DOCUMENTACION

JAVA POO

autor; Benjamin Ponce

fechaEmision; 05-12-2024

horaEmision; 03:47

fechaCulmino;

ÍNDICE

[Conceptos claves 4](#_Toc196267934)

[Tipografía 4](#_Toc196267935)

[¿Qué es una clase? 4](#_Toc196267936)

[¿Qué es una variable? 4](#_Toc196267937)

[¿Qué es una entidad? 4](#_Toc196267938)

[¿Qué es un objeto? 4](#_Toc196267939)

[Tipos de datos: 5](#_Toc196267940)

[Primitivos: ejemplos 5](#_Toc196267941)

[Por referencia: ejemplos 5](#_Toc196267942)

[Características de una clase 6](#_Toc196267943)

[Atributos: 6](#_Toc196267944)

[Constructores: 6](#_Toc196267945)

[Métodos/Conmutadores: 6](#_Toc196267946)

[Creación de proyecto en java 7](#_Toc196267947)

[Como crear un proyecto: 7](#_Toc196267948)

[Creación de un paquete 8](#_Toc196267949)

[Creación de una clase 8](#_Toc196267950)

[pruebas unitarias 9](#_Toc196267951)

[¿Qué son las pruebas unitarias? 9](#_Toc196267952)

[Características clave 9](#_Toc196267953)

[Características de JUNIT 9](#_Toc196267954)

[Anotaciones 9](#_Toc196267955)

[Uso de Assertions 10](#_Toc196267956)

[¿Como se hacen estas pruebas unitarias? 11](#_Toc196267957)

[Resolución ejemplo toString: 13](#_Toc196267958)

[API 16](#_Toc196267959)

[¿Qué es una API? 16](#_Toc196267960)

[¿Por qué se usa una API? 16](#_Toc196267961)

[Componentes de una API: 16](#_Toc196267962)

[Conexión a base de datos 17](#_Toc196267963)

[Clase conexión 18](#_Toc196267964)

[Crear una base de datos en netbeans 20](#_Toc196267965)

[Buenas pracaticas DB 22](#_Toc196267966)

[Modelado de datos: 22](#_Toc196267967)

[DAO 23](#_Toc196267968)

[DAO Interface 23](#_Toc196267969)

[Como se crea 23](#_Toc196267970)

[Creamos la interface DAO 24](#_Toc196267971)

[MODELTM 25](#_Toc196267972)

[Crear package 25](#_Toc196267973)

[Crear clase TM 26](#_Toc196267974)

[EJEMPLO DE UNA TABLE MODEL 27](#_Toc196267975)

# Conceptos claves

## Tipografía

La Tipografía utilizada en este lenguaje de programación es **camelCase**

## ¿Qué es una clase?

Una clase es una plantilla en donde se definen sus características, podemos instanciar un objeto y ver su comportamiento. Se utilizan para modelar entidades del mundo real o incluso en uno abstracto.

## ¿Qué es una variable?

Una variable es un espacio en memoria reservado para almacenar un valor. Cada variable posee un nombre(identificador) y un tipo de dato (puede ser String, int, boolean, etc) determina el tipo de información que puedes almacenar.

## ¿Qué es una entidad?

Una entidad representa algo del mundo real. Por ejemplo, una clase llamada “Persona”, sus características pueden ser género, edad y nombre y sus comportamientos respirar, moverse, caminar y pensar.

## ¿Qué es un objeto?

Un objeto es la instancia de una clase. Es importante saber que estas instancias de clases poseen atributos y métodos, estos objetos son entidades del mundo real

* Se crean a partir de una clase (la cual actúa como plantilla)
* Tienen métodos que se pueden ejecutar (el comportamiento o lo que puede hacer)
* Tienen atributos (se les puede llamar campos o propiedades, estos describen sus características)
* Tiene identidad

Ejemplo básico: Clase: Auto   
 objeto: un auto especifico: “Toyota corolla blanco, año 2020”

# Tipos de datos:

## Primitivos: ejemplos

byte = Entero pequeño (8bits) byte x = 100;

short = Entero corto (16bits) shot y = 2000;

int = Entero(32bits) int edad = 25;

long = Entero largo(64bits) long z = 123456789L;

float = Decimal simple (32bits) float pi = 3.14f;

double = Decimal doble precision(64bits) double peso = 70.5;

char = Un solo carácter char letra = ‘ A ‘;

boolean = Verdadero o falso boolean activo = true;

## Por referencia: ejemplos

Estos hacen a objetos o clases

String = Cadena de texto String nombre = “Ana”;

Array = Arreglo de elementos int[] números = {1,2,3};

Clases = cualquier cosa creada por el programador Auto miAuto = new Auto();

# Características de una clase

Una clase es una plantilla en la cual podemos instanciar objetos, las clases poseen características principales las cuales son **siempre lleva la primera letra del nombre con mayúscula**, sus atributos, constructores y métodos o llamados de otra manera conmutadores.

Ahora bien, como identificamos cada uno:

Atributos: Los atributos poseen un alcance inicial el cual se divide entre private o public.

Private: los private o privados en español son aquellos los cuales serán llamados fuera de la clase con un conmutador, un GET (un obtener en pocas palabras), para poder obtener lo que definimos “private int Id”, al llamarlo con un GET obtendremos la Id.

Public: los public no requieren de un método para ser llamados, ya que al ser públicos podemos llamarlos fácilmente fuera de la clase.

Constructores: Los constructores también poseen un alcance, pero estos tienen dos características fundamentales, sabiendo estas características podrás identificar fácilmente un constructor. Características:

1. El nombre del constructor es el mismo que el de tu clase (clase) “public class Producto” (constructor) “public Producto(){

}”.

2- No retornan nada, como estarás viendo en el constructor de tu clase, este no retorna nada, y esa es otra de las características principales para identificar uno, ya que si puedes notar en algún Método GET estos si retornan (GET) “public int getId() { return id;}”.

Métodos/Conmutadores: Los métodos son aquellos los cuales nos ayudan a obtener o cambiar algún valor, entre lo mencionado encontramos los GET y SET o más conocidos como Getter y Setter estos son importantes en tu clase, ya que sin ellos no podrás cambiar o obtener algún valor preasignado o por asignar. Estos no son los únicos métodos, existen otros como los “Override Method” y Eventos los cuales también son métodos estos serán explicados más adelante.

# Creación de proyecto en java

## Como crear un proyecto:

NetBeans: Primero hay que crear un nuevo proyecto, para crear uno nos dirigimos a File y en la primera opción new Project o presionamos Ctrl+Shift+N

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Seleccionas Java With Ant y en la barra derecha java aplication

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Una vez seleccionado recordar nombrar tu proyecto y abajo pregunta en la casilla si deseas crear una Main Class, si la dejas activada se creará y si la desactivas decides tu luego si creas una Main o una App.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

## Creación de un paquete

Creamos un nuevo Java Package , este package será llamado “model”, en este se almacenan nuestras clases.

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Creación de una clase

Ahora creamos la primera clase, para ello nos dirigimos a nuestro paquete model, damos click derecho, seleccionamos “New” y escogemos la opción que diga “Java Class…”

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Una vez seleccionado le darán el nombre a su clase y finish

Recuerda que en el nombre de tu clase siempre la primera letra será **Mayúscula**.

# pruebas unitarias

## ¿Qué son las pruebas unitarias?

Las pruebas unitarias son ejecutadas con Junit y Junit4, estas pruebas nos ayudan a verificar si el código presenta algún tipo de fallo en alguna clase, data, interfaz o incluso en nuestra conexión a nuestra base de datos. Estas pruebas comprueban en nuestra clase atributos, métodos y funciones con el fin de que todo funcione correctamente. Las pruebas ayudan a mejorar la calidad final de nuestro código, permiten poder probar distintas partes del proyecto como se mencionaba anteriormente, ayudan a asegurar que los cambios nuevos a nuestro código no sean defectuosos.

## Características clave

En vista general las características claves

* Posee una estructura sencilla para definir y ejecutar pruebas.
* tiene un soporte para pruebas de regresión y pruebas parametrizadas.
* integración con herramientas de construcción como Maven y Gradle.

## Características de JUNIT

* Simplicidad: Proporciona una API fácil de usar para definir pruebas.
* Anotaciones: Permiten estructurar pruebas de manera clara.
* Ejecutar pruebas: soporta la ejecución de pruebas en IDEs y desde la línea de comandos.
* Reportes de pruebas: Generación de reportes que muestran resultados de las pruebas.

## Anotaciones

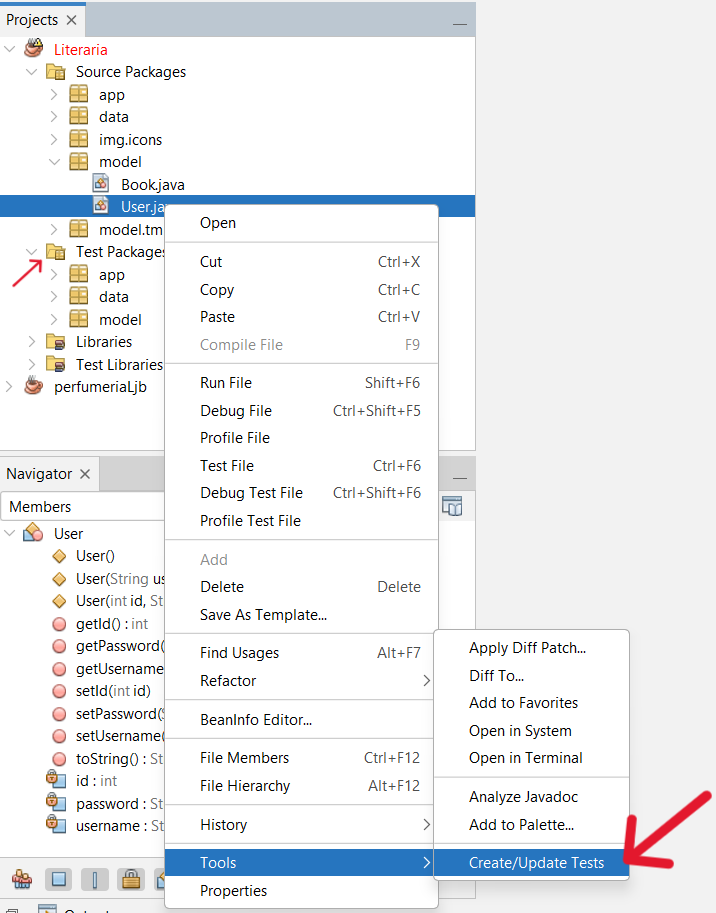
* @Test: Define un método como prueba.
* @Before: Ejecutado antes de cada prueba.
* @After: Ejecutado después de cada prueba.
* @BeforeClass y @AfterClass: Ejecutan código una vez antes o después de todas las pruebas.

## Uso de Assertions

* Definición de Assertions: Métodos que verifican condiciones en las pruebas.
* assertEquals(expected, actual): Verifica si dos valores son iguales.
* assertTrue(condition): Verifica si una condición es verdadera.
* assertNotNull(object): Verifica que un objeto sea nulo.

## ¿Como se hacen estas pruebas unitarias?

En nuestro código debemos dirigirnos a nuestra clase y hacer click derecho en la que deseamos hacer el test, ir a la penúltima opción “Tools” y seleccionar la última opción “Create/Update Tests”.



Esto hará que se genere un nuevo paquete “Java package” con el nombre de “Test packages” como se puede ver en la foto a la izquierda.

Ahora dentro de la carpeta “model” creada en “Test Packages” estarán las clases, dentro de estas, el código tendrá los métodos para ejecutar las pruebas unitarias. Para ejecutar esto puede hacerlo con SHIFT + F6 o hacer click en la pantalla en un lugar vacío y presionar “Run File”

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Al correr el código se abrirá una mini pantalla en la cual se indicará si pasó o no pasó el test

Ejemplo de test con error:

Interfaz de usuario gráfica, Aplicación, Word

Descripción generada automáticamente

En este caso el test no pasó y muestra que es lo que está dando error en la clase, el cual es un ToString(). Cuando el test pase se mostrará un 100% y sin errores.

Ejemplo de test correcto:

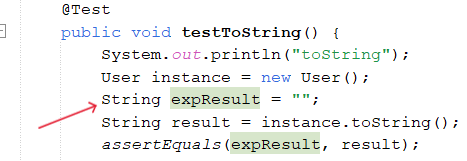
Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

## Resolución ejemplo toString:

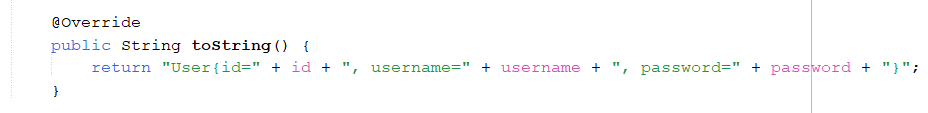
Imaginemos que tenemos una clase llamada “User” la cual es para nuestros usuarios y le hicimos una prueba para verificar si todo estaba correcto, pero el toString nos dice que tiene un error ahora, cual es el error en nuestro código o ¿será que no tenemos bien implementado algo?, eso lo averiguaremos en este ejemplo

En la imagen anterior se vio que el toString no pasa el test y aquí se resolverá para que pase el test. Este error se debe a que en el Test se espera una cadena vacía, lo cual no está recibiendo, porque al ejecutar lo que la prueba o el sistema está generando es esto User{id=0, username=, password=} y esto no es una cadena vacia, ahora para solucionar esto:

FORMA 1: la manera “sencilla” es modificar la prueba para que coincida con nuestro toString(), por lo que se necesita modificar el String expResult = “ ”; que vemos en nuestro método de prueba

En esa línea debemos hacer que coincida con nuestro toString()

Nuestro toString()



para hacerlo debemos escribir en String expResult = “ ”; lo siguiente "User{id=0, username=, password=}";

Texto, Carta

Descripción generada automáticamente

Esto hará que al ejecutar nuestra prueba comparará expResult con nuestro toString() y al ser lo que espera que se devuelva pasará el Test (Lo que esperaba era una cadena vacía).

FORMA 2: la otra manera es cambiar nuestro método toString() para que devuelva una cadena con los valores de nuestros atributos creados en nuestra clase “User”, los cuales son id, username y password. Para ello debemos de sobrescribir nuestro método toString para que devuelva los valores que queremos.

Al sobrescribir nuestro método quedaría algo así

@Override public String toString() {

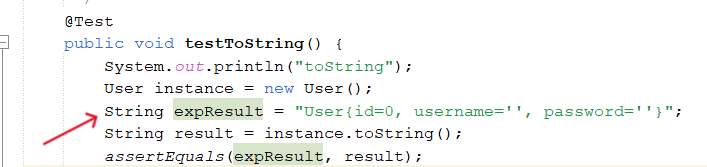
return "User{id=" + id + ", username=' " + username + " ', password=' " + password + " '}";

}

Esto hará que devuelva una cadena en este formato User{id=0, username=’ ‘, password=’ ‘} tomando los valores de nuestros atributos.

Ahora que devuelve los valores de los atributos modificaremos una línea de código en nuestro método @Test la cual será la misma que modificamos anteriormente pero indicando estos valores “User{id=0, username=’ ‘, password=’ ‘}”;

Así es como quedaría en la prueba



Al ejecutar devolverá una cadena con los valores de nuestros atributos y pasará la prueba.

# API

## ¿Qué es una API?

Una API es un intermediario entre un usuario y un sistema, permitiendo la interacción entre ambos sin necesidad de saber como es que es posible tal interacción.

Ejemplo: imaginemos que somos unos usuarios que queremos entrar a nuestra cuenta bancaria, como usuarios no necesitamos saber si el código esta bien hecho o estructurado de buena manera, no necesitamos saber si tiene clases o base de datos. Solo necesitamos entrar a nuestra cuenta y hacer lo que teníamos planeado ya sea una transacción o ver el dinero que poseemos y que la aplicación responda de buena manera dándonos lo solicitado. En pocas palabras una API permite la comunicación entre usuario y sistema sin necesidad de saber como se hace todo por detrás.

Ahora si hablamos de manera técnica una API es un conjunto de reglas que define como un programa puede interactuar con otro programa.

## ¿Por qué se usa una API?

Una API se crea para que los programas se comuniquen entre si de manera sencilla. Como programador no es necesario tener que crear de cero una rueda, si alguien ya la hizo puedes usar una API para acceder a la rueda sin tener que crearla desde cero.

## Componentes de una API:

* Solicitudes (Request): Haces una solicitud a la API para pedir algo.
* Respuesta (response): La API te devuelve lo que pediste.
* Métodos o funciones: Acciones que puedes hacer a través de la API.
* Datos: Lo que envías y recibes en la solicitud/respuesta.

# Conexión a base de datos

La conexión a base de datos puede ser manejada mediante una clase que se encarga de administrar la conexión y las operaciones necesarias sobre nuestra DB. Para lograr esto es necesario el tener una librería descargada la cual se llama MySQL Conector, esta librería nos ayudará a poder tener nuestra conexión a la DB. Para introducir la librería en un proyecto debemos ir al paquete con nombre “Libraries”, click derecho y escoges la penúltima opción “Add JAR/Folder”

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

Cuando selecciones y emita la nueva ventana, debes buscar tu conector y seleccionarlo.

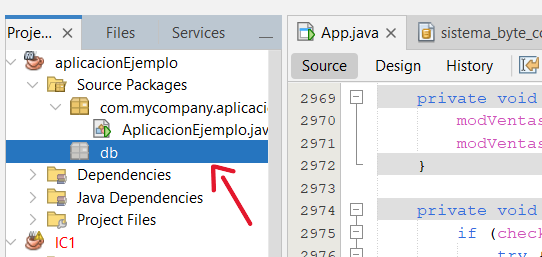
Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

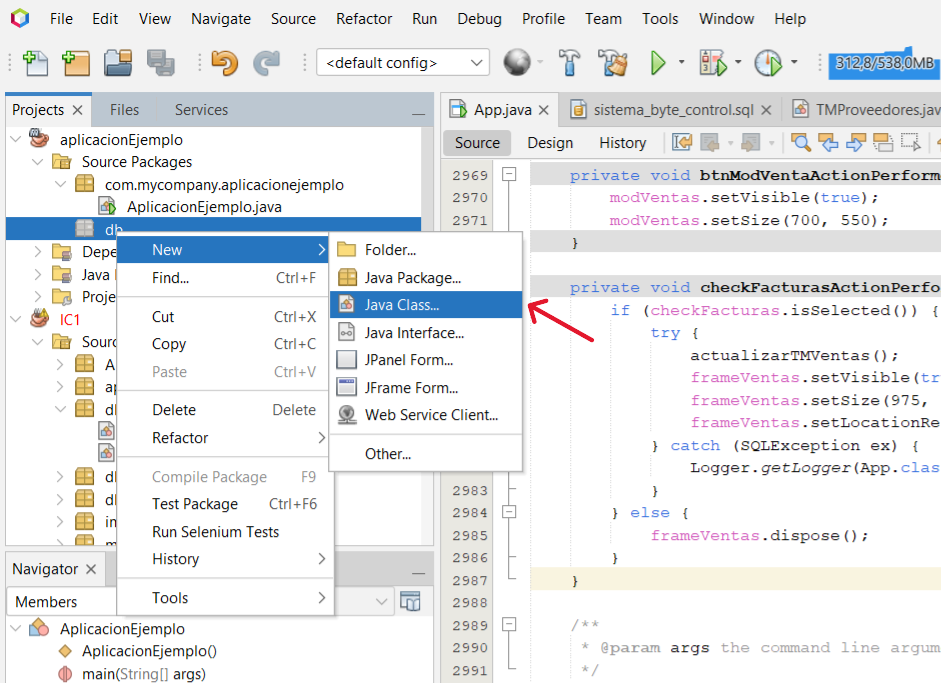
Ya hecho lo mencionado, tendrás tu librería agregada y podrás crear la clase para manejar tu conexión a tu DB.

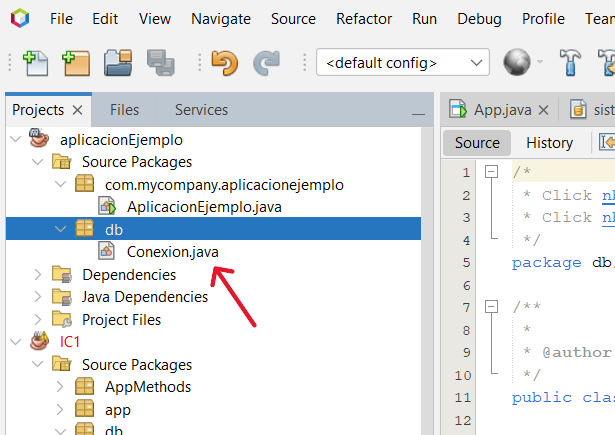
## Clase conexión

Para crear una clase de conexión a base de datos es necesario tener un Package designado a nuestra base base de datos. Entonces creamos el Package “db”



Agregamos al Package db una java class

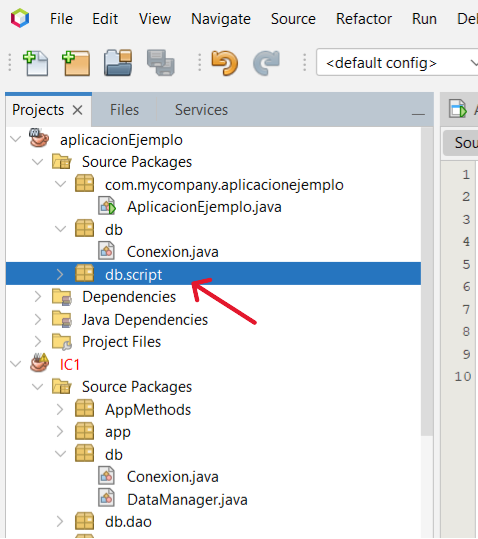


Nombramos a esta clase como “Conexión” **(RECORDANDO QUE EN LAS CLASES, LA PRIMERA LETRA VA CON MAYUSCULA)**

En la clase conexión creada, escribiremos nuestra conexión para que podamos ejecutar consultas e iniciar nuestra base de datos sin problemas.

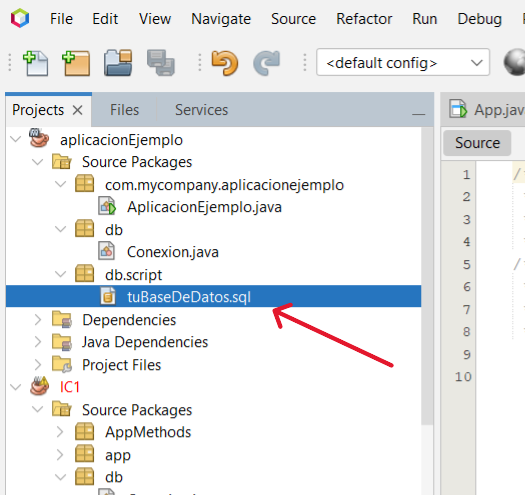
# Crear una base de datos en netbeans

Para crear una base de datos en netbeans debemos de crear un nuevo Package con nombre “db.script” para poder identificar fácilmente donde estarán nuestros scripts de base de datos.

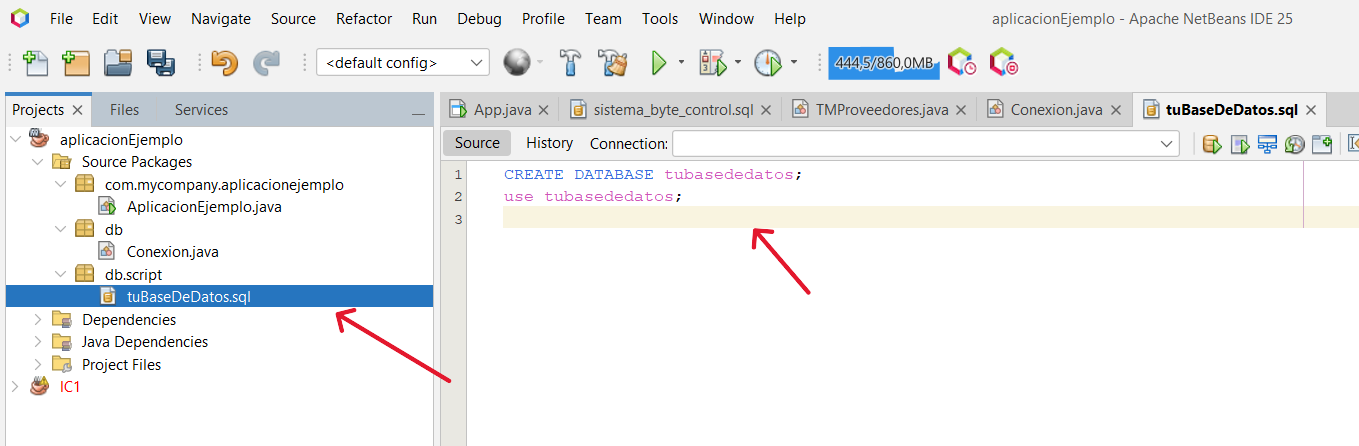


¿Por qué es el “script”?

El”. script” se debe a que es el lugar donde generaremos y crearemos nuestra base de datos, la cual lleva por nombre “tuBaseDeDatos.sql”.



Una vez dentro de nuestro script crearemos nuestra base de datos. Para crear una base de datos utilizamos la palabra clave “CREATE DATABASE tubasededatos” y para usarla “USE tubasededatos”, las tablas por agregar quedaran a gusto de lo que se requiera



## Buenas pracaticas DB

Importante revisar o evaluar Aspectos claves del diseño, administración y optimización de la base de datos. Con las siguientes notas podrás saber si estas siguiendo las buenas practicas.

## Modelado de datos:

* Normalización: Asegúrate que la base de datos cuente con al menos hasta la 3FN para evitar la redundancia de datos y mejore la consistencia.
* Desnormalización: para mejorar el rendimiento de las consultas a veces es necesario el desnormalizar ciertas tablas es común, más aún en bases de datos orientadas a análisis o reportes.
* Claves primarias y foráneas. El uso de claves primarias únicas ayuda a poder identificar de mejor manera cada columna y fila. Las claves foráneas son referenciales entre tablas para mantener la relación entre estas.
* Tipado de datos: Escoger el tipo de dato adecuado para cada columna, un ejemplo el uso de un INT para número enteros y VARCHAR para los textos.

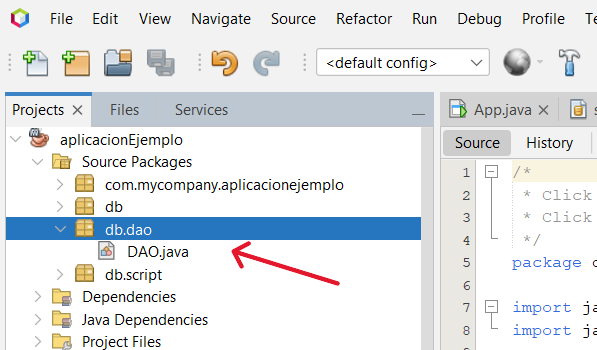
# DAO

Un DAO (Data Access Object) es una clase en programación orientada a objetos que se encarga de gestionar la conexión entre la aplicación y la base de datos. Su función principal es realizar operaciones como insertar, eliminar, actualizar y consultar datos sin que el resto del programa tenga que preocuparse por los detalles de la base de datos. Ayuda a mantener el código organizado y separado.

DAO Interface  
Un **DAO (Data Access Object)** es una **interfaz genérica** que define los métodos básicos para interactuar con una base de datos, como **crear, leer, actualizar, eliminar y listar** objetos.  
La usamos para **separar la lógica del programa** de la lógica de acceso a datos, y hacer el código más **ordenado, reutilizable y fácil de mantener**.

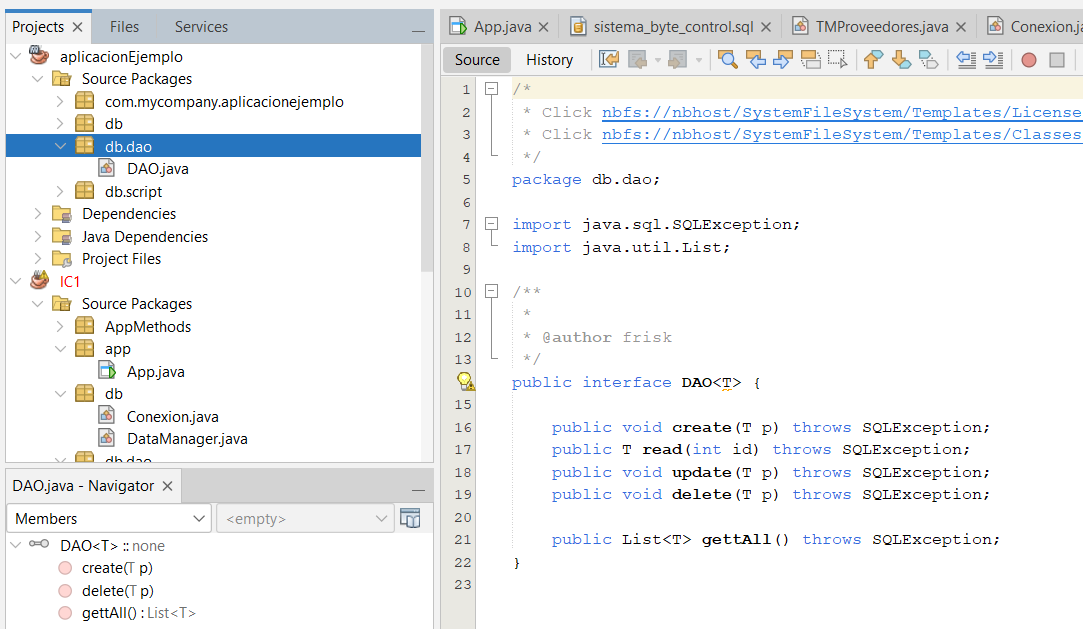
## Como se crea

Creamos un Package con nombre “db.dao” para almacenar nuestra clase DAO



## Creamos la interface DAO

Para crear nuestra interface DAO necesitamos primero comenzar por su alcance el cual es publico para poder llamarlo a otras clases sin dificultad, le decimos a netbeans que nuestro DAO es una interfaz, le ponemos su respectivo nombre agregando <T> para que sea genérica.

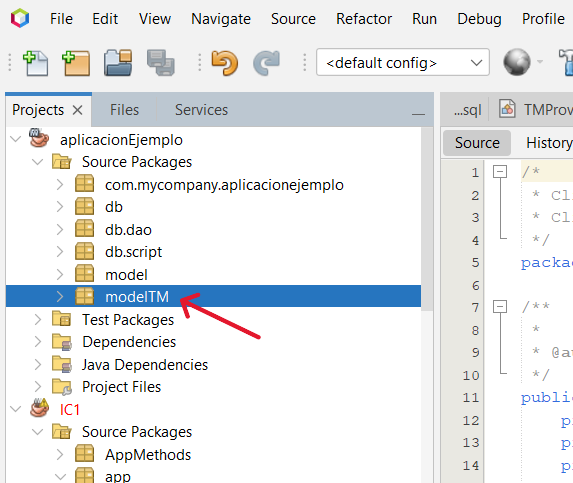


# MODELTM

Model TM, es un Package en donde crearemos nuestras clases en las que queramos mostrar algo en las jTables de java que son utilizadas en la App. Utilizar este Pacakge MODEL TABLE MODEL ayuda a organizar nuestro código de mejor manera, sabiendo así donde buscar cuando tengamos un error o queramos cambiar algo

## Crear package

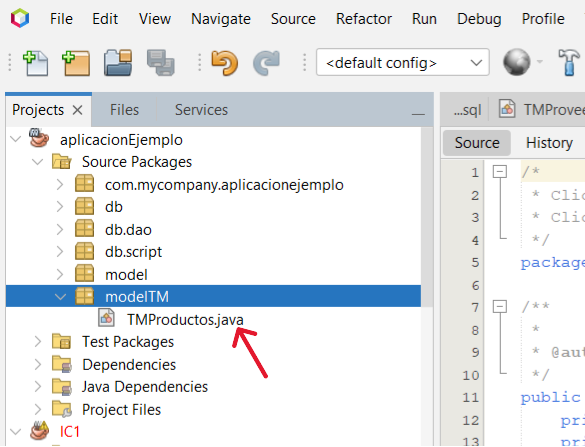
Creamos el package “modelTM” en nuestro source package, aquí es donde crearemos nuestras clases para las TM



## Crear clase TM

Agregamos la Clase de la TM que necesitemos, en este caso utilizaremos una “TMProductos” para el ejemplo. Al momento de crear nuestra clase es importante que le agreguemos un “TM” al inicio (TM = table model) para poder identificar de mejor manera nuestrar tm y por buenas prácticas ejemplo:

Nuestra TM se tratará sobre productos, entonces yo creo una TMProductos, en caso que tratara sobre alimentos, creo una TM para alimentos llamándola TMAlimentos



## EJEMPLO DE UNA TABLE MODEL

