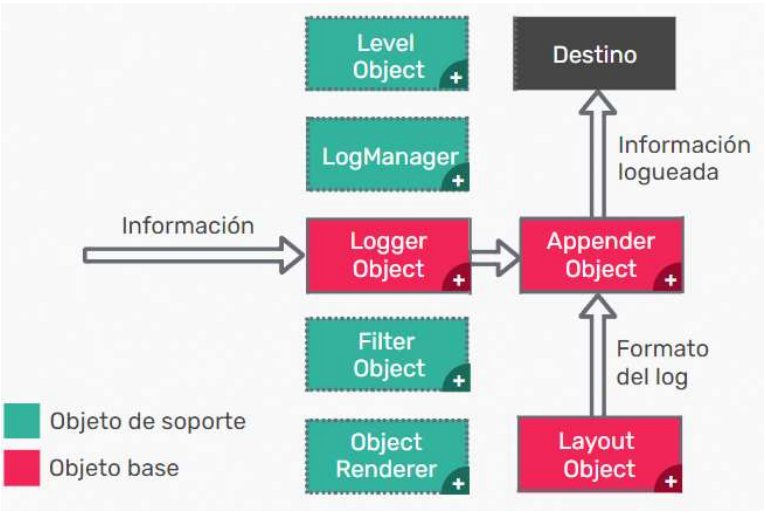
BackEndI
Profesor: Christian Ariel Díaz

Módulo 2. Logging y acceso a datos

Sesión 10. Logging (trazas y debug)
Mayo 25 de 2022

¿Qué es Log4j?
Log4j es una librería desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores elegir la salida y el nivel de granularidad de los mensajes o logs en tiempo de ejecución. En otras palabras, es utilizada para generar mensajes de logging de una forma limpia, sencilla, permitiendo filtrarlos por importancia y pudiendo configurar su salida tanto por consola, fichero u otras diferentes.

Arquitectura de Log4J



la API de Log4J sigue una estructura en capas donde cada una proporciona diferentes objetos para realizar diferentes tareas. Esta estructura hace que el diseño sea flexible y fácil de ampliar en el futuro. Los dos tipos de objetos disponibles con el marco Log4J son:

- Objetos base:
Objetos de marco obligatorios y necesarios para utilizar el marco. Losobjetos base incluyen los tipos Logger object y Appender object.

- Objetos de soporte:
Objetos de marco opcionales que soportan objetos básicos para realizar tareas adicionales.

Logger Object:

La capa de nivel superior es el Logger que proporciona el objeto Logger. Este es responsable de capturar la información de registro y se almacena en una jerarquía de espacio de nombres.

Level Object:
El objeto de nivel define la granularidad y prioridad de cualquier información de registro. Hay siete niveles de registro definidos en la API: OFF, DEBUG, INFO, ERROR, WARN, FATAL y ALL.

Log Manager:
Administra el marco de registro. Es responsable de leer los parámetros de configuración inicial de una configuración de todo el sistema, un archivo de configuración o una class de configuración.

Filter Object:
El objeto de filtro se utiliza para analizar la información de registro y tomar otras decisiones sobre si esa información debe registrarse o no.

Object Renderer:
Se especializa en proporcionar una representación de cadena de varios objetos pasados a la infraestructura de registro. Los objetos layout utilizan este objeto para preparar la información de registro final.

Appender Object:
Esta es una capa de nivel inferior que proporciona el objeto appender. Este es responsable de publicar información de registro.

Layout Object:
La capa de diseño proporciona objetos que se utilizan para formatear la información del registro en diferentes estilos. Los objetos de diseño juegan un papel importante en la publicación de información de registro de una manera que sea legible.

Niveles de registro

OFF:
Este es el nivel de mínimo detalle, deshabilita todos los logs.

FATAL:
Se utiliza para mensajes críticos del sistema, generalmente después de guardar el mensaje, el programa se cierra.

ERROR:
Indica eventos de error que aún podrían permitir que la aplicación continúe ejecutándose.

WARN:
Se utiliza para mensajes de alerta sobre eventos.

INFO:
Se refiere a mensajes informativos que resaltan el progreso de la aplicación en un nivel aproximado.

DEBUG:
Designa los eventos informativos detallados más útiles para depurar una aplicación.

TRACE:
Se utiliza para mostrar mensajes con un mayor nivel de detalle que debug.

ALL:
Es el nivel de máximo detalle, habilita todos los logs.

Logging:

Falta

Ejemplo

```
import org.apache.log4j.Logger;

public class TestLog {
	private static final Logger logger = Logger.getLogger(TestLog.class);
	public static void main(String[] args) {
		logger.info("Empezamos nuestro metodo MAIN");
		try {
			String variable = "Hola";
			int division = 1 / 0;
		} catch (Exception e) {
			logger.error("Error por dividir por cero ", e);
		}
		logger.warn("Advertencia el metodo MAIN esta por finalizar");
		logger.debug("Esto va a mostrarse solo si el infoLogger esta en
		DEBUG");
		logger.info("Finalizamos el thread MAIN");
	}
}
```

Ejercicio
Resolver la siguiente actividad:
Crear una aplicación con dos clases: León y Tigre; cada una de las cuales debe tener dos atributos:

nombre String
edad int

y para la clase León agregar el atributo: esAlfa boolean.

Para los dos animales crear un método correr, que loguee un info de que está corriendo y otro método que calcule si es mayor a 10 años y es alfa, en caso de serlo, deberá loguear un info con la información.

También arrojar una Exception si la edad del animal es menor a cero y agregar un log de error. Crear una clase main, donde se creen leones y tigres que cumplan que al correr y esMayorA10 ejecutan los métodos:

public void correr()
public void esMayorA10()

También se debe chequear que los logs existan.

La salida debe ser algo como:

```
[2021-07-18 18:27:46] [ INFO ] [Leon:37] El León Simba está corriendo
[2021-07-18 18:27:46] [ INFO ] [Leon:37] El León Bom está corriendo
[2021-07-18 18:27:46] [ ERROR] [Leon:45] La edad no puede ser negativa
[2021-07-18 18:27:46] [ ERROR] [Test:30] La edad del León Bom es
incorrecta
java.lang.Exception
        at com.main.Leon.esMayorA10(Leon.java:46)
        at com.main.Test.main(Test.java:28)
[2021-07-18 18:27:46] [ INFO ] [Tigre:28] El Tigre está corriendo
[2021-07-18 18:27:46] [ INFO ] [Tigre:28] El Tigre está corriendo
```

Sesión 11. Acceso a base de datos
Mayo 26 de 2022

JDBC
JDBC, por las siglas de Java DataBase Conectivity, es un framework que consiste en múltiples interfaces y solo algunas clases de soporte. Esto se debe a que la idea detrás de JDBC es que cualquiera pueda crear u propia implementación del framework y adaptarla a sus necesidades. Dado que se trata de un conjunto de interfaces, cualquier código que interactúe con el framework no se verá afectado si se altera la implementación.

Es por esto que JDBC define interfaces que solo declaran el comportamiento que debe llevarse a cabo para conectarse e interactuar con una base de datos. Así, existen las interfaces Connection (abstracción del comportamiento de una conexión), Statement (define el comportamiento para realizar sentencias contra una base de datos, como queries u otras instrucciones), ResultSet (que abstrae el comportamiento para extraer resultados de las consultas), entre otras.

Drivers
Cada implementación de jdbc consiste en un paquete de clases, típicamente empaquetados en archivos .jar. Se pueden descargar de los sitios oficiales de cada motor de base de datos y son conocidos como driver jdbc.

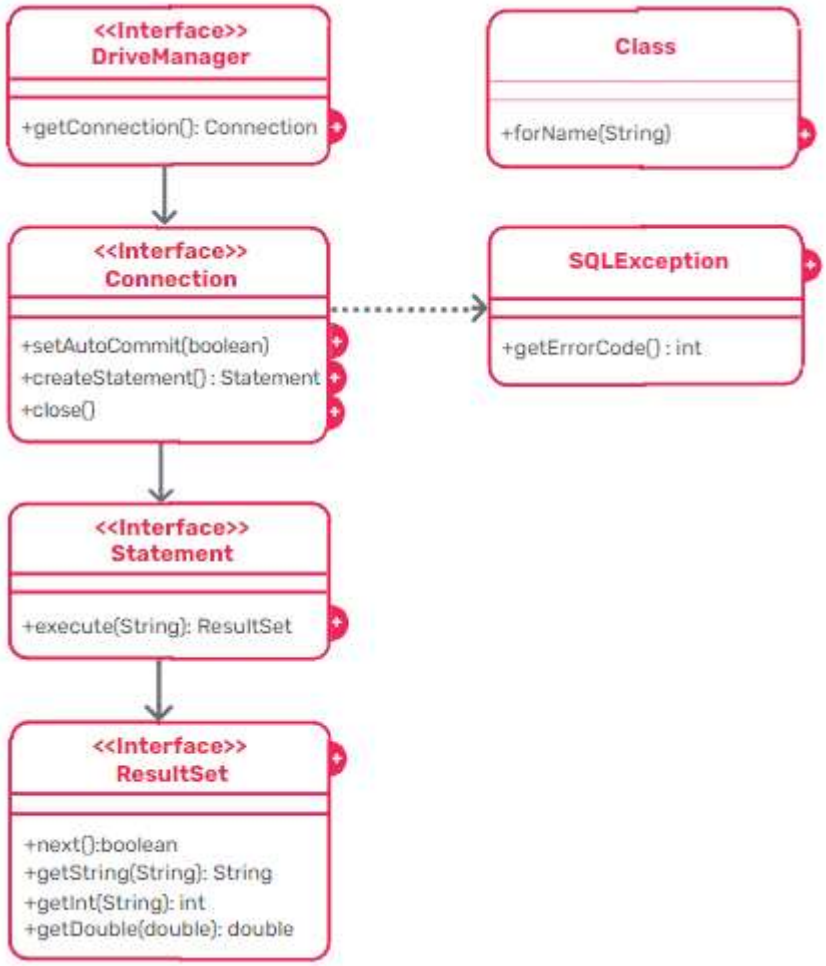
Uso:
1. Cargar el driver.
2. Obtener la conexión.
3. Ejecutar la consulta contra la base.
4. Cerrar la conexión.

Ejemplo

```
Connection c = null;
try {
	Class.forName("org.h2.Driver");
} catch (ClassNotFoundException e) {
	e.printStackTrace();
	System.exit(0);
}

try {
	String url = "jdbc:h2:tcp://localhost//D:/base_de_datos/ejemplo";
	c = DriverManager.getConnection(url, "sa", "sa");
	c.setAutoCommit(false);
} catch (SQLException e) {
	e.printStackTrace();
	System.exit(0);
}
```

```
try {
	Statement s = c.createStatement();
	s.execute("AQUÍ_SENTENCIA_SQL");
} catch (Exception e) {
	e.printStackTrace();
} finally {
	try {
		c.close();
	} catch(SQLException e) {
		e.printStackTrace();
	}
}
```



Interface DriveManager
Con la clase disponible, se hace uso de la clase DriverManager que es el registro de los drivers jdbc que se tienen configurados y se obtiene una conexión a la base de datos.

DriverManager.getConnection necesita tres parámetros:

- URL de conexión
- Usuario.
- Password.

Class
forName
Con este método se carga o ‘’registra’’ la clase principal del driver. Esto hace disponible al DriverManager para poder administrar los drivers jdbc e ir instanciando las clases —que implementan las interfaces— del driver. Como lo que se estáhaciendo es buscar una clase por su nombre, mediante un parámetro String, el compilador puede tomar esa sintaxis como válida, pero al momento de la ejecución, puede que la clase no se encuentre en el CLASSPATH. Por eso, este método arroja una ClassNotFoundException que debe manejarse adecuadamente.

Interface Connection

setAutoCommit(boolean)
Con este método se indica si esta conexión debe manejar las transacciones automáticamente (true) o manualmente (false).

createStatement():
Con este método, se prepara el camino para ejecutar una sentencia contra la base de datos. Aquí nuevamente, se obtiene una instancia de una clase que implementa la interfaz Statement.

close()
Para cerrar la conexión. Siempre que se usa algún recurso, se debe liberar y, es uno de los usos típicos del bloque finally en un esquema try-catch-finally.

Interface Statement

execute(String): ResultSet
Lo que se hace es ejecutar la sentencia contra la base de datos. Toma como argumento un String con la sentencia SQL propiamente dicha. Existen diferentes maneras de ejecutar sentencias:

- boolean execute(String): se utiliza para ejecutar cualquier sentencia a la base de datos.

- int executeUpdate(String): se utiliza para ejecutar sentencias DML, o sea, sentencias que manipulen datos (insert, update, delete). Devuelve un entero con la cantidad de filas afectadas por la sentencia.

- ResultSet executeQuery(String): se utiliza para ejecutar consultas a base de datos.

Interface ResultSet

Se utiliza para obtener resultados de una consulta a la base de datos. Para recorrer los resultados ResultSet tiene una serie de operaciones fundamentales:

- next(): devuelve true o false, indicando si hay una tupla siguiente para analizar o no. Si la hay, avanza el puntero una posición.

- getXYZ(nombreCampo: String):son métodos para obtener los valores de cada campo, dependiendo de su tipo. Ej: getInt(), getString(), getDate().

H2 Database

¿Qué es H2?

H2 es una base de datos open source escrita en Java que permite integrar aplicaciones en Java o ejecutarse en modo cliente-servidor. Principalmente, se puede configurar para que se ejecute como una base de datos en memoria. Entonces, los datos no persistirán en el disco, debido a que la base de datos no se utiliza para el desarrollo de producción, sino principalmente para el desarrollo y las pruebas.

Características:

- Alta integración
- Multiplataforma
- Rápida
- Tamaño reducido
- Modo embebido
- Modo "en memoria"

Ejercicio

Crear una aplicación para guardar en una base de datos la clase Empleado. Agregar cuatro atributos a esta clase: Nombre, edad, empresa y fecha que empezó a trabajar. No olvidar que el ID es obligatorio.

Pasos:

- Crear la conexión a la base de datos.
- Crear la tabla mediante un Statement.
- Insertar 3 filas con nombre de la empresa: Digital, Google y Facebook.
- Mostrar los datos en un ResultSet por System.out.println.

Sesión 12. Integradora 4
Mayo 27 de 2022

Sesión 13. Consultas y transacciones sobre base de datos
Mayo 30 de 2022

Statement vs PreparedStatement

Statement

En este caso se construye una sentencia y se aporta un parámetro a la consulta de forma dinámica. Esto básicamente se convierte en una consulta SQL que se ejecuta vía el driver JDBC contra la base de datos.

Ejemplo

```
Connection conexion = DriverManager.getConnection("jdbc:h2:" +
"./Database/my", "root", "myPassword");
Statement sentencia = conexion.createStatement();
String nombre="pepe";
String consulta = "select * from Persona where nombre='"+nombre+"'";
ResultSet rs=sentencia.executeQuery(consulta);
```

PreparedStatement

A dos consultas diferentes, se crean dos planes de ejecución diferentes aunque ambas consultas sean realmente muy similares y únicamente entre en juego el valor del parámetro que se les pasa. Para solventar este problema existen, los JDBC Prepared statement. Estas estructuras permiten mantener las consultas neutras sin tener en cuenta los parámetros que se las pasa, ya que realizan un binding de ellos.
De esta forma, cuando la base de datos genera un hash para el plan de ejecución, ambas consultas devuelven el mismo hash y comparten el plan de ejecución.

Ejemplo:

```
String consulta = "select * from Persona where nombre = ? ";
Connection conexion= DriverManager.getConnection("jdbc:h2:" +
"./Database/my", "root", "myPassword");
PreparedStatement sentencia= conexion.prepareStatement(consulta);
sentencia.setString(1, "pepe");
ResultSet rs = sentencia.executeQuery();
```

El uso de PreparedStatement no solo ahorra la construcción de planes de ejecución, sino que también evita la inyecciones SQL, ya que al parametrizar la consulta, la API de JDBC protege contra este tipo de ataques. Normalmente el uso de consultas parametrizadas mejora el rendimiento entre un 20 y un 30 % a nivel de base de datos.

Modificar datos con PreparedStatement

Update

```
String query = ("UPDATE PERSONAS SET NOMBRE=? WHERE APELLIDO=?”);
try (PreparedStatement pstmt = con.prepareStatement(query)) {
	pstmt.setString(1, Mariano");
	pstmt.setString(2, "Martinez");
	pstmt.executeUpdate();
}
catch (SQLException ex) {

}
```

Insert

```
String query = ("INSERT INTO PERSONAS (NOMBRE, APELLIDO) VALUES
(?,?));
try (PreparedStatement pstmt = con.prepareStatement(query)) {
	pstmt.setString(1, Mariano");
	pstmt.setString(2, "Martinez");
	pstmt.executeUpdate();
}
catch (SQLException ex) {

}
```

Delete

```
String query = ("DELETE FROM PERSONAS WHERE APELLIDO=?);
try (PreparedStatement pstmt = con.prepareStatement(query)) {
	pstmt.setString(1, Mariano");
	pstmt.executeUpdate();
}
catch (SQLException ex) {

}
```

Transacciones

Una transacción es un conjunto de operaciones sobre una base de datos que se deben ejecutar como una unidad.

Un objeto Connection por defecto realiza automáticamente cada operación sobre la base de datos. Esto significa que cada vez que se ejecuta una instrucción, se refleja en la base de datos y no puede ser deshecha. Por defecto está habilitado el modo auto-commit en la conexión.

La interfaz Connection proporciona los siguientes métodos para gestionar las transacciones:

void setAutoCommit(boolean valor)
void commit()
void rollback()

- Para iniciar una transacción se deshabilita el modo auto-commit mediante el método setAutoCommit(false). Esto permite tener control sobre lo que se realiza y cuándo se realiza.
- Una llamada al método commit() realizará todas las instrucciones emitidas desde la última vez que se invocó el método commit().
- Una llamada a rollback() deshace todos los cambios realizados desde el último commit(). Una vez se ha emitido una instrucción commit(), esas transacciones no pueden deshacerse con rollback().

Pasos para invocar Stored Procedures

1. Las llamadas a los procedimientos almacenados al igual que las PreparedStatements y las consultas simples se hacen sobre la conexión, en este caso, con el método prepareCall() que nos devuelve un CallableStatement.

2. La llamada al procedimiento almacenado, además de ir entre comillas por ser un string, tiene que ir también entre llaves y con el siguiente formato: «{call nombre_procedimiento(?,?,…)}» —con tantos signos de interrogación como parámetros tenga el procedimiento almacenado—.

3. Los parámetros de entrada, como con las PreparedStatements, se definen con los métodos setXXX().

4. Hay que definir el tipo de los parámetros de salida con registerOutParameter() donde debemos indicar el tipo del que será ese dato en la base de datos y no en Java.

5. El procedimiento se ejecuta cuando se llama al método execute; y, como es lógico, en el momento en el que se ejecute tienen que estar definidos todos los parámetros tanto de entrada como de salida.

6. Finalmente, una vez ejecutado el procedimiento almacenado, se pueden obtener los valores que devuelve usando el método getXXX() adecuado para cada caso, recordando que ahora se están obteniendo los valores en Java, por lo que para obtener un varchar se usa getString() y no getVarchar().

Ejemplo

Aplicación

```
public static void main(String[] args) {

	BufferedReader entrada = new BufferedReader(new
InputStreamReader(System.in));
	int id = -1;
	Connection cn = null;

	try {
	   // Carga el driver
	   Connection cn = DriverManager.getConnection("jdbc:h2:" +
"./Database/my", "root", "myPassword");

	   // Llamada al procedimiento almacenado
	   CallableStatement cst = cn.prepareCall("{call ObtenerDatosAlumno
(?,?,?,?)}");

	   do {
	      System.out.println("\nIntroduce el ID del alumno:");
	      try {
	         id = Integer.parseInt(entrada.readLine());
	      } catch (IOException ex) {
	         System.out.println("Error...");
	      }

	      // Parametro 1 del procedimiento almacenado
	      cst.setInt(1, id);

	      // Definimos los tipos de los parametros de salida del procedimiento
almacenado
	      cst.registerOutParameter(2, java.sql.Types.VARCHAR);
	      cst.registerOutParameter(3, java.sql.Types.VARCHAR);
	      cst.registerOutParameter(4, java.sql.Types.VARCHAR);

	      // Ejecuta el procedimiento almacenado
	      cst.execute();

	      // Se obtienen la salida del procedimineto almacenado
	      String nombre = cst.getString(2);
	      String sexo = cst.getString(3);
	      String curso = cst.getString(4);
	      System.out.println("Nombre: " + nombre);
	      System.out.println("Sexo: " + sexo);
	      System.out.println("Curso: " + curso);

	   } while (id > 0);
```

```
        } catch (SQLException ex) {
            System.out.println("Error: " + ex.getMessage());
        } finally {

            try {
                cn.close();
            } catch (SQLException ex) {
                System.out.println("Error: " + ex.getMessage());
            }

        }
    }
```

Stored Procedure

```
CREATE OR REPLACE PROCEDURE ObtenerDatosAlumno(p_id alumnos.id
%TYPE,
      p_nombre OUT alumnos.nombre%TYPE, p_sexo OUT alumnos.sexo
%TYPE,
      p_curso OUT alumnos.curso%TYPE) AS

BEGIN
    SELECT nombre,
          CASE sexo
               WHEN 'H' THEN 'HOMBRE'
               ELSE 'MUJER'
          END,
          curso
          INTO p_nombre, p_sexo, p_curso
          FROM alumnos WHERE id=p_id;
END;
```

¿Cómo generar log con log4j en una base de datos?

1. Configurar el appender JDBCAppender en el archivo log4j.properties:

```
# Define the file appender
log4j.appender.sql=org.apache.log4j.jdbc.JDBCAppender
log4j.appender.sql.URL=jdbc:h2:./Database/my

# Set Database Driver
log4j.appender.sql.driver=org.h2.Driver

# Set database user name and password
log4j.appender.sql.user=root
log4j.appender.sql.password=password
```

```
# Set the SQL statement to be executed.
log4j.appender.sql.sql=INSERT INTO LOGS VALUES ('%x', now()
,'%C','%p','%m')
# Define the xml layout for file appender
log4j.appender.sql.layout=org.apache.log4j.PatternLayout
```

2. Crear una tabla Logs enla base de datos:

```
CREATE TABLE LOGS
(
	USER_ID VARCHAR(20) NOT NULL,
	DATED DATETIME NOT NULL,
	LOGGER VARCHAR(50) NOT NULL,
	LEVEL VARCHAR(10) NOT NULL,
	MESSAGE VARCHAR(1000) NOT NULL
);
```

Ejercicio

- Crear una entidad que se llame Pacientes en la base de datos H2 que tenga los siguientes campos: nombre, apellido, domicilio, DNI, fecha de alta, usuario y password.
- Crear una connection a la base de datos e insertar una fila paciente. Luego, abrir una transacción (setAutocommit(false)) y asignar otro password con una sentencia update y, paso siguiente, generar una excepción (throw new Exception).
- Por último, corroborar con una consulta que el paciente existe y que el campo password mantuvo su valor inicial del punto 1.

Sincrónico

Comprobar versión de Java:

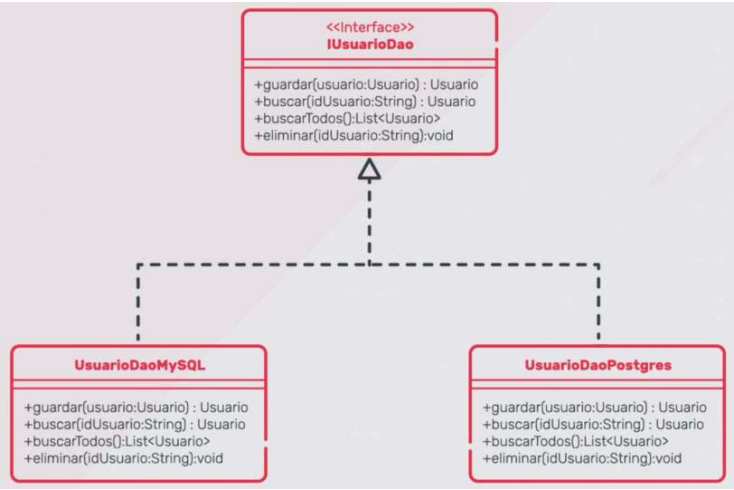
```
java -version
```

Configuración log4j:

```
log4j.rootLogger=DEBUG, stdout

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{yyyy-MM-dd HH:mm:ss}
%-5p %c{1}:%L - %m%n
```

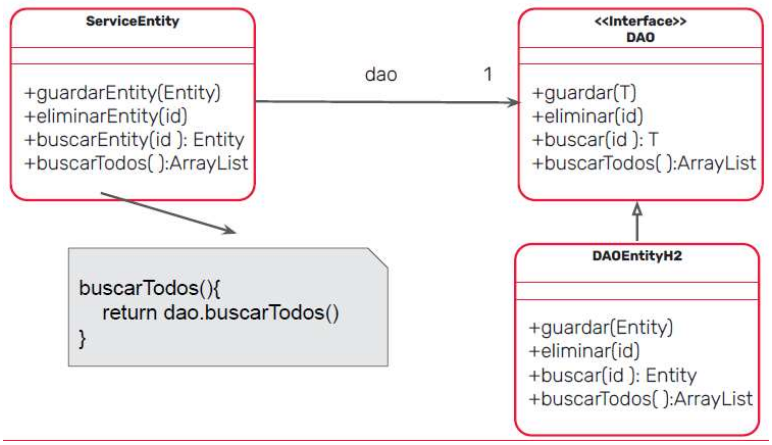
Sesión 14. Patrón DAO (Data Access Object)
Mayo 30 de 2022.



Propósito
Con el patrón DAO se desacoplan los datos propiamente dichos del lugar donde se almacenan o de la tecnología de almacenamiento. Es decir, para el sistema será indiferente si se utiliza PostgreSQL o MySQL ya que en cualquiera de los dos casos la forma de comunicarse será la misma.

Solución:
El patrón DAO propone:
1. Crear una interfaz en la que se definan todas las operaciones que se desee realizar, generalmente las más utilizadas son crear, actualizar, eliminar y leer.
2. Crear las diferentes implementaciones de esta interfaz, por ejemplo una implementación para postgreSQL y otra para MySQL.

De esta manera la capa de negocio, es decir donde se encuentra la lógica principal del sistema, se comunicará con la capa de persistencia, pero no conocerá los detalles de implementación ni se enterará si se está utilizando PostgreSQL o MySQL ya que cualquiera sea la implementación, tendrá los mismos métodos que la interfaz.



Capas
En un sistema se denomina “capa” a una agrupación de componentes (clases) de iguales responsabilidades.

Clases que intervienen:

Entity:
Son las clases de negocio Ej Estudiante, Curso, Cuenta, etc.

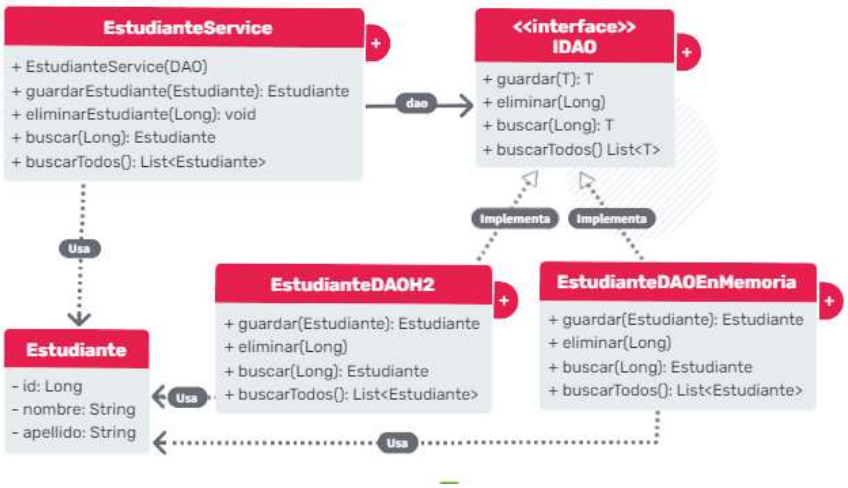
Clases ServiceEntity:
Por cada entidad se tendrá una clase de servicio. Las clases de servicio serán utilizadas por la capa de presentación y permiten desacoplar el acceso a datos de la vista.

Interface DAO:
Es la interface que obligará a las clases DAOs que la implementen a implementar las operaciones que se necesita realizar sobre la base de datos.

Clases DAO Entity:
Implementa la interface DAO y realizará todas estas operaciones en una base de datos en particular ej H2.

Las clases ServiceEntity tienen una referencia, es decir, un atributo llamado dao del tipo DAO. Este atributo puede ser cualquier clase que implemente dicha interface y esto permite en el futuro cambiar el mecanismo de persistencia mucho más fácil y de manera dinámica, simplemente apuntando a la clase service al nuevo DAO (dinamic binding).
DAO implementa el patrón Strategy, donde se tienen diferentes estrategias de persistencia que son las clases DAO.

Ejemplo



Se desea modelar un sistema para una academia y se requiere realizar un ABM para los estudiantes. El sistema debe tener la posibilidad de cambiar la forma de persistir los estudiantes, sin afectar el funcionamiento de la capa de servicio.

Ejemplo Patrón DAO

- Permite desacoplar el mecanismo de persistencia.
- Transformar una relación de herencia pura en composición.
- El patrón DAO es una solución de Strategy en el contexto de acceso a datos.

Patrón DAO: Diferentes estrategias de acceso a datos.

Diagrama UML para herencia pura:

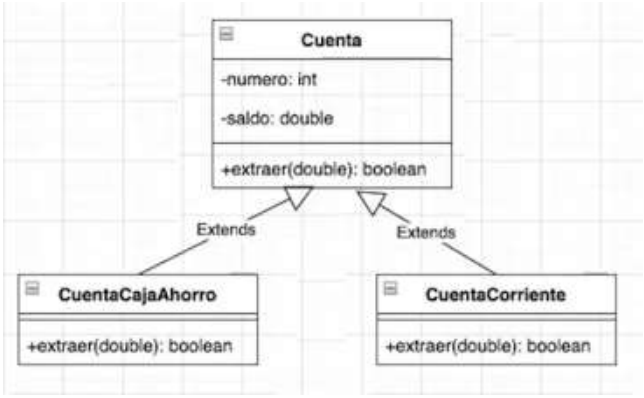
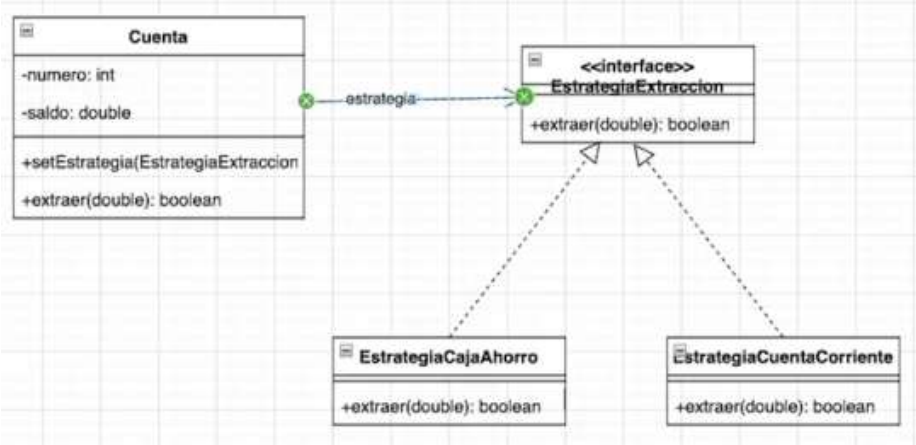
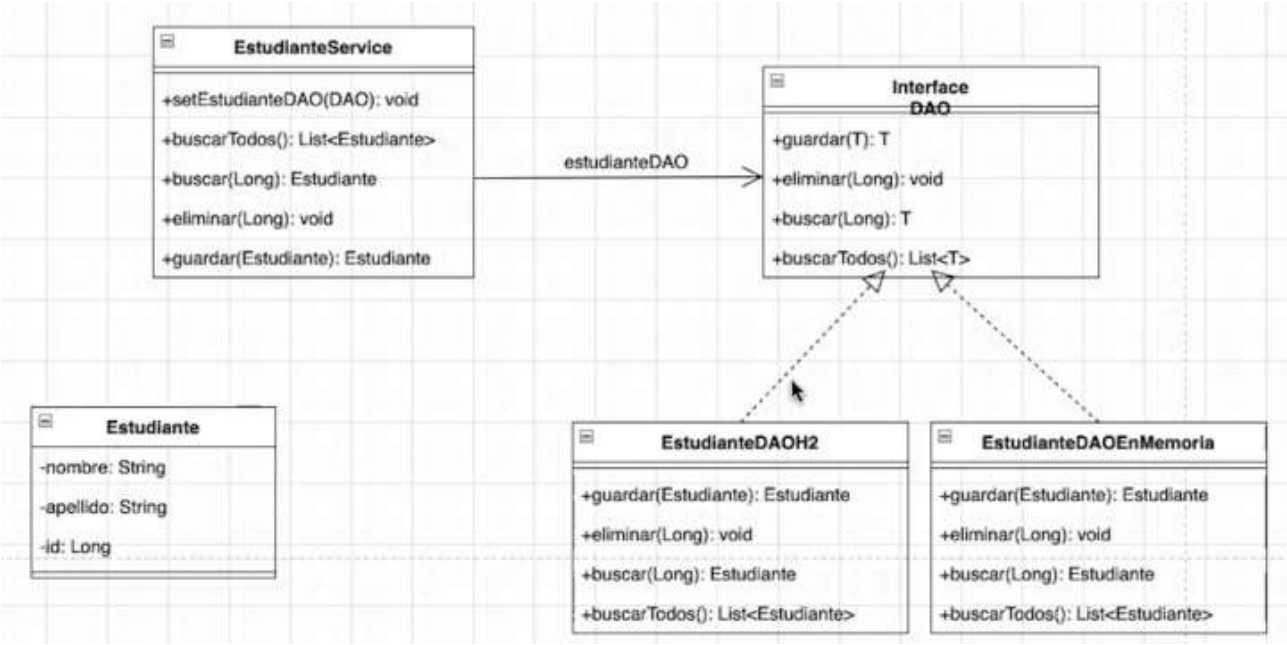


Diagrama UML para composición:



- La clase Cuenta debe tener un atributo ExtrategiaExtraccion y un método que permita setearlo.
- La interfaz EstrategiaExtracción debe recibir la ccuenta como parámetro.
- La Cuenta tiene un método extraer que delega la función de extraer al método correspondiente en la Interfaz Estrategia. Para que esto funcione, se debe pasar como parámetro tanto el monto de la transacción como la cuenta (this).

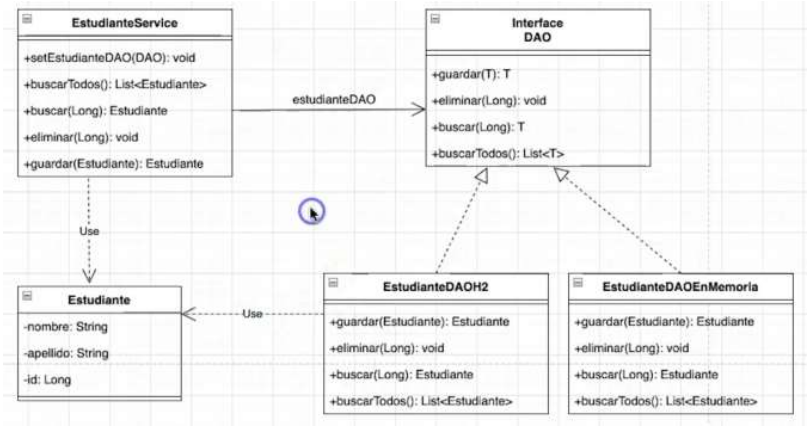
Implementación del patrón DAO



- En la interfaz se pueden utilizar objetos Generics (T) para que pueda ser aplicable a otras clases (Estudiantes, Profesores, Cursos, etc)

9 Pasos para la implementación del Patrón DAO:

1. Crear la entidad Estudiante.
La clase EstudianteService y la clase EstudianteDAOH2 hacen uso de la clase Estudiante.



2. Crear la interfaz IDAO.

Para recibir el tipo como parámetro se usa Generics:

public Interface Idao<T> {

```
}
```

3. Crear la clase de servicio: EstudianteService.

```
public class EstudianteService {
	private IDao<Estudiante> estudianteIDao; //Atributo para el mecanismo
	de persistencia
	// Otros atributos y métodos (setters, getters, etc)
	public Estudiante guardarEstudiante(Estudiante e) {
		estudianteIDao.guardar(e);
	}
	// Otros métodos que delegan las responsabilidades al DAO.
}
```

4. Crear la clase DAO: EstudianteDAOH2

Se requiere antes descargar H2 y crear la base de datos en H2 console.
Agregar el jar como dependencia.

Las variables de la conexión se definen como final static, ya que no cambian.
Lo sparámetros son Strings.

```
private final static String DB_JDBC_DRIVER = "org.h2.Driver";
private final static String DB_URL = "jdbc:h2:~/db_estudiantes"; // El mismo
nombre definido en H2 console.
private final static String DB_USER = "sa";
private final static String DB_PASSWORD = "";
```

5. Implementar el método guardar.

Pasos:
1. Levantar el driver y hacer la conexión

```
Connection conection = null;
Class.forName(DB_JDBC_DRIVER);
connection = DriverManager.getConnection(DB_URL, DB_USER,
DB_PASSWORD)
```

Dado que estas instrucciones pueden generar excepciones, se requiere incorporarlas en un bloque try-catch:

```
Connection conection = null;
try {
	Class.forName(DB_JDBC_DRIVER);
	connection = DriverManager.getConnection(DB_URL, DB_USER,
	DB_PASSWORD)
} catch(ClassNotFoundException e) {
	e.printStackTrace();
} catch(SQLException throwables) {
```

```
        throwables.printStackTrace();
}
```

2. Crear una sentencia

```
preparedStatement = connection.prepareStatement("INSERT INTO estudiantes
VALUES(?, ?, ?)");
preparedStatement.setLong(1, estudiante.getId());
preparedStatement.setString(2, estudiante.getNombre());
preparedStatement.setString(3, estudiante.getApellido());
```

3. Ejecutar la sentencia

```
preparedStatement.executeUpdate();
preparedStatement.close();
```

6. Implementar el método eliminar.

Pasos:

1. Levantar el driver y hacer la conexión

2. Crear una sentencia

```
preparedStatement = connection.prepareStatement("DELETE FROM
estudiantes WHERE id=?");
preparedStatement.setLong(1, id);
```

3. Ejecutar la sentencia

```
preparedStatement.executeUpdate();
preparedStatement.close();
```

7. implementar el método buscar.

Los pasos son similares a los casos anteriores.

1…
2…
3. Ejecutar la sentencia:

```
ResultSet result = preparedStatement.executeQuery();
preparedStatement.close();
```

4. Evaluar los resultados:

```
while(result.next()) {
        Long idEstudiante = result.getLong("id");
        String nombre = result.getString("nombre");
        String apellido = result.getString("apellido");
        estudiante = new Estudiante();
        estudiante.setId(idEstudiante);
```

```
        estudiante.setNombre(nombre);
        estudiante.setApellido(apellido);
}
```

8. Implementar el métdo buscarTodos
Similar al anterior, pero retorna una lista.

```
List<Estudiante> estudiantes = new ArrayList();
```

9. Probar la implementación.

Ejercicio

Una empresa de vuelos chárter desea desarrollar un sistema para llevar la gestión de su flota. Desean poder registrar sus aviones, tener la posibilidad de modificarlos, darlos de baja y visualizar toda la flota.

Luego del relevamiento se obtuvo la siguiente información:

Un Avión está compuesto por:

- Marca
- Modelo
- Matrícula
- Fecha de entrada en servicio
- Id

Se sugiere implementar el patrón DAO, utilizando H2 como base de datos e implementar los métodos que nos permitan:

- Registrar un nuevo avión.
- Buscar un avión por id.
- Eliminar un avión por id.
- Buscar todos los aviones.
- Se pide crear la capa de negocio y utilizarla para testear los 4 métodos con JUnit.

Para ejecutar el ejercicio es necesario agregar estas librerías al proyecto: H2 / JUnit / Hamrest-core.

Sincrónico

Examen: entregable hasta las 12 m del día sábado, hora Colombia.
Ejercitación 12S.

Temas: Junit, Tests, Log4J, H2.

Para permitir crear la base de datos:
https://stackoverflow.com/questions/55349373/database-not-found-and-ifexists-true-so-we-cant-auto-create-it

Cambiar la sentencia:
jdbc:h2:~/db_estudiantes

por:
jdbc:h2:mem:db_estudiantes

Sesión 15. Integradora 5.
Mayo 31 de 2022

Sesión 16. Taller de Coding
Junio 1 de 2022

Sesión 17. Examen.
Junio 3 de 2022.

https://docs.google.com/forms/d/e/1FAIpQLScazUvwreOGD8wrBlMnOasxiDX_HLGDhshu6XvyJusqQZpBPQ/viewform

Sesión 18. Maven.
Junio 3 de 2022.

Apache Maven

Es una librería que facilita la construcción de proyectos Java, el testeo, el empaquetado y el despliegue de los proyectos, ya sea en un servidor remoto o local. Además automatiza la gestión de dependencias (librerías).

Maven se centra en el concepto de archivos POM (Project Object Model). Un archivo POM es la representación del proyecto ante Maven, ya que contiene la configuración mínima para que el proyecto se ejecute correctamente.

Ejemplo

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.empresa.project-group</groupId>
  <artifactId>project</artifactId>
  <version>1.0</version>
<dependencies>
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.16</version>
</dependency>

<dependency>
   <groupId>junit</groupId>
   <artifactId>junit</artifactId>
   <version>4.12</version>
   <scope>test</scope>
</dependency>
```

- groupId, artifact y version son campos obligatorios para crear un proyecto Maven.
- En el grupo dependencies se agregan las dependencias requeridas para el proyecto.

Ciclo de vida de un proyecto Maven

Maven define una serie de fases por las que pasa el código:

- Maven validate
Valida que el proyecto tiene toda la información necesaria para ser procesado.

- Maven compile
Compila el código, transformando los archivos .java en .class y enviándolos a una carpeta de destino.

- Maven test
Ejecuta los tests unitarios que se tengan en el proyecto para asegurar el correcto funcionamiento del código.

- Maven package
Empaqueta el proyecto en un formato estándar de Java que permitirá luego ser ejcutado. Los posibles formatos del empaquetado son .jar, .war y .ear.

- Maven verify
Ejecuta los tests de integración para asegurar el correcto funcionamiento del proyecto.

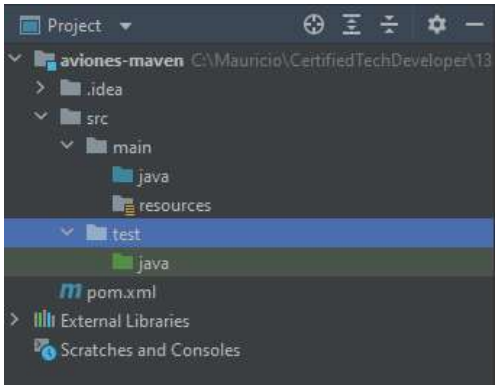
- Maven install
Envía el proyecto a un repositorio local para que otros proyectos puedan hacer uso de él agregándolo a sus dependencias.

- Maven deploy
Envía el proyecto a un repositorio remoto para que otros desarrolladores puedan descargarlo y hacer uso de él agregándolo a sus dependencias.

Ejemplo
Al crear el proyecto Maven, se crea el archivo POM para el cual es necesario definir el groupId y el artifact. JUnit, H2 y log4j se se agregan como dependencias, copiando el código de mvnrepository.com. Para que los test corran automáticamente, se agrega un plugin:

.

Estructura de carpetas por defecto creada por Maven:



Sincrónico

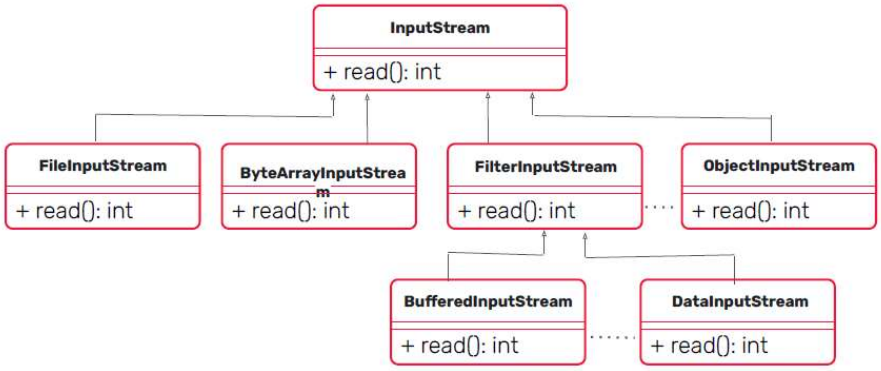
Versión requerida del plugin
3.0.0-M5

Sesión 19. Serialización de Objetos y E/S Archivos.
Junio 6 de 2022

Flujo de datos: Entrada y Salida

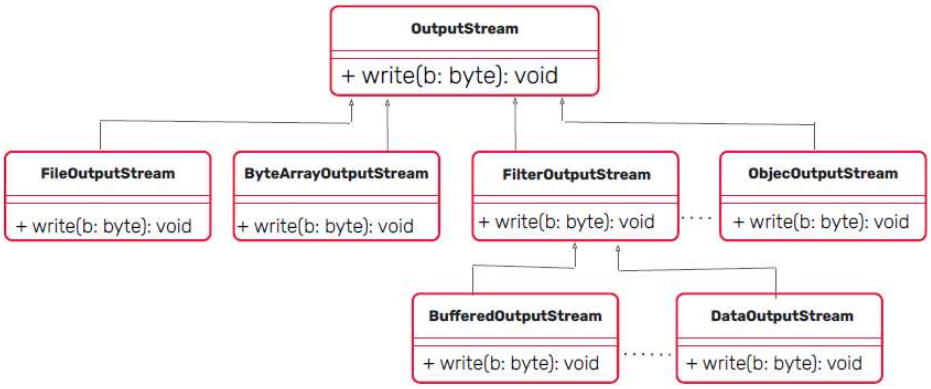
Los flujos de entrada y de salida sirven para leer y escribir datos desde y hacia la plataforma.

Lectura



La operación básica de los flujos de entrada es un método read().

Escritura

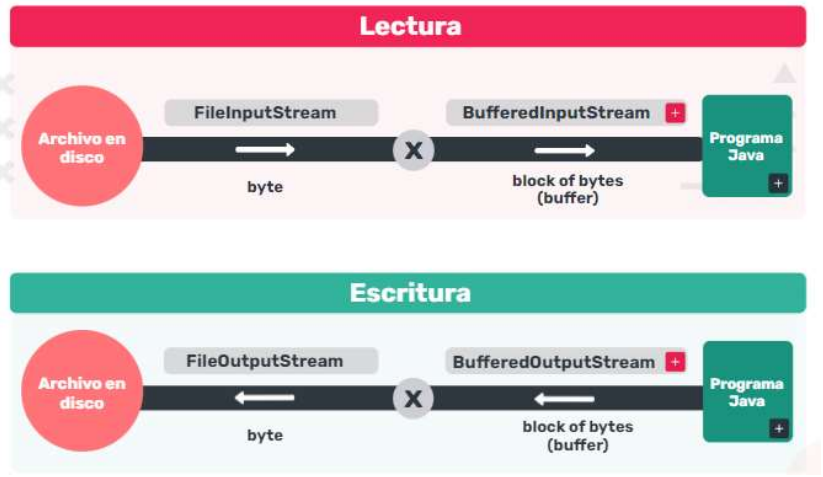


La operación básica de los flujos de salida es un método write().

Buffers

Todos los flujos de entrada y salida pueden funcionar de dos maneras: byte-a-byte o por "lotes". Es decir, que, si hay que leer un archivo de 1024 bytes, se puede leer 1024 veces un byte o se puede leer 8 veces un lote de 128 bytes. Para ello, los flujos binarios o de carácter establecen subclases "Buffered". Si bien los buffers dan velocidad a las operaciones de lectura, consumen algo más

de memoria. Por otro lado, no usar buffers consume menos memoria pero intensifica el uso de CPU.



Serializar un objeto

Para que un programa Java pueda convertir un objeto entero con todos sus atributos en un montón de bytes y pueda luego recuperarlo, el objeto necesita ser serializable. Al poder convertir el objeto a bytes, el mismo se puede, por ejemplo, enviar a través de la red o guardarlo en un archivo y después reconstruirlo al otro lado de la red o leerlo del archivo.

Para que un objeto sea serializable basta con que implemente la interfaz **Serializable**. Como esta interfaz no tiene métodos, es muy sencillo implementarla.

Ejemplo

```
import java.io.*;

public class Persona implements Serializable {
	private String nombre;
	private String apellido;
	private int DNI;
}
```

ObjectOutputStream y ObjectInputStream

Para poder convertir un objeto en un array de bytes, se necesita usar la clase ObjectOutputStream; y para poder reconstruir un objeto a partir de un array de bytes, la clase ObjectInputStream.

Ejemplo

```
import java.io.*;
public class Prueba{
	public static void main(String[] args) throws IOException {

		/*Guardar el objeto en un archivo*/
		FileOutputStream fo = new FileOutputStream("OutputFile.txt");
		ObjectOutputStream oos = new ObjectOutputStream(fo);
		oos.writeObject(persona); // persona es un objeto existente

		/*Recuperar el objeto en un archivo*/
		FileInputStream fi = new FileInputStream("OutputFile.txt");
		ObjectInputStream ois = new ObjectInputStream(fi);

		System.out.println(ois.readObject());
		dos.close();
	}
}
```

Ejercicio

Se desea persistir un listado de páginas favoritas en un archivo. Las páginas tienen una URL, que es la dirección web de la página del sitio y un nombre.

Probar en el método main guardar toda la colección de páginas en un archivo y luego recuperar en otra colección el contenido del archivo para mostrar la colección por consola.

Sesión 20. Integradora 6
Junio 6 de 2022.