada a Objetos

Paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora utilizando solamente **subrutinas** y tres **tipos de estructuras**:

* **Secuencia**
* **Selección** (*if, switch*)
* **Iteración** (*for, while, do-while*)

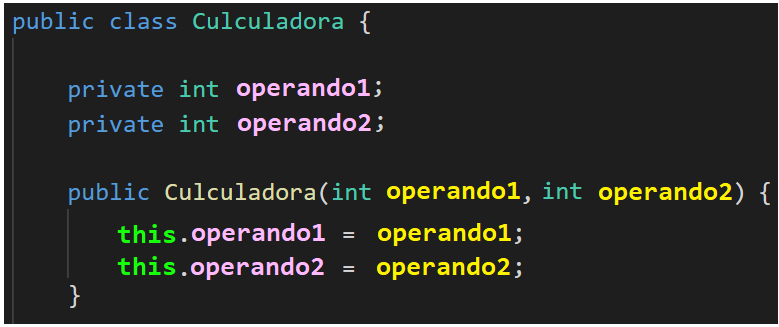
Se trata de una manera de diseñar y desarrollar software que trata de imitar la realidad tomando algunos conceptos esenciales de ella.

* **Objeto**
  + Identidad nombre del objeto.
  + Estado características del objeto.
  + Comportamiento métodos y funciones.

En la programación orientada a objetos, se pueden utilizar variables que se encuentren en otras clases. Para ello se debe crear una instancia de clases.

Instancia de clases

Una instancia de clases es cuando hacemos que dos clases interactúen entre sí.



Al acceder a variables de instancia de una clase, podemos encontrarnos con variables que se llaman igual que en la clase de donde estamos accediendo.

Cuando esto sucede, se puede usar la palabra **this** ya que esta indica si se están utilizando las variables de instancia o las variables de la clase desde la cual estamos accediendo.

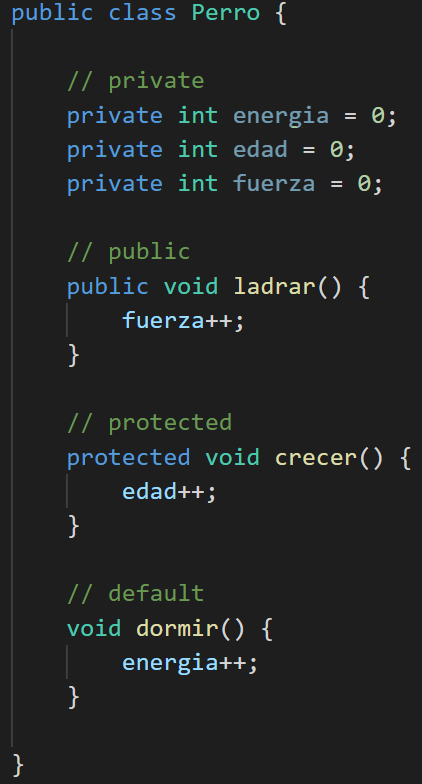
Componentes de una class

Las clases están compuestas por:

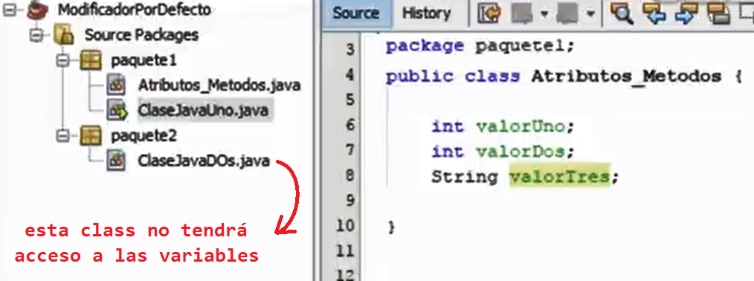
* **Atributos** variables pertenecientes a la class.
* **Métodos** funciones y procedimientos de la class.

Encapsulamiento

Consiste en controlar el acceso a los datos, que conforman un objeto o instancia de clase. Es decir, indica cuáles componentes de una class pueden ser accedidos e incluso modificados.

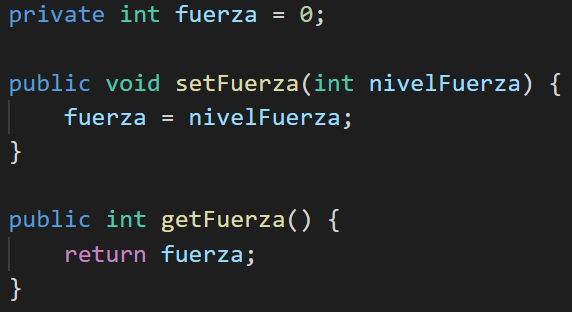
Con el encapsulamiento se pueden ocultar los atributos y métodos de una clase para evitar que sean modificados desde otra clase para evitar cambios imprevistos e incontrolados que puedan provocar un cambio en la funcionalidad de dicha clase.

Para el encapsulamiento, se utilizan **modificadores de acceso**.

* **public** permite el acceso a cualquier otra class.
* **private** sólo tiene acceso la class a la cual pertenece.
* **protected** tienen acceso las class heredadas.
* **package-private** (**default**)sólo tienen acceso las class ubicadas en el mismo subdirectorio o paquete.

Método getter y setter

Son métodos públicos de una class que sirven para otorgar acceso a algunos atributos de dicha class sin eliminar el encapsulamiento.

* **set** procedimiento que permite asignar un valor a un atributo privado de la class.
* **get** función que permite obtener el valor de un atributo privado de la class.

Herencia

Procedimiento utilizado para reutilizar código cuando creamos nuevas clases, ya que permite utilizar atributos (variables) y métodos (funciones y procedimientos) de una clase que ya hemos creado anteriormente y colocarlos dentro de una nueva clase.



* **Clase padre / base**

Es la clase que se debe crear primero, donde se encuentra escrito el código que contiene a los atributos y métodos que se van a reutilizar.

* **Clase hija / derivada**

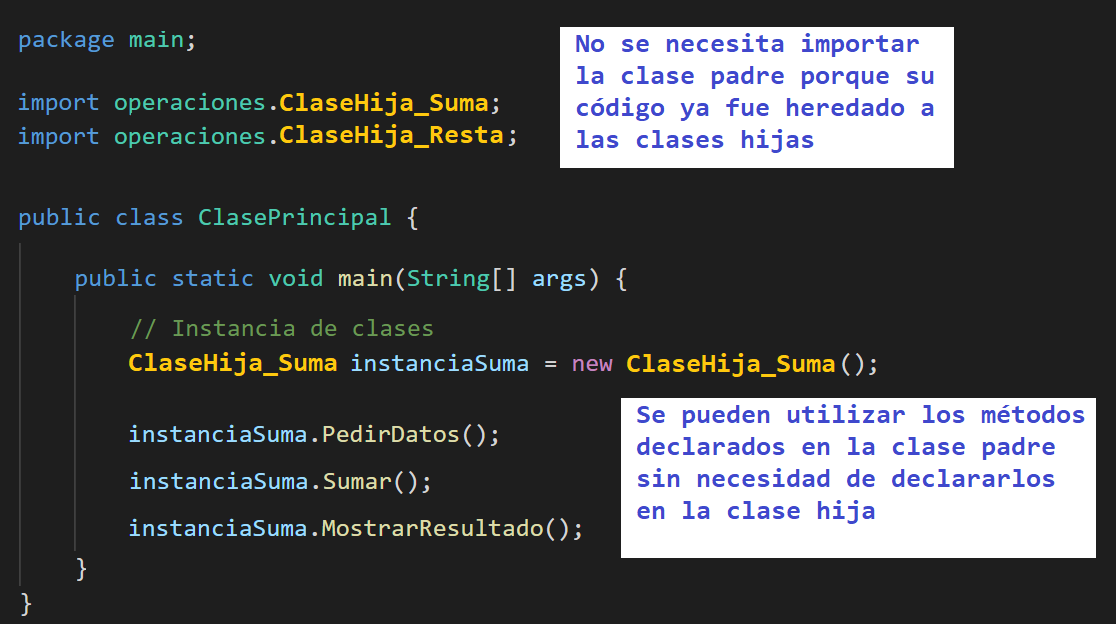
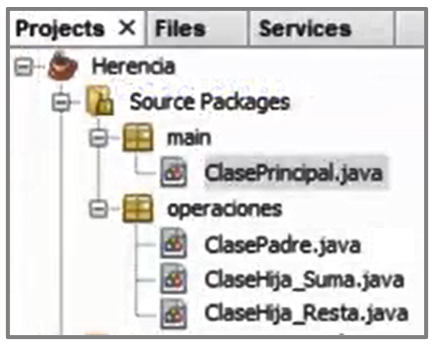
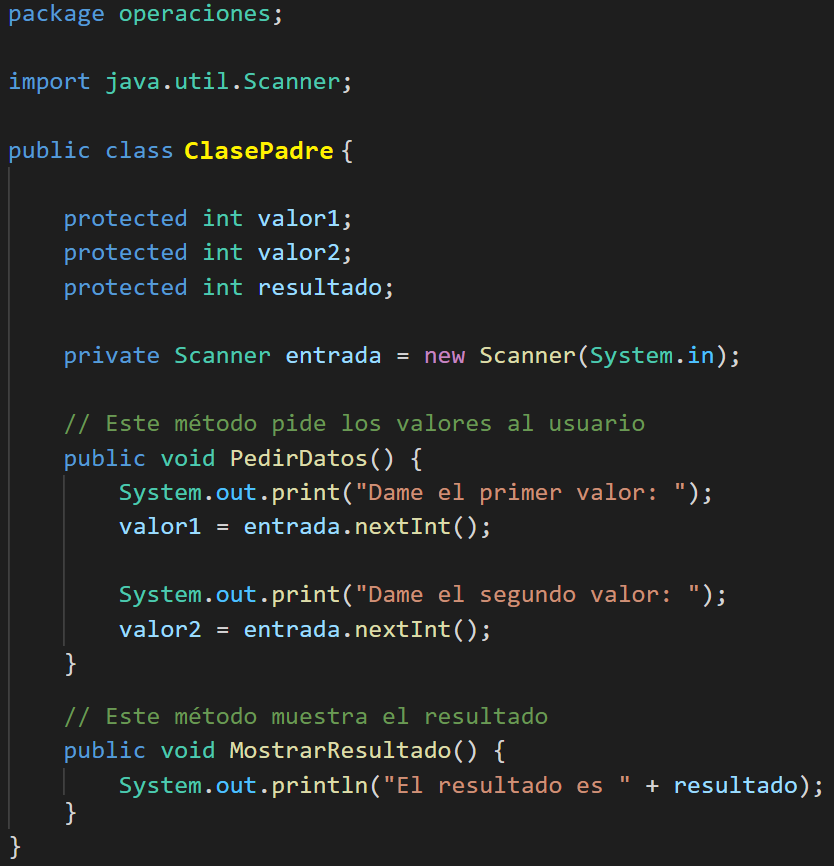
Es la clase que se creará luego, donde se reutilizarán los atributos y métodos que se crearon en la clase padre / base sin necesidad de volver a escribir el mismo código para poder utilizarlos.

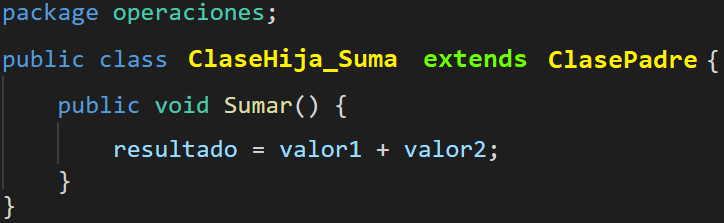
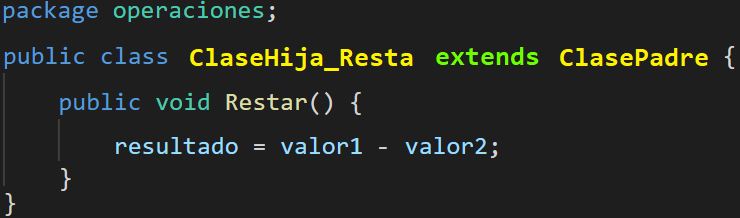
La clase hija sólo puede heredar los atributos y métodos de una única clase padre.

Es buena práctica establecer el modificador de acceso **protected** sólo a los atributos de la clase padre / base, mientras que a los métodos se les establece como **public**.

La palabra **extends** indica que se quiere heredar el código de una clase.





**Se pueden utilizar las variables de la clase padre sin necesidad de declararlas en las clases hijas.**



**CLASE PRINCIPAL** (donde se encuentra el método main)

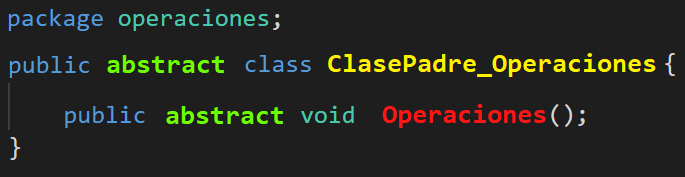
Polimorfismo

Consiste en hacer que un método se comunique con clases diferentes y, dependiendo de la clase con la que se esté comunicando, su comportamiento será completamente diferente.

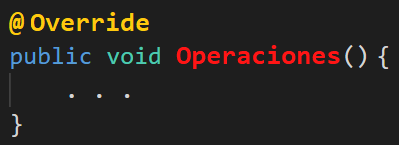
Para hacer uso del polimorfismo, es necesario hacer uso de la herencia (**extends**).

La palabra reservada **abstact** indica que se trabajará con polimorfismo.

* En la clase padre, se declara el método sin la parte lógica.

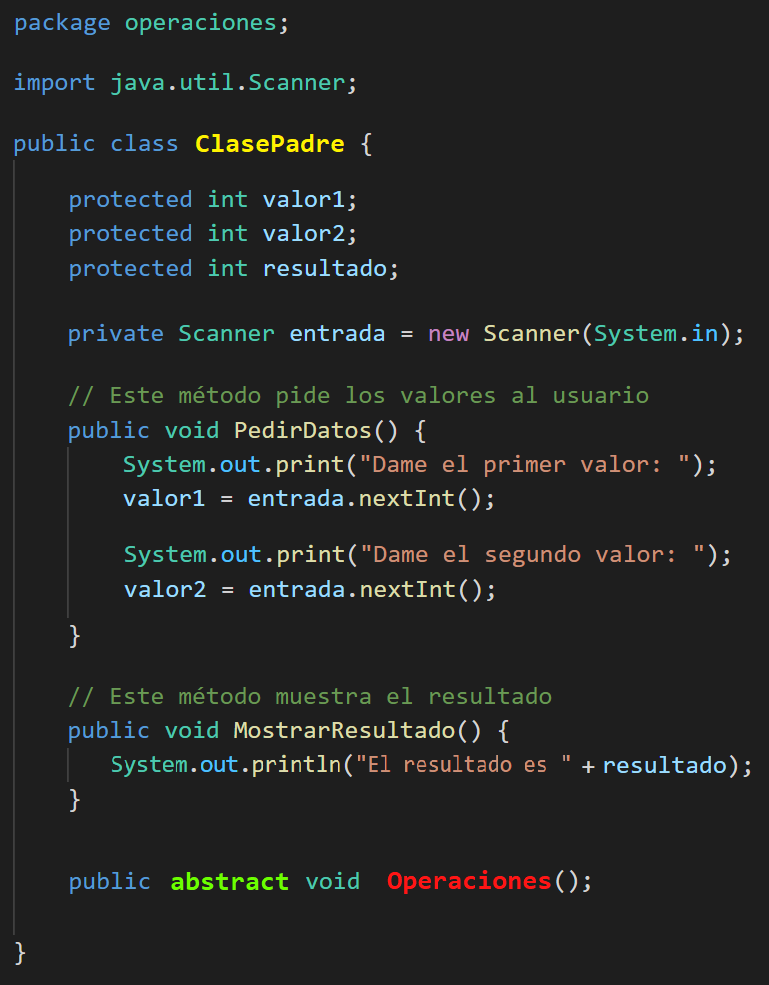
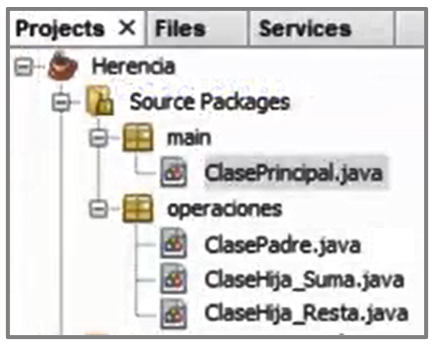


* En la clase hija, se **sobrescribe el método** utilizando la palabra **@Override** y se agrega la parte lógica del método para esa clase.

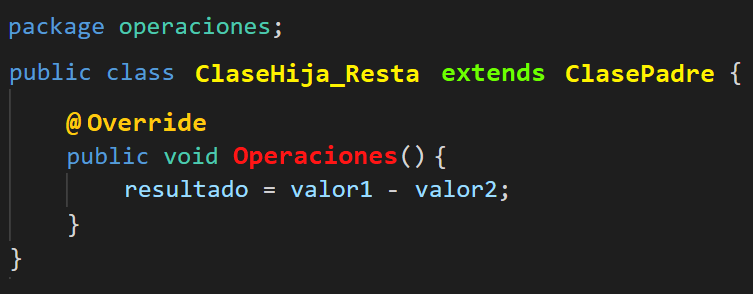


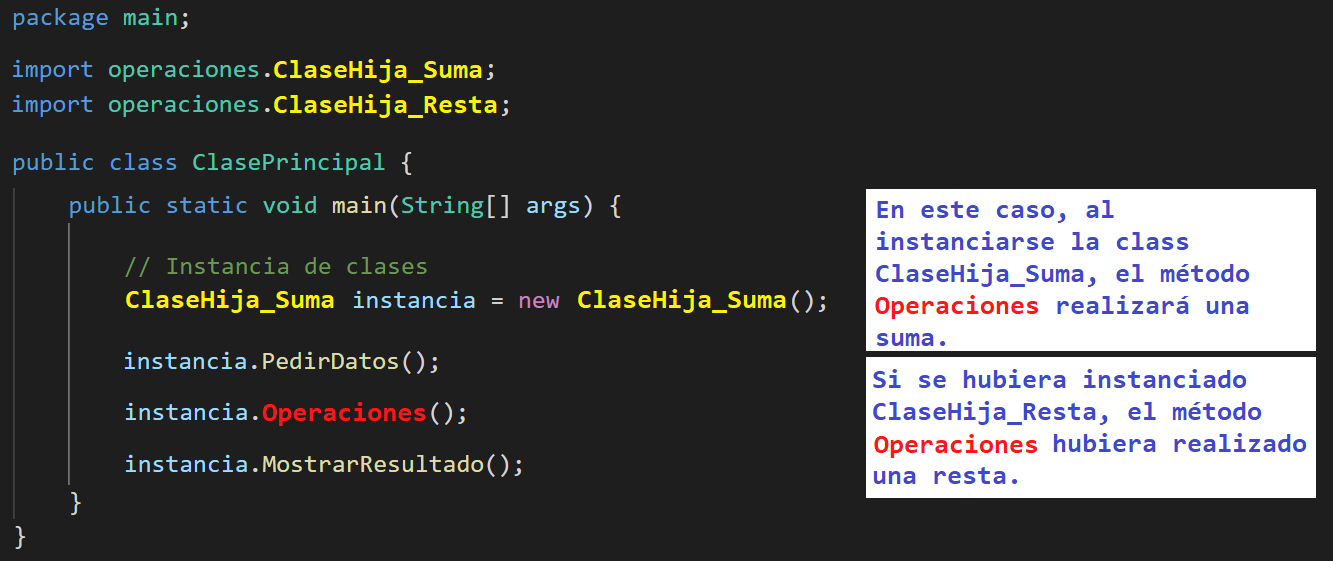
De esta manera, se puede utilizar el mismo método en distintas clases, sin embargo en cada clase tendrá una funcionalidad diferente.

Así, cuando se realice una instancia de clases, se podrán utilizar los métodos de la clase padre definidos con su parte lógica y métodos abstractos definidos en la clase hija (la cual es a la que estaremos comunicándonos mediante la instancia de clases).







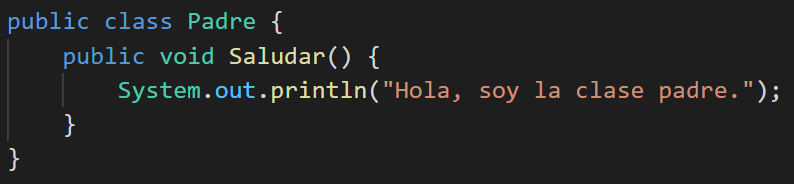


Palabra reservada **super**

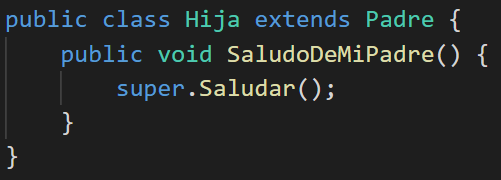
Se utiliza para acceder a un elemento de la clase padre en una herencia.

El uso más común es el de invocar al constructor de la clase padre.

1. Crear un método en la clase padre.



1. Crear un método en la clase hija, el cual llamará al método de la clase padre mediante la palabra reservada **super**.



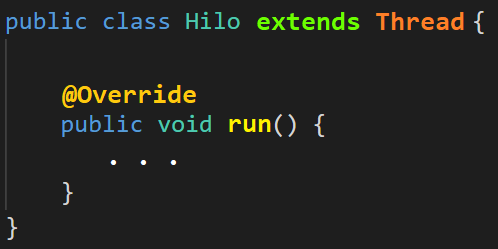
Hilos

Es un flujo de control que permite tener múltiples procesos de forma simultánea.

Con ayuda del **thread**, se puede tener más de un proceso ejecutándose al mismo tiempo en vez de tener que esperar a que finalice uno para que inicie el siguiente.

**Para que funcionen correctamente, es necesario crear todos los hilos y posteriormente iniciarlos todos juntos.**

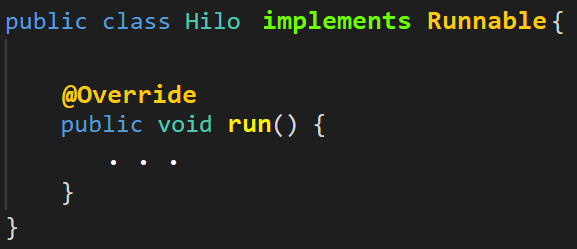
Existen dos maneras de crear un hilo:

* Por herencia de la clase **Thread**

Se debe sobrescribir el método **run** para ejecutar el hilo, el cual es un método de la clase heredada.

Desde otra class se debe crear una instancia de clases.

Luego, una vez creadas todas las instancias de clase para todos los hilos que deseemos, se los debe iniciar.

* Por implementación de la interfaz **Runneable**

Se utiliza **implements** sobre la class en que deseamos tener el hilo, luego sobrescribir el método **run** para ejecutar el hilo.

Desde otra class se debe crear un objeto de tipo **Thread** con la class del Hilo como parámetro **new**.



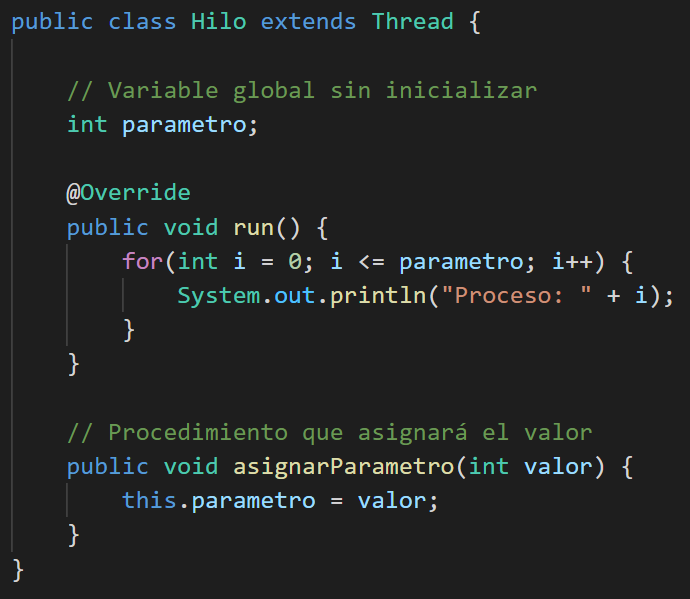
Luego, una vez creados todos los objetos de los hilos, se los debe iniciar.

* Dos procesos sin threads: se envía siempre el Proceso1 primero y el Proceso2 después.
* Dos procesos con threads: no siempre se ejecutan los procesos en el mismo orden.

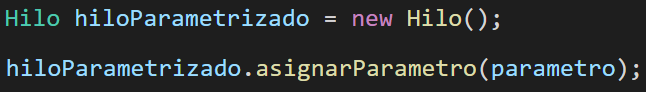
Pasaje de parámetros a un Hilo

Para ello se necesita declarar una variable de ámbito global (sin inicializar) en la class donde está alojado el **thread**, la cual contendrá el valor que pasaremos por parámetros.

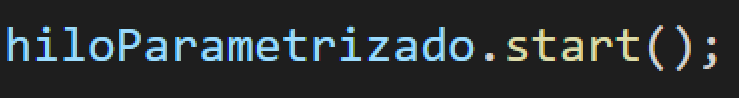
Para que la variable creada reciba el parámetro, se debe crear un procedimiento el cual asigne dicho parámetro a la variable.



Ahora, se puede pasar el parámetro por medio de la instancia de clases.



No hay que olvidar que, para que el hilo se ejecute, hay que iniciarlo.

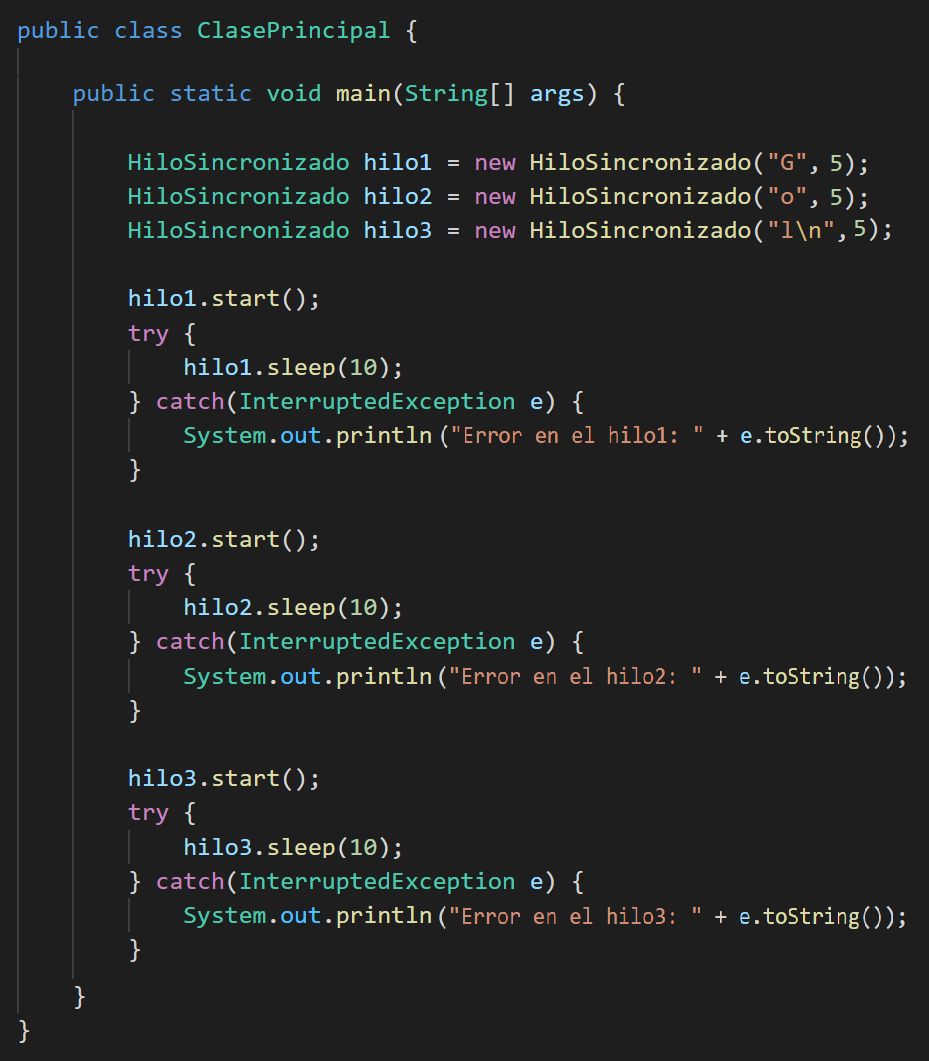
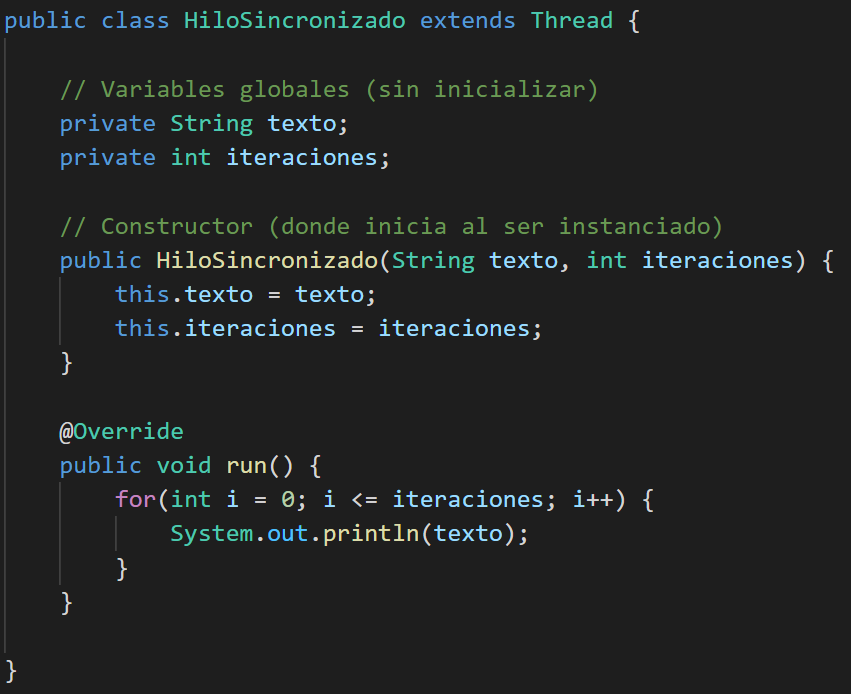
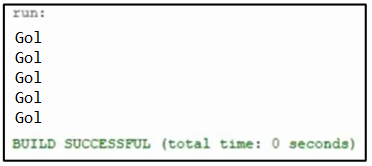


Estados de un Hilo

* **New** El thread fue creado pero no iniciado (start no fue ejecutado).
* **Ejecutable** Cuando el thread fue iniciado (start fue ejecutado) y comienza el método **run**.
* **Bloqueado** El hilo se encuentra en ejecución pero existe una tarea o actividad del mismo hilo que impide que continúe.
  + **sleep(***milisegundos***)** El hilo se “duerme” durante los segundos establecidos. Requiere **try – catch**.
* **Muerto** Se produce cuando su método **run** termina normalmente o cuando existe una instrucción que obligue al hilo a finalizar sin haber concluido su tarea por completo.

Sincronización de hilos

Permite controlar el tiempo y forma de ejecución de varios hilos ejecutándose de manera simultánea. Su finalidad es evitar que un hilo entorpezca a otro hilo al momento de estar ejecutando sus respectivas tareas o bien para establecer un orden de ejecución.

Para lograrlo, se necesita utilizar el método **sleep** tras ser iniciado un hilo, estableciendo un estado durmiente de unos pocos milisegundos.

**Listener**

Los escuchadores o listener son formas en que una clase puede comunicarse con otra a tiempo real, por lo general por medio de hilos (threads).

Para ello, se utiliza en la clase creada el siguiente bloque de código:



**private** OnListenerResult **onListenerResult**;  
**public interface** OnListenerResult {  
 **void** funcionEnviada1(**int** unParametroEnviado);  
 **void** funcionEnviada2();  
}  
**public void** setOnListenerResult(OnListenerResult listener) {  
 **this**.**onListenerResult** = listener;  
}

Luego, dentro de esta misma clase creada, se debe utilizar la siguiente función para que el listener se active

**onListenerResult**.funcionEnviada1(1234);

Finalmente, en la activity donde se quiere recibir dicha información, se debe utilizar el listener asignado a la clase o activity creada.



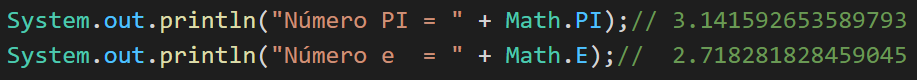
**final** MainActivity mainActivity = **this**;  
OtraActivity otraActivity = **new** OtraActivity(**...**);

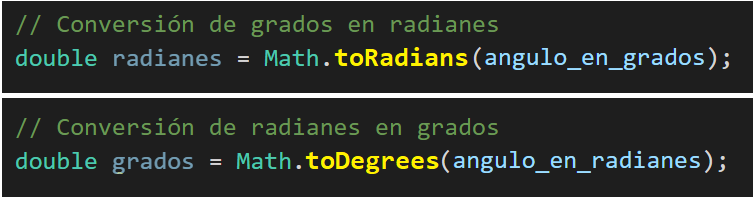
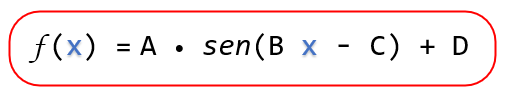
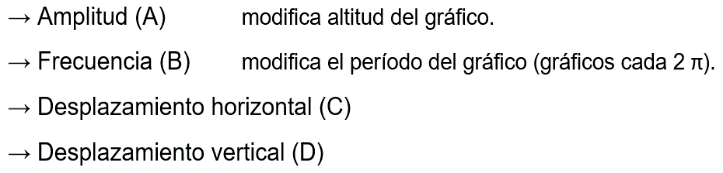
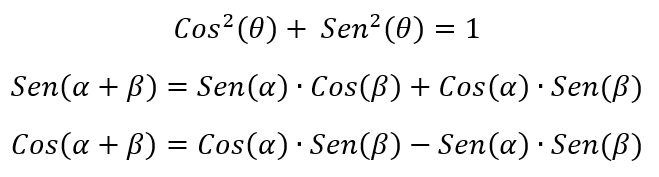
otraActivity.setOnDialogResult(**new** BasicDialog.OnDialogResult() {  
 @Override  
 **public void** funcionEnviada1(**int** unParametroEnviado) {  
 *// Acciones que realizará esta función cuando sea llamada*  
 *// El parámetro recibido será el mismo que se especificó antes*  
 }  
  
 @Override  
 **public void** funcionEnviada2() {  
 *// Acciones que realizará esta función cuando sea llamada*  
 }  
});

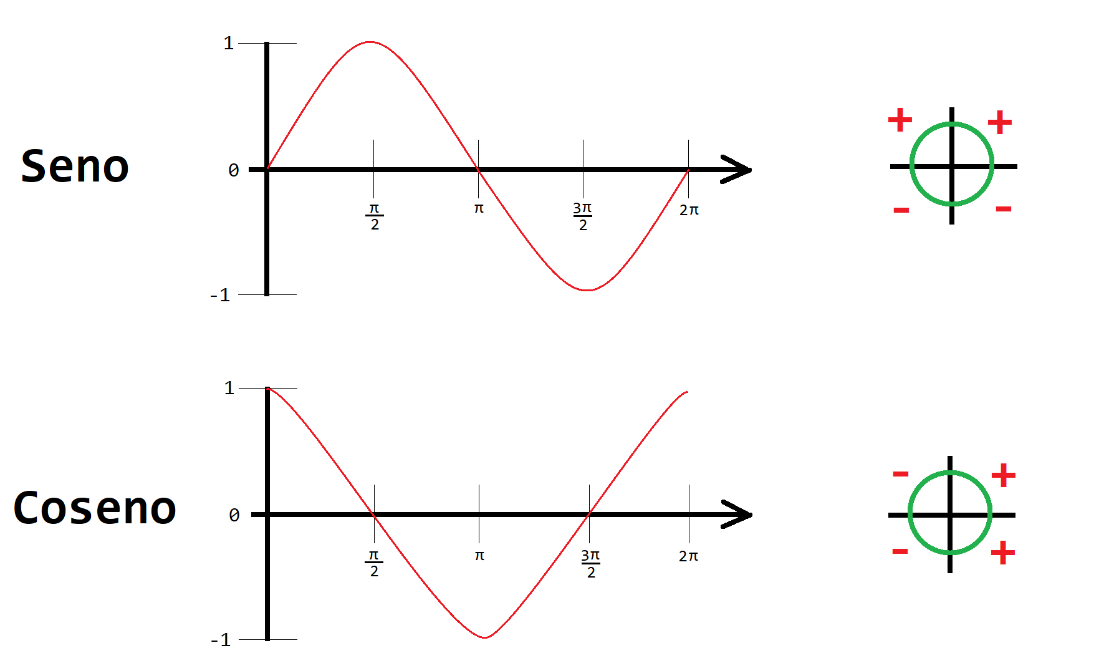
Math

Es una class nativa de Java que permite realizar operaciones matemáticas. Se trabaja con el tipo de dato **double**, números reales de hasta 15 decimales.

La class Math se puede utilizar sin necesidad de importarla.



* Raíces cuadradas
* Conversión de ángulos
* Funciones trigonométricas: **utiliza ángulos en radianes**.



|  |  |
| --- | --- |
|  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |

Control de decimales

* **DecimalFormat**

Se puede recurrir a esta biblioteca para disminuir la cantidad de decimales mostrados.

Luego, se crea una instancia de clases donde, como parámetro, se establecerá la cantidad de decimales que deseamos mostrar con el siguiente formato (la cantidad de 0 es la cantidad de decimales):



Finalmente, se utiliza el método **.format**

* **String format**

Si se desea expresar el número en un string, se puede utilizar este método para controlar los decimales a mostrar (indicado con el ***número***).



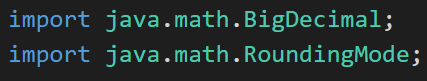


* **Math round**

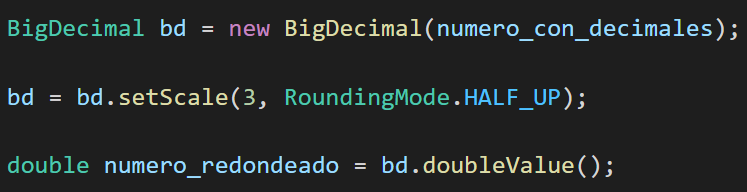
Teniendo un número con decimales, se aplica el casting **(double)** ya que la clase Math trabaja únicamente con este tipo de datos.



Donde la cantidad de ceros es la cantidad de decimales que se mostrará (debe usarse la misma cantidad tanto en la multiplicación como en la división.

* **BigDecimal**

Se puede recurrir a esta clase:

Luego, mediante una instancia de clase con el número como parámetro, se aplica el método de redondeo deseado y se aplica sobre una variable.

Números aleatorios

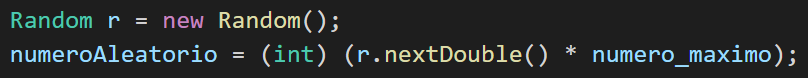
Para crear un número aleatorio se puede utilizar la clase Math con el método:



De esta manera, se genera un número desde el **0** hasta el **numero\_maximo**.

Otra manera de generar un número aleatorio es con la class Random.

Luego, mediante una instancia de clases, se puede generar el número aleatorio de la siguiente manera:

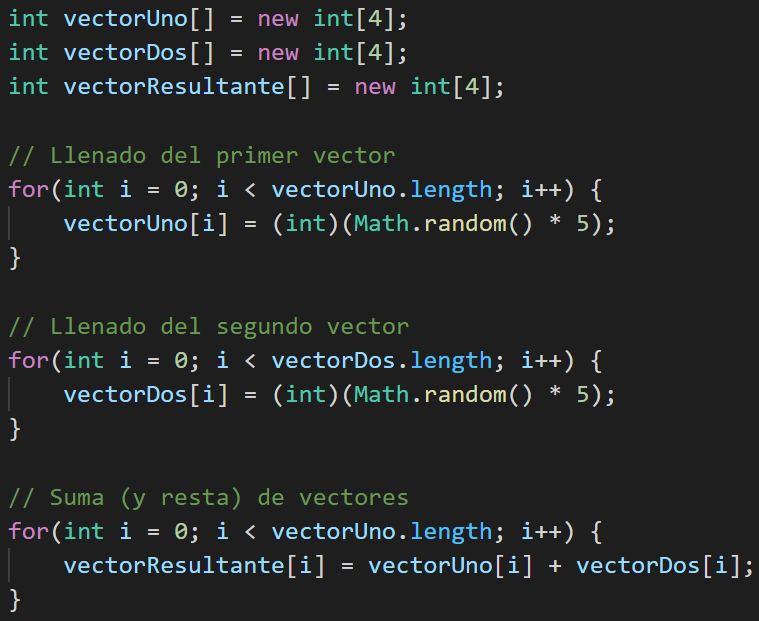


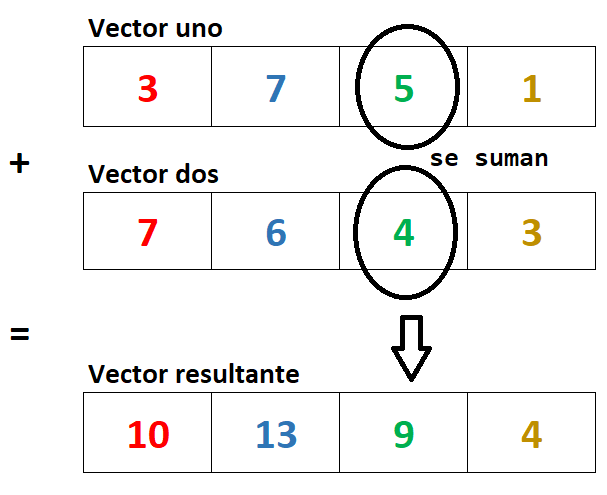
Vectores

* **Suma (y resta)**

La suma de vectores (traslación) se realiza sumando elemento a elemento del vector.

Los vectores a operar deben tener la misma dimensión.

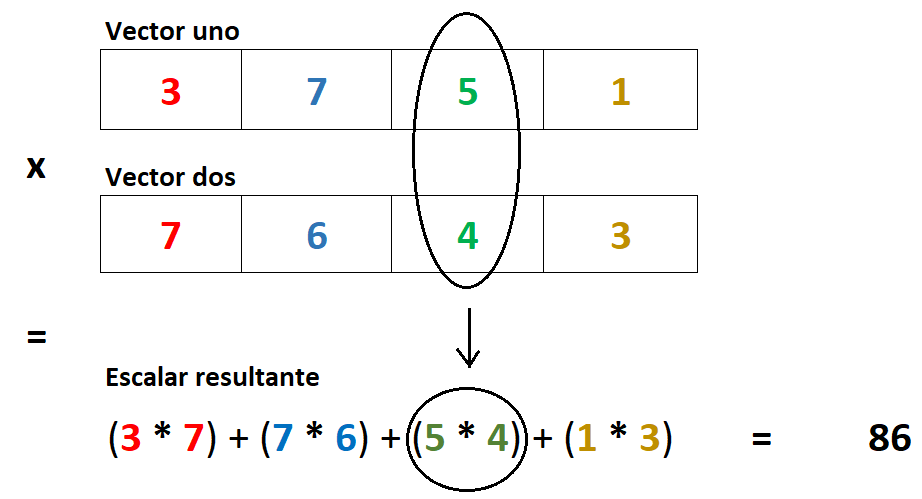


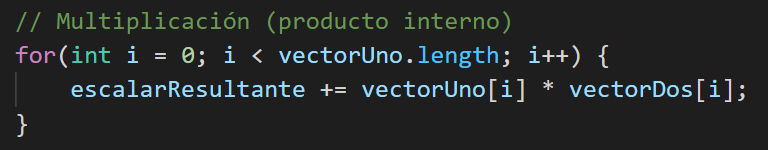


* **Multiplicación**

Para multiplicar vectores, se realiza la multiplicación entre cada elemento de los vectores y, al resultado de dicha multiplicación, se la suma al resultado de la multiplicación del siguiente par de elementos.

El resultado de la multiplicación de dos vectores, es un número escalar.



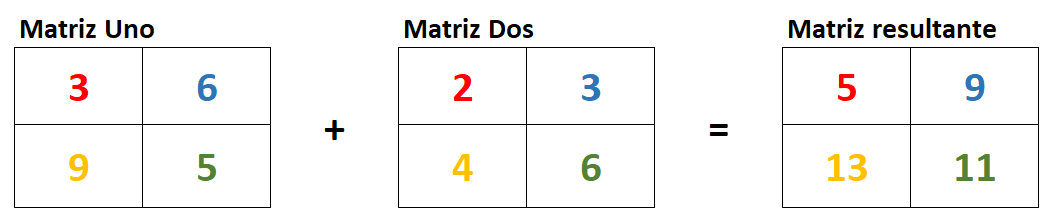


Matrices

* **Suma (y resta)**

La suma de matrices solo se puede realizar entre matrices de la misma dimensión, es decir, deben tener la misma cantidad de filas y columnas. Así mismo, la matriz resultante mantendrá las mismas dimensiones.

La suma se efectúa en cada uno de los elementos correspondiente a cada matriz en su posición actual:



Se sumará matrizUno**[0,0]** con matrizDos**[0,0]** y el resultado se asignará en matrizResultante**[0,0]**.



int matrizUno[][] = new int[3][3];

        int matrizDos[][] = new int[3][3];

        int matrizResultante[][] = new int[3][3];

        // Llenado de la primer matriz

        for(int i = 0; i < matrizUno.length; i++) {

            for(int j = 0; j < matrizUno.length; j++) {

                matrizUno[i][j] = (int)(Math.random() \* 5);

            }

        }

        // Llenado de la segunda matriz

        for(int i = 0; i < matrizDos.length; i++) {

            for(int j = 0; j < matrizDos.length; j++) {

                matrizDos[i][j] = (int)(Math.random() \* 5);

            }

        }

        // Suma (y resta) de matrices

        for(int i = 0; i < matrizDos.length; i++) {

            for(int j = 0; j < matrizDos.length; j++) {

                matrizResultante[i][j] = matrizUno[i][j] ± matrizDos[i][j];

            }

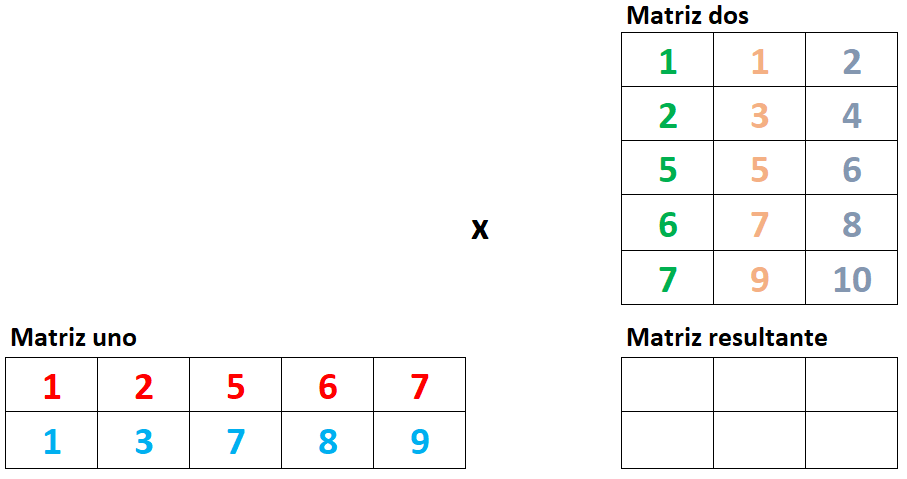
        }

* **Multiplicación de matrices**

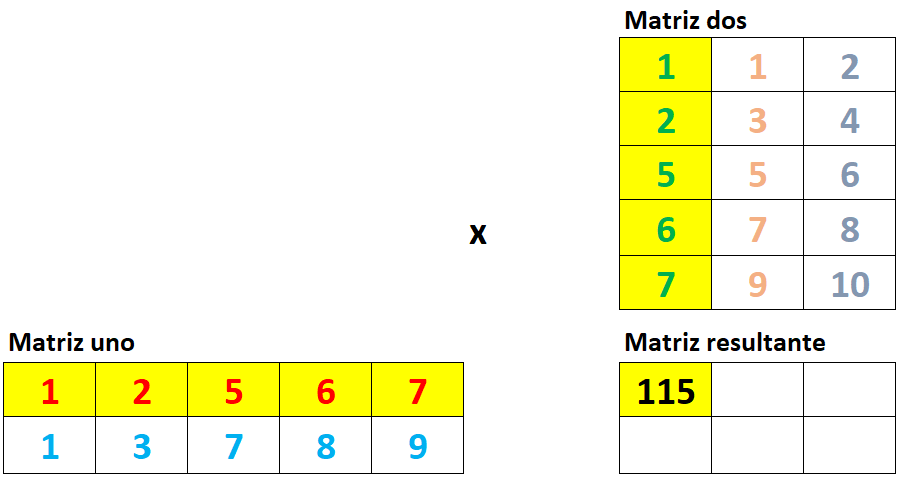
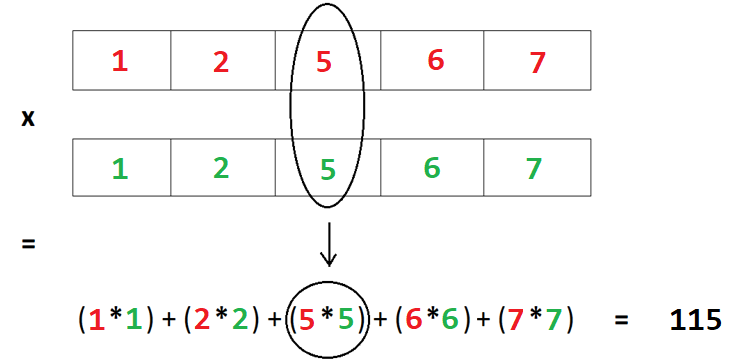
Para multiplicar matrices, es necesario que:

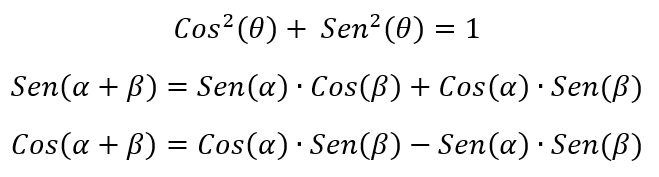
* El **número de columnas** de la **primer matriz** sea igual al **número de filas** de la **segunda matriz**.

La matriz resultante tendrá el número de filas igual al número de filas de la primer matriz, y el número de columnas igual al número de columnas de la segunda matriz.



Una forma de representarlo y comprenderlo adecuadamente es la siguiente:

Se multiplica la primera fila de la primera matriz con la primera columna de la segunda matriz para obtener el primer elemento de la matriz resultante. La multiplicación se efectúa tal como se hace con vectores.



        // Multiplicar matrices

        int matrizResultante[][];

        if(matrizUno[0].length == matrizDos.length) {

            matrizResultante = new int[matrizUno.length][matrizDos[0].length];

            for(int fila = 0; fila < matrizResultante.length; fila++) {

                for(int columna = 0; columna < matrizResultante[0].length; columna++) {

                    // Multiplicar fila de matrizUno con columna de matrizDos

                    int escalar = 0;

                    for(int k = 0; k < matrizDos.length; k++) {

                        escalar += matrizUno[fila][k] \* matrizDos[k][columna];

                    }

                    matrizResultante[fila][columna] = escalar;

                }

            }

        } else throw new IndexOutOfBoundsException("Columnas de matrizUno no es igual a Filas de matrizDos");

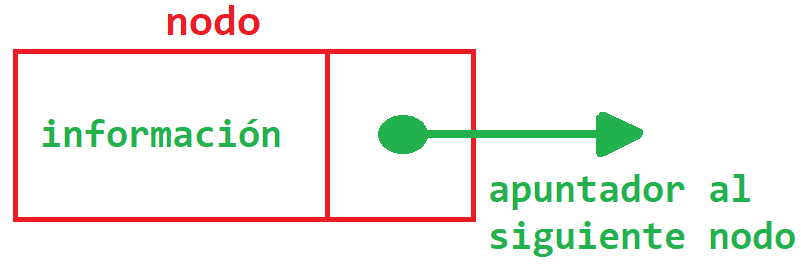
Estructura de datos

Listas

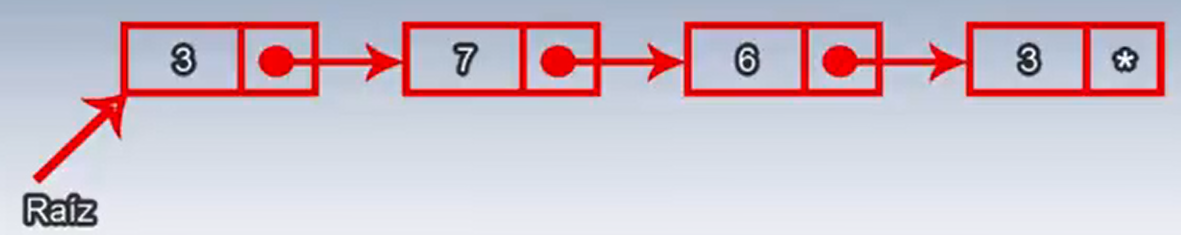
Tipo de dato **abstracto** que permite almacenar datos de una forma organizada al igual que los vectores, pero estos tienen una **estructura** **dinámica**.

Cada elemento de la lista (**nodo**) tiene dos campos:

* Información Dato a almacenar.
* Enlace Apuntador al siguiente elemento, con excepción del último que enlaza un **null**.



Representación de una lista



public class Nodo {

    int informacion;// Dato del elemento

    Nodo enlace;// Siguiente elemento de la lista

    public Nodo(int informacion) {

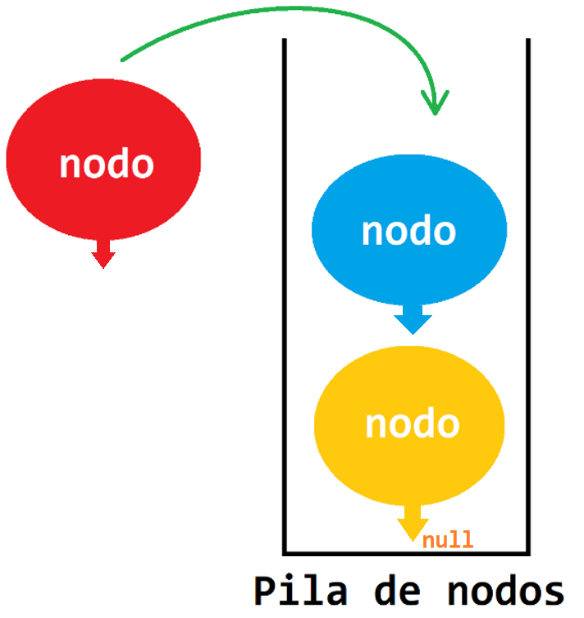
**this.**informacion = informacion;

        enlace = null;

    }

}

Dependiendo del procedimiento de inserción y extracción de nodos en la lista, existen distintos tipos de listas.

* **Lista de tipo pila**

Las inserciones y extracciones se realizan por el mismo lado de la lista.

El último elemento en entrar a la lista es el primero en poder salir.

public class Pila {

    private Nodo ultimoNodo;

    int tamano = 0;

    // Inicialización de la lista

    public Pila() {

        ultimoNodo = null;

        tamano = 0;

    }

    public boolean EstaLaPilaVacia() {// Método para saber si la pila está vacía

        return ultimoNodo == null;

    }

    public void InsertarNodo(int informacion) {// Método para ingresar un nodo en la pila

        Nodo nuevoNodo = new Nodo(informacion);

        nuevoNodo.enlace = ultimoNodo;

        ultimoNodo = nuevoNodo;

        tamano++;

    }

    public int EliminarNodo() {// Método para borrar el último nodo añadido a la pila

        if(!EstaLaColaVacia()) {

        int informacion = ultimoNodo.informacion;

        ultimoNodo = ultimoNodo.enlace;

        tamano--;

        return tamano;

        }

    }

    public int UltimoValorIngresado() {// Método para obtener el último valor ingresado

        return ultimoNodo.informacion;

    }

    public int TamanoPila() {// Método para conseguir el tamaño de la pila

        return tamano;

    }

    public void VaciarPila() {// Método para vaciar la pila

        while(!EstaLaPilaVacia()) {

            EliminarNodo();

        }

    }

    public int[] getPila() {// Método para obtener el contenido de la pila

      Nodo nodo = ultimoNodo;

        int pila[] = new int[tamano];

        int i = 0;

        while(nodo!= null) {

            pila[i] = nodo.informacion;

            nodo = nodo.enlace;

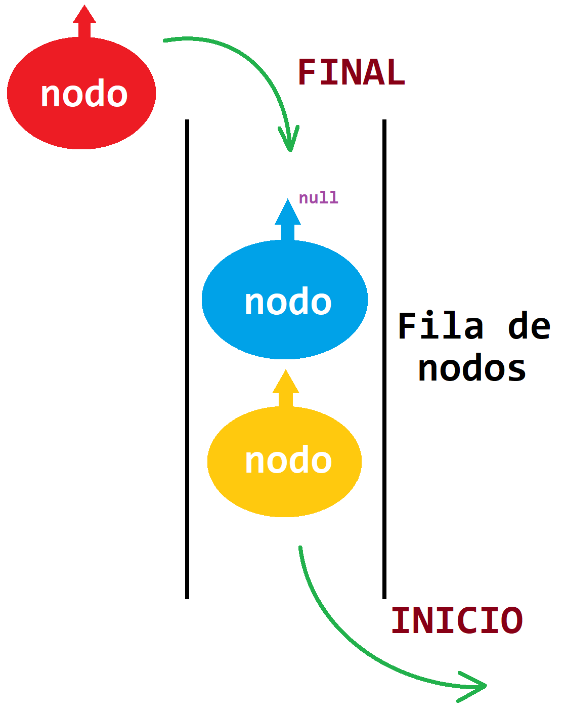
            i++;

        }

        return pila;

    }

}

* **Lista de tipo cola (FIFO)**

Las inserciones se realizan en el inicio de la lista y las extracciones se realizan al final de la lista.

El último elemento en entrar es el último en poder salir.

public class Cola {

    private Nodo inicioCola, finalCola;

    private int tamano = 0;

    public Cola() {// Inicialización de la lista de tipo Cola

        inicioCola = null;

        finalCola = null;

    }

    // Método para saber si la cola está vacía

    public boolean EstaLaColaVacia() {

        return inicioCola == null;

    }

    public void InsertarNodo(int informacion) {// Método para insertar elemento a la cola

        Nodo nuevoNodo = new Nodo(informacion);

        if(EstaLaColaVacia()) {

            inicioCola = nuevoNodo;

            finalCola = nuevoNodo;

        } else {

            finalCola.enlace = nuevoNodo;

            finalCola = nuevoNodo;

        }

tamano++;

    }

    public int EliminarNodo() {// Método para borrar el nodo al inicio de la cola

        if(!EstaLaColaVacia()) {

            int informacion = inicioCola.informacion;

            if(inicioCola == finalCola) {

                inicioCola = null;

                finalCola = null;

            } else {

                inicioCola = inicioCola.enlace;

            }

tamano--;

            return informacion;

        } else {

            return Integer.MAX\_VALUE;

        }

    }

    public int TamanoCola() {// Método para conseguir el tamaño de la cola

        return tamano;

    }

    public int[] getCola() {// Método para obtener el contenido de la cola

        Nodo nodo = inicioCola;

        int cola[] = new int[tamano];

        int i = 0;

        while(nodo != null) {

            cola[i] = nodo.informacion;

            nodo = nodo.enlace;

            i++;

        }

        return cola;

    }

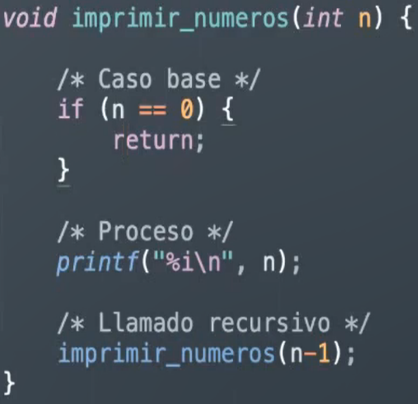
}

Recursividad

Técnica que permite que un bloque de instrucciones codificadas se ejecute una cierta cantidad de veces, logrando en ocasiones reemplazar a las estructuras repetitivas como **while** y **for**.

Para lograr esto, los métodos deben llamarse (invocarse) a sí mismos. Se puede controlar la cantidad de ejecuciones mediante estructuras condicionales.

Un procedimiento recursivo está dividido en tres fases principales.

* **Caso base**

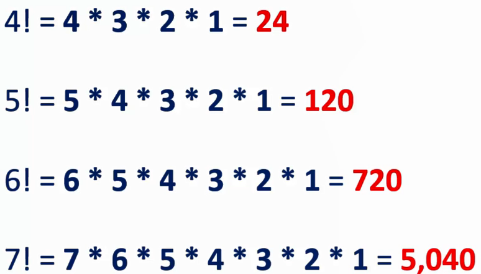
Cuando termina la recursión.

* **Proceso**

Valor agregado o acción de la función.

* **Llamada recursiva**

La función/procedimiento se llama a sí mismo

Ejemplo para el cálculo de un número factorial (n!)

El factorial de un número es un valor que se obtiene como resultado de la multiplicación de todos los números enteros que anteceden al número del cual se desea conocer su factorial, a excepción del cero.

    public int Factorial(int **n**) {

        int resultado = 1;

        if(**n** > 0) {

            resultado = n \* Factorial(**n** - 1);

        } else if(**n** == 0) {

            resultado = 1;

        } else {

            resultado = Integer.MAX\_VALUE;

        }

        return resultado;

    }

Archivos

Se puede trabajar con archivos externos mediante la biblioteca File.

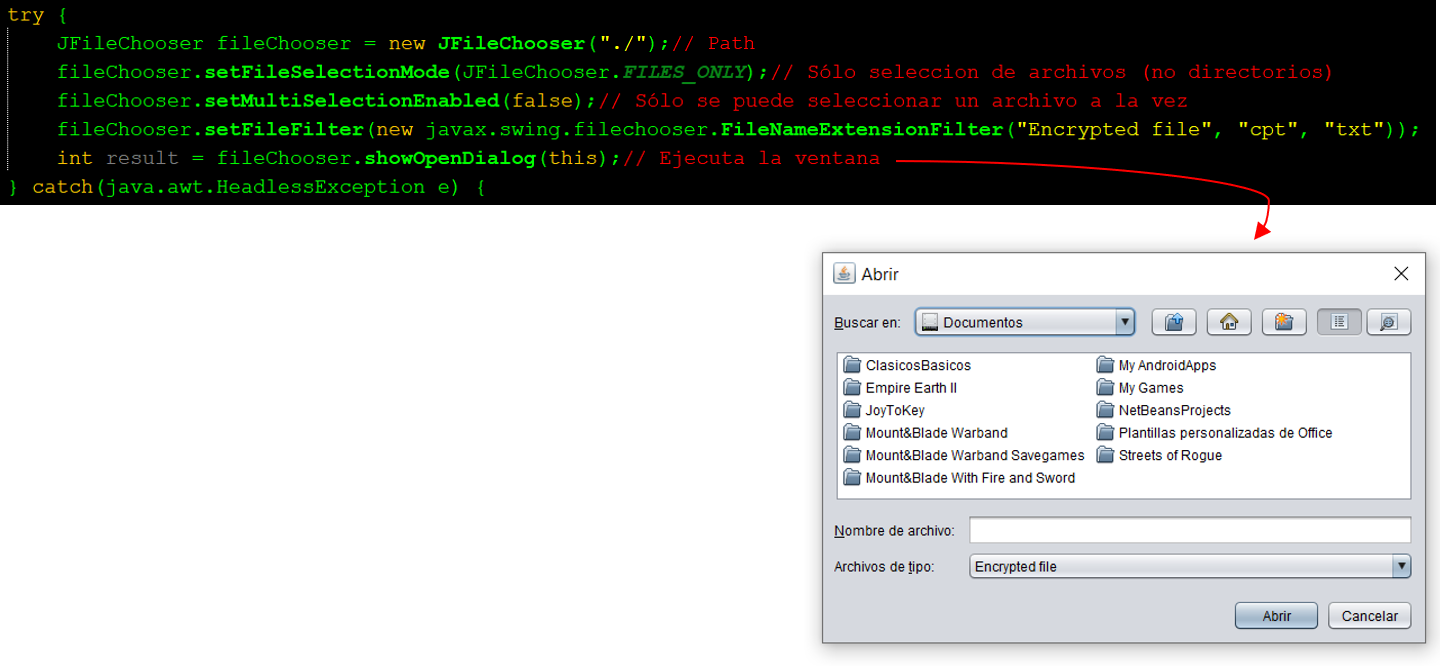
Para ello, es necesario saber los siguientes conceptos:

* Path
* Absoulte path
* Canonical path
* Name
* Parent

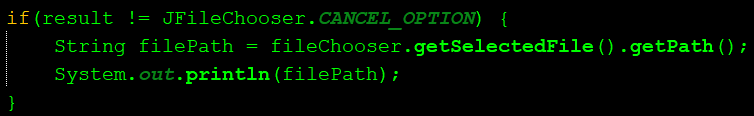
Apertura de un archivo externo con explorer.exe

Se puede abrir una carpeta con la siguiente línea de código:



Buscador de archivos externos

Para obtener el file seleccionado, se utiliza el siguiente condicional:



Bases de datos

Colección de datos almacenados de forma organizada con una lógica coherente.

Permite compartir estos datos entre distintos usuarios y programas, con lo cual se facilita el intercambio y consulta de información.

* Registros
* Campos

Para crear una base de datos, es necesario utilizar un programa externo:

