Temas a tratar

-Colecciones

-excepciones

-API

-Maven

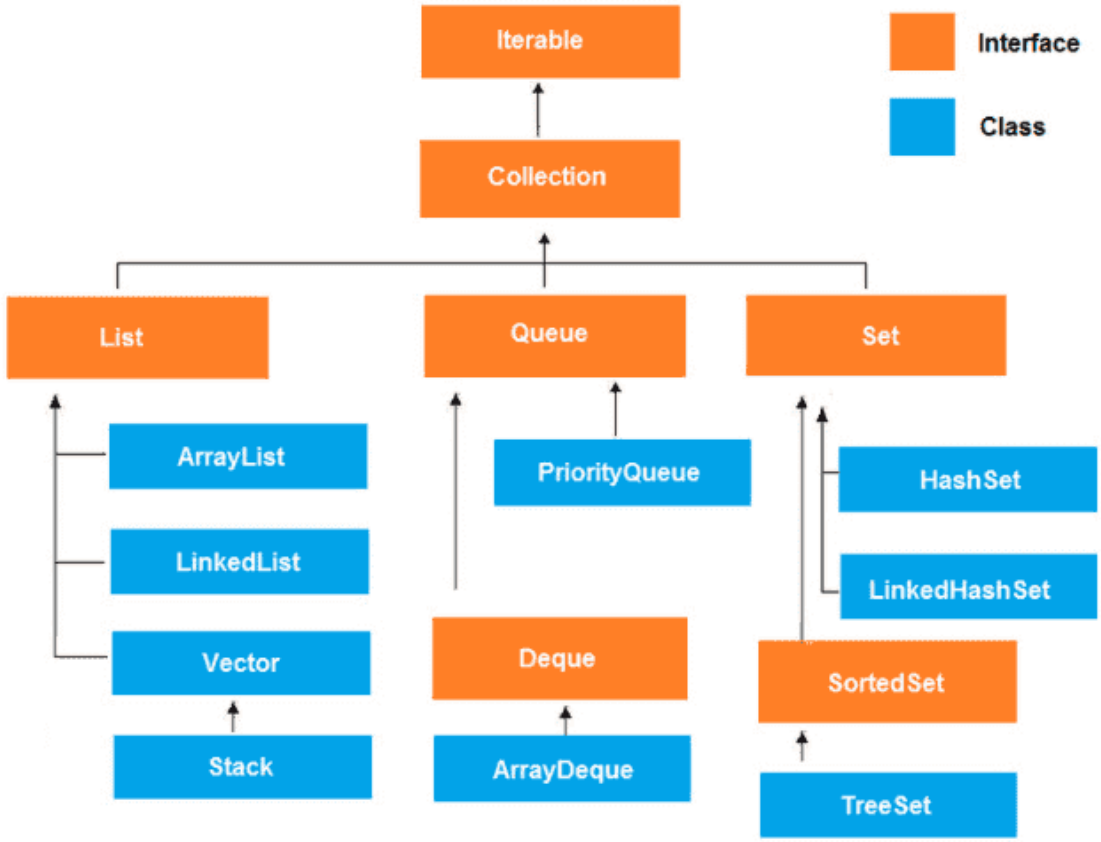
-JDBC

Colecciones en Java

En esta clase vamos a trabajar con las colecciones en Java. Vamos a ver qué son y los distintos tipos de colecciones más usados que existen (haremos foco en List). También, vamos a ver que cada uno de los distintos tipos de colecciones puede tener, además, distintas implementaciones, lo que ofrece funcionalidad distinta.

¿Qué son las colecciones?

Una colección representa un grupo de objetos. Esto objetos son conocidos como elementos. Cuando queremos trabajar con un conjunto de elementos, necesitamos un almacén donde poder guardarlos. En Java, se emplea la interfaz genérica Collection para este propósito. Gracias a esta interfaz, podemos almacenar cualquier tipo de objeto y podemos usar una serie de métodos comunes, como pueden ser: añadir, eliminar, obtener el tamaño de la colección… Partiendo de la interfaz genérica Collection extienden otra serie de interfaces genéricas. Estas subinterfaces aportan distintas funcionalidades sobre la interfaz anterior.



Set

La interfaz Set define una colección que no puede contener elementos duplicados. Esta interfaz contiene, únicamente, los métodos heredados de Collection añadiendo la restricción de que los elementos duplicados están prohibidos. Es importante destacar que, para comprobar si los elementos son elementos duplicados o no lo son, es necesario que dichos elementos tengan implementada, de forma correcta, los métodos equals y hashCode. Para comprobar si dos Set son iguales, se comprobarán si todos los elementos que los componen son iguales sin importar en el orden que ocupen dichos elementos.

Dentro de la interfaz Set existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

HashSet: esta implementación almacena los elementos en una tabla hash. Es la implementación con mejor rendimiento de todas, pero no garantiza ningún orden a la hora de realizar iteraciones. Es la implementación más empleada debido a su rendimiento y a que, generalmente, no nos importa el orden que ocupen los elementos. Esta implementación proporciona tiempos constantes en las operaciones básicas siempre y cuando la función hash disperse de forma correcta los elementos dentro de la tabla hash. Es importante definir el tamaño inicial de la tabla ya que este tamaño marcará el rendimiento de esta implementación.

TreeSet: esta implementación almacena los elementos ordenándolos en función de sus valores. Es bastante más lento que HashSet. Los elementos almacenados deben implementar la interfaz Comparable. Esta implementación garantiza, siempre, un rendimiento de log(N) en las operaciones básicas, debido a la estructura de árbol empleada para almacenar los elementos.

LinkedHashSet: esta implementación almacena los elementos en función del orden de inserción. Es, simplemente, un poco más costosa que HashSet.

Los tiempos obtenidos demuestran que, efectivamente, el tiempo de inserción es menor en HashSet y mayor en TreeSet. Es importante destacar que la inicialización del tamaño inicial del Set a la hora de su creación es importante ya que, en caso de insertar un gran número de elementos, podrían aumentar el número de colisiones y; con ello, el tiempo de inserción.

List

La interfaz List define una sucesión de elementos, la interfaz List admite elementos duplicados. A parte de los métodos heredados de Collection, añade métodos que permiten mejorar los siguientes puntos:

Acceso posicional a elementos: manipula elementos en función de su posición en la lista.

Búsqueda de elementos: busca un elemento concreto de la lista y devuelve su posición.

Iteración sobre elementos: mejora el Iterator por defecto.

Rango de operación: permite realizar ciertas operaciones sobre rangos de elementos dentro de la propia lista.

Dentro de la interfaz List existen varios tipos de implementaciones realizadas dentro de la plataforma Java. Vamos a analizar cada una de ellas:

ArrayList: esta es la implementación típica. Se basa en un array redimensionable que aumenta su tamaño según crece la colección de elementos. Es la que mejor rendimiento tiene sobre la mayoría de situaciones.

LinkedList: esta implementación permite que mejore el rendimiento en ciertas ocasiones. Esta implementación se basa en una lista doblemente enlazada de los elementos, teniendo cada uno de los elementos un puntero al anterior y al siguiente elemento.

Ninguna de estas implementaciones es sincronizada; es decir, no se garantiza el estado del List si dos o más hilos acceden de forma concurrente al mismo. Esto se puede solucionar empleando una serie de métodos que actúan de wrapper para dotar a estas colecciones de esta falta de sincronización:

El cuándo usar una implementación u otra de List variará en función de la situación en la que nos encontremos. Generalmente, ArrayList será la implementación que usemos en la mayoría de situaciones. Sobre todo, varían los tiempos de inserción, búsqueda y eliminación de elementos, siendo en unos casos una solución más óptima que la otra.

Como declarar una colección del tipo ArrayList: *En este ejemplo crearemos una colección de tipo List. En este caso un ArrayList, lo haremos de forma genérica. Es decir, no sabemos el tipo de dato que se almacenará en la colección.*

```
1
2  import java.util.ArrayList;
3
4  public class ArrayListJava {
5
6      public static void main(String[] args) {
7
8          ArrayList colecciones = new ArrayList();
9
10         colecciones.add("Carlos");
11         colecciones.add("Martin");
12         colecciones.add("Javier");
13         colecciones.add(10);
14         colecciones.add(10.5);
15
16     }
17
18 }
19
```

Para recorrer la lista podemos utilizar un ciclo for o un ciclo Foreach:

```
for(int i=0; i<colecciones.size();i++){

    System.out.println("Listado elementos: " + colecciones.get(i));
}
```

```
   for(Object elemento: colecciones){

       System.out.println("Listado elementos: " + elemento);

   }
```

También podemos crear un Arraylist sabiendo previamente el tipo de dato que se va a almacenar:

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

tipoString.add("Marco");
tipoString.add("Darma");
tipoString.add("Fio");
```

Métodos de listas:

add (int index, E element): inserta el elemento especificado en la posición especificada en la lista

get ((int index, E element): Devuelve el elemento en la posición especificada en esta lista

set (int index, E element): Elimina el elemento en la posición especificada en esta lista con el elemento especificado

indexOf(Object o): Devuelve el índice de la primera aparición del elemento especificado en esta lista, o -1 si esta lista no contiene el elemento.

Conclusiones

Como hemos visto a lo largo de esta lección Java proporciona una serie de estructuras muy variadas para almacenar datos. Estas estructuras, ofrecen diversas funcionalidades: ordenación de elementos, mejora de rendimiento, rango de operaciones… Es importante conocer cada una de ellas para saber cuál es la mejor situación para utilizarlas. Un buen uso de estas estructuras mejorará el rendimiento de nuestra aplicación.

Excepciones

En los sistemas pueden ocurrir eventos excepcionales que corten el flujo correcto y provoquen comportamientos inesperados, que quizá en clases anteriores las hemos visto con los Array.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 4 out of bounds for length 4
```

Una excepción no es más que un error que se presenta en nuestro software, estos errores pueden ser más o menos graves, se producen durante la ejecución del programa

En Java no se pueden evitar, pero se pueden gestionar y evitar la interrupción total de nuestro programa, tratando correctamente el problema y así tener un software que sea tolerante a fallas, más robusto.

Para comprender un poco esto tenemos que tener en cuenta que tenemos distintos tipos de errores:

Errores Tiempo de compilación

- Sintaxis erronea
- Asignacion incorrecta
- No seguir reglas

Errores en tiempo de ejecución.

- procesos no validos
- lógicos de tipo resultado incorrecto
- lógicos de tipo bucle infinito

Bloques try, catch y finally

Los siguientes bloques están conformados para capturar errores y ejecutar las instrucciones sin tener interrupciones en el sistema.

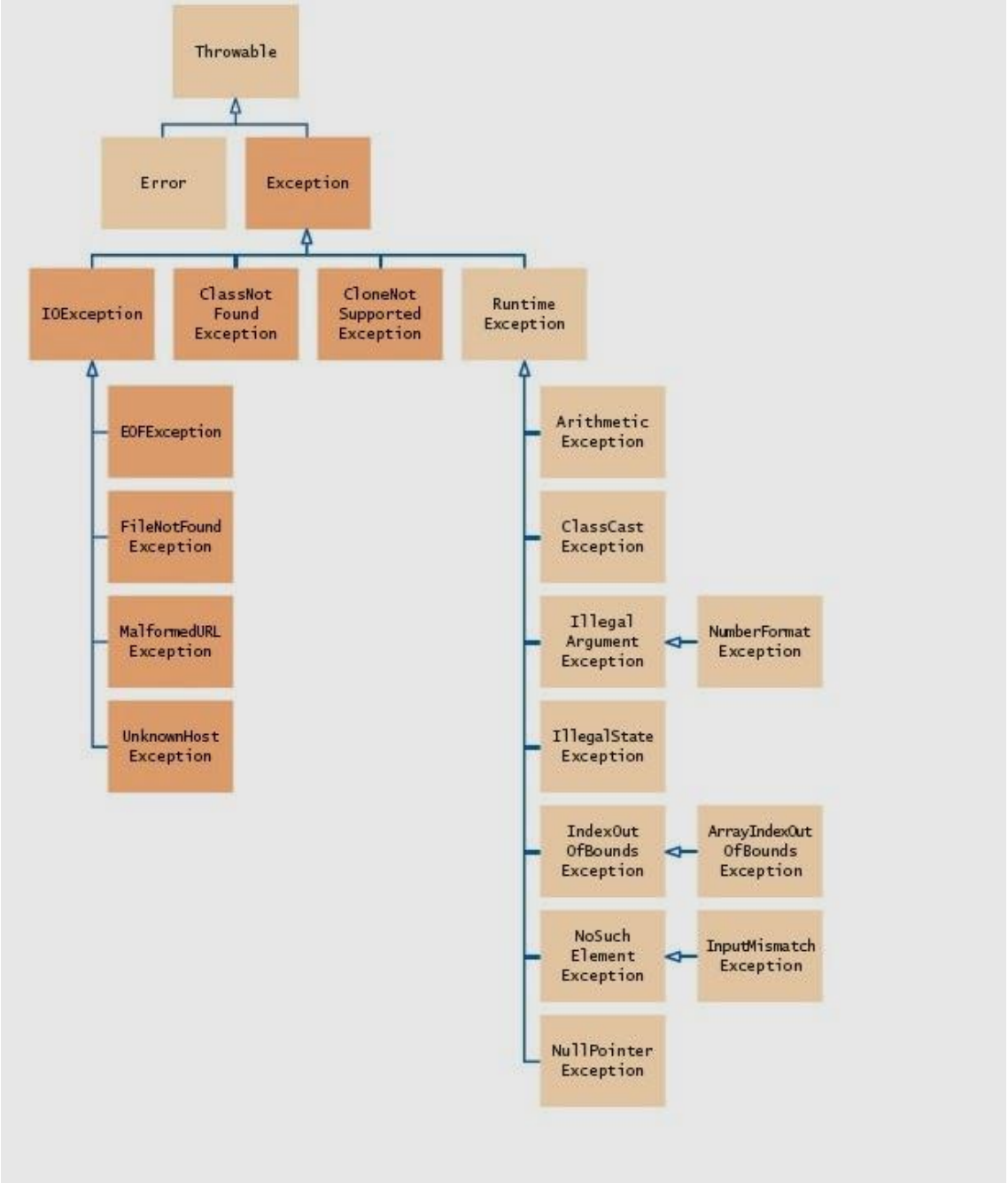
try: contendrá las instrucciones que pueden provocar el error. De este tipo de bloque sólo se puede crear uno por grupo.

catch: contendrá el código necesario para gestionar el error. Si se presenta un error en el bloque try el catch lo capturara creando un objeto según el tipo de excepción esperada, es por ello que este bloque contiene un parámetro y se pueden crear tantos bloques catch crea conveniente. No es obligatorio tener un catch pero es recomendable hacerlo.

finally: contendrá el código que se ejecutara suceda o no un error, este bloque no es obligatorio y de requerir sólo puede existir uno al final del try si no posee bloque catch y de poseerlo al final de todo el grupo.

Recordemos que el manejo de excepciones está diseñado para procesar errores sincrónicos, que ocurren cuando se ejecuta una instrucción. Ejemplos comunes de estos errores que

veremos en este libro son los índices fuera de rango, el desbordamiento aritmético (es decir, un valor fuera del rango representable de valores), la división entre cero, los parámetros inválidos de un método y la interrupción de hilos, así como la asignación fallida de memoria. El manejo de excepciones no está diseñado para procesar los problemas asociados con los eventos asíncronos (por ejemplo, completar las operaciones de E/S de disco, la llegada de mensajes de red, clics del ratón y pulsaciones de teclas), los cuales ocurren en paralelo con e independientemente del flujo de control del programa.



Todas las excepciones son un subtipo de la clase java.lang.Exception, la cual a su vez es una subclase de la clase Throwable. Los errores son situaciones anormales de nuestro programa y como no pueden ser manejados por el mismo para notificarlos debemos hacerlo desde la clase Throwable, en cambio de la clase Exception tendremos dos subclases encargadas de manejar

las excepciones chequeadas (IOException) y las excepciones no chequeadas (RuntimeException), pasemos a ver algunos métodos disponibles en la clase Throwable:

public String getMessage(), nos devuelve una información detallada de la excepcion

public Throwable getCause(), devuelve la causa de la excepcion por medio de un objeto Throwable

public String toString(), devuelve el nombre de la clase concatenada con el resultado de getMessage()

public void printStackTrace(), imprime el resultado de toString() junto con el seguimiento de la pila a System.err

public StackTraceElement[] getStackTrace(), devuelve un array conteniendo los elementos del seguimiento de la pila, donde el indice cero sera el elemento de arriba de la pila y el ultimo indice el del ultimo elemento de la pila

public Throwable fillInStackTrace(), Rellena el seguimiento de pila de este objeto Throwable con el seguimiento de pila actual, agregando a cualquier información anterior en el seguimiento de pila.

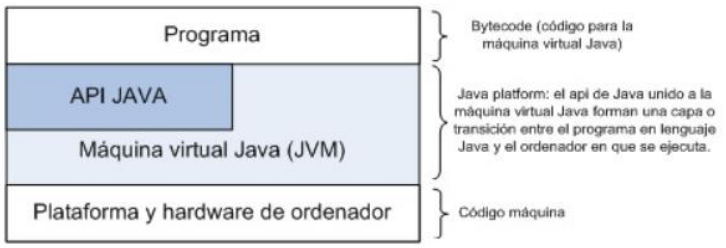
¿QUÉ ES Y PARA QUÉ SIRVE EL API DE JAVA?

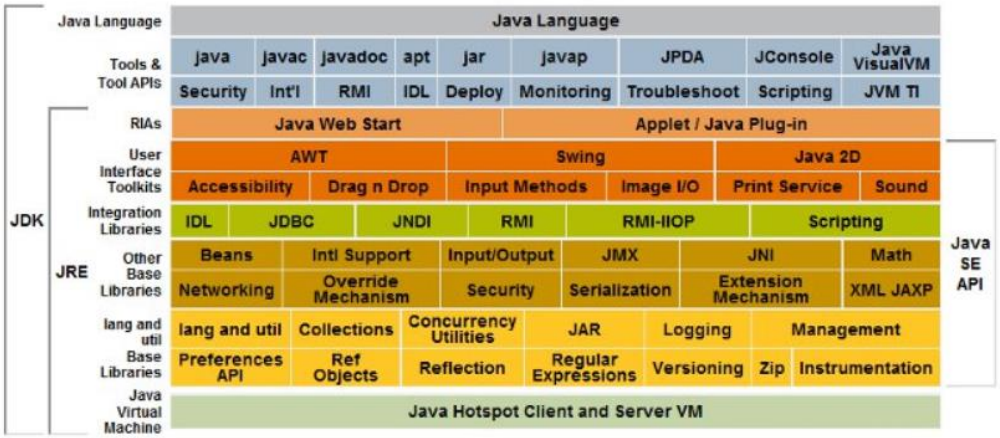
Durante las clases hemos visto ejemplos donde utilizábamos la clase System (por ejemplo en la invocación System.out.println) o la clase String (que es la clase gracias a la que podemos usar los objetos String). ¿De dónde salen estas clases si nosotros no las hemos programado como la clase Auto o la clase Persona?

La respuesta está en que al instalar Java (el paquete JDK) en nuestra computadora, además del compilador y la máquina virtual de Java se instalan bastantes muchos más elementos. Entre ellos, una cantidad muy importante de clases que ofrece la multinacional desarrolladora de Java y que están a disposición de todos los programadores listas para ser usadas. Estas clases junto a otros elementos forman lo que se denomina API (Application Programming Interface) de Java.

La mayoría de los lenguajes orientados a objetos ofrecen a los programadores bibliotecas de clases que facilitan el trabajo con el lenguaje.

Los siguientes esquemas, parte de la documentación de Java, nos dan una idea de cómo funciona el sistema Java y de qué se instala cuando instalamos Java (el paquete JDK) en nuestro ordenador.





En este esquema lo único que queremos mostrar es que cuando instalamos Java en nuestro ordenador instalamos múltiples herramientas, entre ellas una serie de "librerías" (paquetes) a cuyo conjunto solemos referirnos como "biblioteca estándar de Java". Las librerías contienen código Java listo para ser usado por nosotros. Ese es el motivo de que podamos usar clases como System o String sin necesidad de programarlas.

No podemos acceder al código de estas librerías, la biblioteca estándar de Java se facilita como código cerrado, es decir, como código máquina. No podemos acceder al código fuente. Esto tiene su lógica porque si cada persona accediera al código fuente de los elementos esenciales de Java y los modificara, no habría compatibilidad ni homogeneidad en la forma de escribir programas Java.

Aunque no podamos acceder al código fuente, sí podemos crear objetos de los tipos definidos en la librería e invocar sus métodos. Para ello lo único que hemos de hacer es conocer las clases y la signatura de los métodos, y esto lo tenemos disponible en la documentación de Java accesible para todos a través de internet.

Java: ¿Qué es Maven?

Apache Maven es una potente herramienta de gestión de proyectos que se utiliza para gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación. Es de código abierto y gratuita.

Aunque se puede utilizar con diversos lenguajes (C#, Ruby, etc) se usa principalmente en proyectos Java, donde es muy común ya que está escrita en este lenguaje. De hecho, frameworks Java como Spring y Spring Boot la utilizan por defecto, por lo que conviene conocerla si trabajas en la plataforma Java y, desde luego, con Spring.

Al contrario que otras herramientas anteriores y más limitadas como Apache Ant (también muy popular), Maven utiliza convenciones sobre dónde colocar ciertos archivos para el proceso de build de un proyecto, por lo que solo debemos establecer las excepciones y por lo tanto simplifica mucho el trabajo. Además, es una herramienta declarativa. Es decir, todo lo que definamos (dependencias en módulos y compontes externos, procesos, orden de compilación, plugins del propio Maven...) se almacena en un archivo XML que Maven lee a la hora de funcionar. Otras alternativas, como Gradle no utilizan archivos XML, sino de otro tipo, pero usan los mismos conceptos de Maven.

Con Maven se puede:

- Gestionar las dependencias del proyecto, para descargar e instalar módulos, paquetes y herramientas que sean necesarios para el mismo.
- Compilar el código fuente de la aplicación de manera automática.
- Empaquetar el código en archivos .jar o .zip.
- Instalar los paquetes en un repositorio (local, remoto o en el central de la empresa)
- Generar documentación a partir del código fuente.
- Gestionar las distintas fases del ciclo de vida de las build: validación, generación de código fuente, procesamiento, generación de recursos, compilación, ejecución de test.

Además, la mayor parte de los entornos de desarrollo y editores de Java disponen de plugins específicos o soporte directo de Maven para facilitarnos el trabajo con esta, puesto que se ha convertido en una herramienta casi universal.

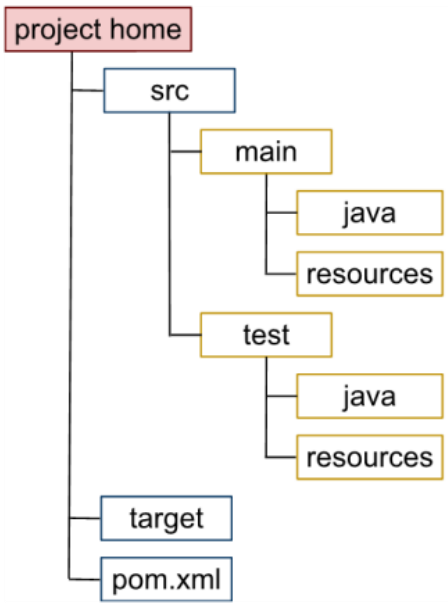
¿Qué es el archivo pom.xml?

La unidad básica de trabajo en Maven es el llamado Modelo de Objetos de Proyecto conocido simplemente como POM (de sus siglas en inglés: Project Object Model).

Se trata de un archivo XML llamado pom.xml que se encuentra por defecto en la raíz de los proyectos y que contiene toda la información del proyecto: su configuración, sus dependencias, etc...

Incluso, aunque nuestro proyecto, que usa Maven, tenga un archivo pom.xml sin opciones propias, prácticamente vacío, estará usando el modelo de objetos para definir los valores por defecto del mismo. Por ejemplo, por defecto, el directorio donde está el código fuente es src/main/java, donde se compila el proyecto es target y donde ubicamos los test unitarios es en src/main/test, etc... Al pom.xml global, con los valores predeterminados se le llama Súper POM.

Esta sería la estructura habitual de un proyecto Java que utiliza Maven:



Este es el contenido de un archivo pom.xml sencillo de ejemplo:

```
<project>
 <modelVersion>4.0.0</modelVersion>
 <groupId>es.campusmvp</groupId>d>
 <artifactId>demo</artifactId>
 <version>1.0</version>
 <!-- Dependencias -->
 <dependencies>
  <dependency>
                <groupId>org.springframework.boot</groupId>
```

```
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
  <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
            <exclusions>
                <exclusion>
                    <groupId>org.junit.vintage</groupId>
                    <artifactId>junit-vintage-engine</artifactId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>org.apache.tomcat.embed</groupId>
            <artifactId>tomcat-embed-jasper</artifactId>
            <scope>compile</scope>
        </dependency>
  </dependencies>
</project>
```

Como puedes ver incluye una serie de dependencias: la primera es el starter de Spring Boot para generar una aplicación Web, la segunda es el starter que trae todas las dependencias para poder ejecutar test (por ejemplo JUnit o Mockito) y que excluye todo lo relativo al motor antiguo de JUnit 4. La final añade el soporte para JSP en el servidor Apache Tomcat.

4 primeros nodos son los únicos obligatorios y definen respectivamente el modelo de objetos que utilizará Maven, el identificador del grupo del proyecto, el id del proyecto (artifact) y su versión. Esto es lo mínimo que debe incluir el archivo si utilizamos Maven y la mayor parte de los editores y herramientas nos lo crearán automáticamente al crear el proyecto.

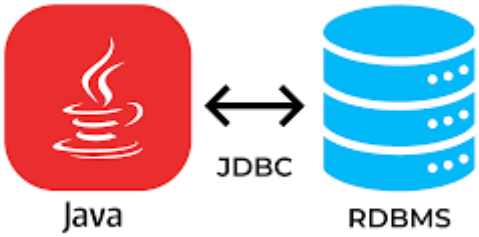
Herencia de POMs

Dentro de un proyecto pueden existir varios archivos pom.xml en distintas subcarpetas. Cuando una subcarpeta tiene su propio POM, este hereda los valores de las carpetas superiores, sobrescribiéndolos en caso de estar definidos de nuevo dentro de él.

De hecho el archivo pom.xml que tenemos en la raíz está actuando ya de esta manera: hereda todos los valores del Súper POM que son los que tenemos por defecto y, si en nuestro pom.xml propio en el proyecto establecemos otros diferentes, estos prevalecerán sobre los que hay en el Súper POM.

Este tipo de funcionamiento es típico en muchas herramientas de configuración y nos ofrece una manera muy potente de definir valores particulares para casos concretos teniendo una configuración global por defecto.

JDBC



Java Database Connectivity (JDBC) Es una API de acceso a bases de datos estándar SQL que proporciona un acceso uniforme a una gran variedad de bases de datos relacionales.

JDBC también proporciona una base común para la construcción de herramientas y utilidades de alto nivel. El paquete actual de JDK incluye JDBC.

La API de JDBC se compone de dos paquetes:

• **java.sql:** Paquete para acceder y procesar datos almacenados en una fuente de

datos (generalmente una base de datos relacional).

• **javax.sql:** Paquete para acceso y procesamiento de fuentes de datos del lado

del servidor.

Consiste en un conjunto de clases e interfaces escritas en el lenguaje de programación Java. JDBC suministra un API estándar para los desarrolladores y hace posible escribir aplicaciones de base de datos usando un API puro Java.

Librerías adicionales

Java Standard WEB Programming, J2SE Al trabajar con JDBC resulta necesario agregar un jar al proyecto que contiene las clases necesarias que se utilizan para “dialogar” con un DBMS. Cada DBMS tiene su propio archivo jar.

Recordemos que nuestros proyectos están creados con un arquetipo de Maven y a través del archivo pom.xml podemos gestionar las librerías que necesitamos.

Nuestro proyecto trabajara con MySQL así que nos dirigimos al repositorio de Maven y buscamos el Driver de Mysql:

Agregamos la siguiente dependencia en el archivo pom.xml

```
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.19</version>
        </dependency>
    </dependencies>
```

Connection

Esta interfaz provee una sesión de trabajo con una base de datos. Sus métodos, aparte de permitir modificarla y consultar sus tablas, también permiten obtener información sobre su estructura.

Metodos:

commit() Confirma los cambios realizados provisionalmente por las transacciones.

close() Libera la conexión a la base de datos.

createStatement() Crea un objeto Statement para enviar sentencias SQL a la base de datos.

prepareStatement (String sql) Crea un objeto PreparedStatemento para enviar sentencias SQL parametrizadas a la base de datos.

rollback() Deshace todos los cambios realizados en la transacción actual y libera los bloqueos de la base de datos.

setAutoCommit (boolean autoCommit) Establece el modo de confirmación automática.

isClosed() Recupera si la conexión se ha cerrado.

Crear la conexión

Para registrar el driver de la librería descargada lo primero que debemos hacer es indicar la clase que nos provee dicha librería.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

Una vez registrado el driver debemos establecer la conexión a través de un método estático que nos provee la clase

DriverManager getConnection(String url, String user, String password) y nos retorna un objeto de tipo Connection.

```
Connection conexion = DriverManager.getConnection(url,"root","");
```

• La URL espera el driver, el nombre de la PC o su IP, el Puerto de conexión a la base de datos (por defecto en MySQL 3306) y el nombre de la base de datos.

• El usuario espera el nombre del usuario que se conecta a la base de datos, por defecto en MySQL es root.

• La Clave espera por defecto la contraseña que posee el usuario, dependiendo del tipo de instalación puede estar vacía o una clave colocada al momento de instalar.

Importante: El parámetro url debe tener la siguiente característica, ya que por temas de seguridad hay que especificar algunos valores:

url="jdbc:mysql://localhost:3306/desafiobd2022?useSSL=false&useTimezone=true&serverTimezone=UTC&allowPublicKeyRetrieval=true";

** desafiobd2022 es el nombre de la base de datos que utilizo, ustedes deberán colocar el nombre de la base de datos a la cual se conectan.*

Ejecutar instrucciones SQL

La interfaz Statement nos provee una serie de métodos que al ser ejecutados nos devuelven los resultados que producen las sentencias ejecutadas.

Al crear la conexión el objeto nos provee como pudimos ver en el cuadro de métodos de Connection el método createStatement() que nos devuelve un Statement.

Métodos

execute (String sql) Ejecuta la instrucción SQL dada, que puede devolver varios resultados.

executeQuery (String sql) Ejecuta la instrucción SQL dada, que devuelve un solo Objeto ResultSet.

El método execute() se utiliza para ejecutar sentencias SQL del tipo INSERT, UPDATE o DELETE.

ResultSet

Para recuperar la información de una tabla o un origen de datos Java nos proporciona la interfaz ResultSet. Esta interfaz nos provee de varios métodos para obtener los datos de columna correspondientes a un fila. Todos ellos tienen el formato get<Tipo>, siendo <Tipo> un tipo de datos Java. Algunos ejemplos de estos métodos son getInt, getLong, getString, getBoolean y etc.

Casi todos estos métodos toman un solo parámetro, que es el índice que la columna tiene dentro del ResultSet o bien el nombre de la columna. El método executeQuery(String sql) de la interfaz Statement nos devuelve dicho objeto. Para poder recuperar cada una de las filas del objeto si las hubiera utilizaremos por lo común el bucle while colocando como condición el método next() que devuelve un boolean indicando si existe una fila que leer, si existe dicha fila mueve el cursor una fila hacia adelante desde su posición actual.

Dejamos el bloque de código en el cual toma los registros de una tabla:

```java
try {
    Class.forName("com.mysql.cj.jdbc.Driver");
} catch (ClassNotFoundException ex) {
    Logger.getLogger(TestBD.class.getName()).log(Level.SEVERE, null, ex);
}

try {

    Connection conexion = DriverManager.getConnection(url,"root","");
    Statement instruccion =  conexion.createStatement();
    var sql = "SELECT id_empleados FROM empleados";
    ResultSet resultado = instruccion.executeQuery(sql);

    while(resultado.next()){

        System.out.println("Id de usuario: " +resultado.getInt("id_empleados"));
    }

} catch (SQLException ex) {
    Logger.getLogger(TestBD.class.getName()).log(Level.SEVERE, null, ex);
}
```

A continuación, les dejo un enlace que tiene los comandos basicos para hacer la conexión:

https://github.com/alejandrozapata73/crearConexionJavaBDJDBC