



UNIVERSIDAD TECNOLÓGICA DEL PERÚ
Facultad de Ingeniería

Programación Orientada a Objetos

**Sesión 16: Programación orientado a objetos
con bases de datos**

Recordando:

¿Que vimos la clase pasada?



Logro de aprendizaje



Al finalizar la sesión, el estudiante soluciona problemas aplicando la Programación orientado a objetos con bases de datos usando Java en la resolución de ejercicios.

Utilidad

La programación orientada a objetos (POO) es una metodología que organiza el software como una colección de objetos que encapsulan datos y comportamientos relacionados combinado con bases de datos en Java nos permite crear aplicaciones robustas, mantenibles y eficientes.

Saberes Previos

- Conceptos básicos de programación orientada a objetos.
- Familiaridad con el lenguaje de programación (preferiblemente Java).
- Comprender la importancia de la reutilización de código y la mantenibilidad del software.

Ventajas de POO con bases de datos

- **Mantenimiento y Reusabilidad del Código:**

La POO permite la creación de clases y objetos reutilizables. Esto facilita el mantenimiento del código y la implementación de cambios sin afectar otras partes del sistema.

- **Encapsulamiento:**

El encapsulamiento permite que los detalles internos de los objetos permanezcan ocultos. Solo los métodos definidos en la clase pueden acceder y modificar los datos, lo que mejora la seguridad y la integridad de los datos.

Ventajas de POO con bases de datos

- **Abstracción:**

La abstracción permite que los desarrolladores se enfoquen en la lógica del negocio sin preocuparse por los detalles de implementación. Esto es especialmente útil cuando se trabaja con bases de datos, ya que permite manipular datos a nivel de objeto sin tener que manejar directamente las consultas SQL.

- **Herencia:**

La herencia permite que las nuevas clases reutilicen y extiendan el comportamiento de las clases existentes. Esto es útil en aplicaciones con bases de datos complejas, ya que permite crear jerarquías de clases que reflejan la estructura de los datos.

Ventajas de POO con bases de datos

- **Polimorfismo:**

El polimorfismo permite que los objetos de diferentes clases se traten de manera uniforme. En el contexto de bases de datos, esto significa que los métodos pueden operar en diferentes tipos de objetos de manera uniforme, simplificando la manipulación de datos.

Integración de POO y Bases de Datos en Java

Java ofrece varias herramientas y bibliotecas que facilitan la integración de POO con bases de datos:

- **JDBC (Java Database Connectivity):** JDBC es una API estándar de Java para interactuar con bases de datos relacionales. Permite ejecutar consultas SQL y manipular resultados de manera eficiente.
- **ORM (Object-Relational Mapping):** Los frameworks ORM, como Hibernate, permiten mapear clases Java a tablas de bases de datos. Esto simplifica la manipulación de datos al permitir trabajar con objetos en lugar de consultas SQL.
- **JPA (Java Persistence API):** JPA es una API estándar de Java para la persistencia de datos. Proporciona una abstracción sobre los frameworks ORM y facilita el trabajo con bases de datos relacionales.

Agenda

- Programación orientado a objetos con bases de datos.

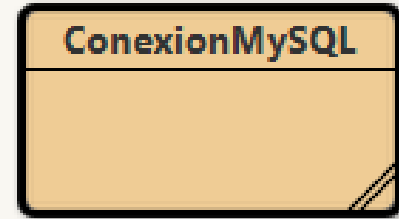


Ejercicio 1

Aquí tienes un ejemplo básico de cómo podrías conectar tu aplicación Java a una base de datos MySQL:

Deberás tener:

- Xampp
- Conector MYSQL
- Base de datos:nombre_base_de_datos



Ejercicio 1

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class ConexionMySQL {
    private static final String URL =
"jdbc:mysql://localhost:3306/nombre_base_de_datos";
    private static final String USER = "root";
    private static final String PASSWORD = "";
    public static void main(String[] args) {
        Connection conexion = null;
        try {
            conexion = DriverManager.getConnection(URL, USER, PASSWORD);
            if (conexion != null) {
                System.out.println("Conexión exitosa a la base de datos.");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Ejercicio 1

```
// Aquí puedes realizar consultas, actualizaciones, etc.
    }
    } catch (SQLException e) {
        System.out.println("Error al conectar a la base de datos: " + e.getMessage());
    } finally {
        try {
            if (conexion != null && !conexion.isClosed()) {
                conexion.close();
            }
        } catch (SQLException e) {
            System.out.println("Error al cerrar la conexión: " + e.getMessage());
        }
    }
}
```

Explicación del código

Asegúrate de reemplazar nombre_base_de_datos, tu_usuario y tu_contraseña con los valores correctos para tu base de datos MySQL.

```
public class ConexionMySQL {  
    private static final String URL =  
    "jdbc:mysql://localhost:3306/nombre_base_de_datos";  
    private static final String USER = "tu_usuario";  
    private static final String PASSWORD = "tu_contraseña";
```

Explicación del código

Manejar excepciones y cerrar la conexión

Siempre es importante manejar las excepciones adecuadamente y cerrar la conexión a la base de datos cuando hayas terminado de usarla, para liberar recursos.

Ejecutar y probar la conexión

Guarda tu código y ejecútalo para probar la conexión a la base de datos MySQL desde NetBeans.

Recuerda que estos son pasos generales y puede haber diferencias dependiendo de tu entorno específico y la configuración de tu base de datos.

Ejercicio 2

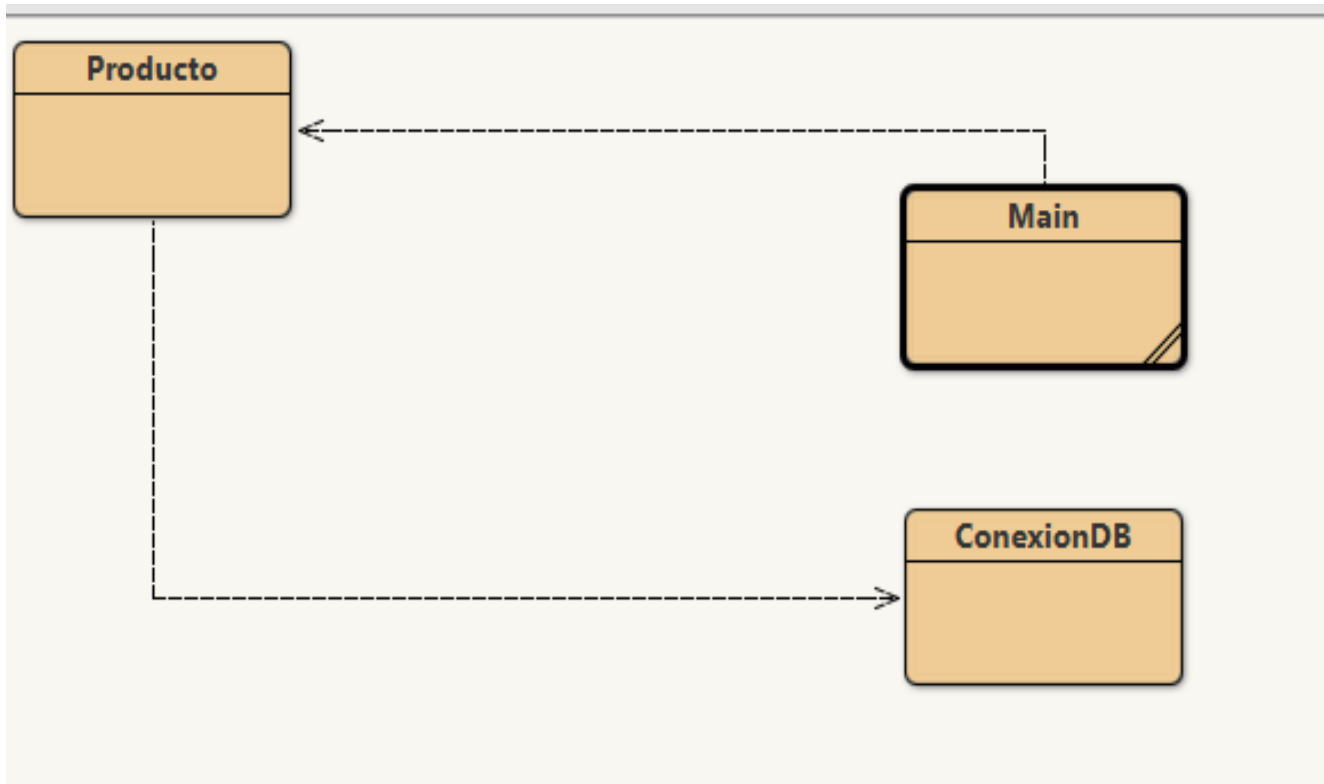
En este ejemplo, se crea un nuevo producto, se guarda en la base de datos, se actualiza su stock y se busca un producto por su nombre en la base de datos.

Base de datos: nombre_base_de_datos

Tabla: Productos

```
create database nombre_base_de_datos  
CREATE TABLE `productos` ( `id` int(11) NOT NULL,  
`nombre` varchar(50) NOT NULL, `precio`  
decimal(10,0) NOT NULL, `stock` int(11) NOT NULL)
```


Ejercicio 2



Ejercicio 2

Clase Producto

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class Producto {
    private int id;
    private String nombre;
    private double precio;
    private int stock;

    public Producto(int id, String nombre, double precio, int stock) {
        this.id = id;
        this.nombre = nombre;
        this.precio = precio;
        this.stock = stock;
    }
}
```

Ejercicio 2

Clase Producto

```
public int getId() {  
    return id;  
}  
  
public void setId(int id) {  
    this.id = id;  
}  
  
public String getNombre() {  
    return nombre;  
}  
  
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}  
  
public double getPrecio() {  
    return precio;  
}  
  
public void setPrecio(double precio) {  
    this.precio = precio;  
}  
  
public int getStock() {  
    return stock;  
}  
  
public void setStock(int stock) {  
    this.stock = stock;  
}
```



Ejercicio 2

Método para guardar el producto en la base de datos

```
public void guardarProductoEnDB() {  
    Connection conexion = ConexionDB.obtenerConexion();  
    if (conexion != null) {  
        try {  
            String query = "INSERT INTO Productos (id, nombre, precio, stock) VALUES (?, ?, ?, ?)";  
            PreparedStatement statement = conexion.prepareStatement(query);  
            statement.setInt(1, this.id);  
            statement.setString(2, this.nombre);  
            statement.setDouble(3, this.precio);  
            statement.setInt(4, this.stock);  
  
            int filasInsertadas = statement.executeUpdate();  
            if (filasInsertadas > 0) {  
                System.out.println("Producto guardado en la base de datos.");  
            } else {  
                System.out.println("No se pudo guardar el producto.");  
            }  
            statement.close();  
        } catch (SQLException e) {  
            System.out.println("Error al guardar el producto: " + e.getMessage());  
        } finally {  
            ConexionDB.cerrarConexion();  
        }  
    }  
}
```

Ejercicio 2

Método para actualizar el stock del producto en la bd

```
public void actualizarStockEnDB(int cantidad) {
    Connection conexion = ConexionDB.obtenerConexion();
    if (conexion != null) {
        try {
            String query = "UPDATE Productos SET stock = stock + ? WHERE id = ?";
            PreparedStatement statement = conexion.prepareStatement(query);
            statement.setInt(1, cantidad);
            statement.setInt(2, this.id);

            int filasActualizadas = statement.executeUpdate();
            if (filasActualizadas > 0) {
                System.out.println("Stock actualizado en la base de datos.");
            } else {
                System.out.println("No se pudo actualizar el stock.");
            }
            statement.close();
        } catch (SQLException e) {
            System.out.println("Error al actualizar el stock: " + e.getMessage());
        } finally {
            ConexionDB.cerrarConexion();
        }
    }
}
```

Ejercicio 2

Método para buscar productos por nombre en la base de datos

```
public static Producto buscarProductoPorNombre(String nombre) {
    Connection conexion = ConexionDB.obtenerConexion();
    Producto productoEncontrado = null;
    if (conexion != null) {
        try {
            String query = "SELECT * FROM Productos WHERE nombre = ?";
            PreparedStatement statement = conexion.prepareStatement(query);
            statement.setString(1, nombre);

            ResultSet resultado = statement.executeQuery();
            if (resultado.next()) {
                int id = resultado.getInt("id");
                double precio = resultado.getDouble("precio");
                int stock = resultado.getInt("stock");
                productoEncontrado = new Producto(id, nombre, precio, stock);
            } else {
                System.out.println("Producto no encontrado.");
            }
            statement.close();
        } catch (SQLException e) {
            System.out.println("Error al buscar el producto: " + e.getMessage());
        } finally {
            ConexionDB.cerrarConexion();
        }
    }
    return productoEncontrado;
}
```

Ejercicio 2

Método main

```
public class Main {  
    public static void main(String[] args) {  
        // Ejemplo de uso de los métodos de la clase Producto  
        // Crear un nuevo producto  
        Producto nuevoProducto = new Producto(1, "Martillo", 15.99, 50);  
        // Guardar el producto en la base de datos  
        nuevoProducto.guardarProductoEnDB();  
        // Actualizar el stock del producto en la base de datos  
        nuevoProducto.actualizarStockEnDB(10);  
        // Buscar un producto por nombre en la base de datos  
        Producto productoEncontrado =  
Producto.buscarProductoPorNombre("Martillo");  
        if (productoEncontrado != null) {  
            System.out.println("Producto encontrado: " +  
productoEncontrado.getNombre() + " - Stock: " +  
productoEncontrado.getStock());  
        } else {  
            System.out.println("Producto no encontrado.");  
        }  
    }  
}
```

Clase ConexionDB :Conexión a la base de datos con JDBC

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class ConexionDB {
    private static final String URL =
"jdbc:mysql://localhost:3306/nombre_base_de_datos";
    private static final String USER = "root";
    private static final String PASSWORD = "";
    private static Connection conexion = null;
    public static Connection obtenerConexion() {
        try {
            conexion = DriverManager.getConnection(URL, USER, PASSWORD);
            if (conexion != null) {
                System.out.println("Conexión exitosa a la base de datos.");
            }
        }
    }
}
```


Clase ConexionDB :Conexión a la base de datos con JDBC

```
} catch (SQLException e) {  
    System.out.println("Error al conectar con la base de datos: " + e.getMessage());  
}  
return conexion;  
}  
public static void cerrarConexion() {  
    try {  
        if (conexion != null) {  
            conexion.close();  
            System.out.println("Conexión cerrada.");  
        }  
    } catch (SQLException e) {  
        System.out.println("Error al cerrar la conexión: " + e.getMessage());  
    }  
}  
}
```

Explicación del código

Clase Producto

Esta clase representa un producto en una tienda o inventario. Tiene varios atributos y métodos para gestionar el producto y sus interacciones con una base de datos.

Métodos para interactuar con la base de datos

- **guardarProductoEnDB()**

Este método guarda un producto en la base de datos. Usa una conexión obtenida de ConexionDB. Prepara una consulta SQL para insertar un nuevo registro en la tabla Productos con los valores del producto actual. Si la inserción es exitosa, imprime un mensaje confirmando la operación.

Explicación del código

Métodos para interactuar con la base de datos

- **actualizarStockEnDB(int cantidad)**
Este método actualiza el stock de un producto en la base de datos. Añade la cantidad especificada al stock actual del producto identificado por su id.
- **buscarProductoPorNombre(String nombre)**
Este método busca un producto en la base de datos por su nombre. Si encuentra un producto con el nombre especificado, crea y devuelve un objeto Producto con los datos encontrados.


Explicación del código

Clase `ConexionDB`

Esta clase gestiona la conexión a la base de datos.

Atributos y Métodos

java

 Copiar código


```
private static final String URL = "jdbc:mysql://localhost:3306/nombre_basedatos";  
private static final String USER = "usuario";  
private static final String PASSWORD = "contraseña";  
private static Connection conexion = null;
```

Define la URL de la base de datos, el usuario y la contraseña para la conexión.

Explicación del código

`obtenerConexion()`

java

 Copiar código


```
public static Connection obtenerConexion() {  
    try {  
        conexion = DriverManager.getConnection(URL, USER, PASSWORD);  
        if (conexion != null) {  
            System.out.println("Conexión exitosa a la base de datos.");  
        }  
    } catch (SQLException e) {  
        System.out.println("Error al conectar con la base de datos: " + e.getMessage());  
    }  
    return conexion;  
}
```

Este método obtiene una conexión a la base de datos usando los datos proporcionados.

Explicación del código

`cerrarConexion()`

java

 Copiar código

```
public static void cerrarConexion() {  
    try {  
        if (conexion != null) {  
            conexion.close();  
            System.out.println("Conexión cerrada.");  
        }  
    } catch (SQLException e) {  
        System.out.println("Error al cerrar la conexión: " + e.getMessage());  
    }  
}
```

Este método cierra la conexión a la base de datos si está abierta.

Explicación del código

Clase `Main`

La clase principal para ejecutar el código y probar los métodos.

```
java Copiar código

public class Main {
    public static void main(String[] args) {
        // Ejemplo de uso de los métodos de la clase Producto
        Producto nuevoProducto = new Producto(1, "Martillo", 15.99, 50);
        nuevoProducto.guardarProductoEnDB();
        nuevoProducto.actualizarStockEnDB(10);
        Producto productoEncontrado = Producto.buscarProductoPorNombre("Martillo");
        if (productoEncontrado != null) {
            System.out.println("Producto encontrado: " + productoEncontrado.getNombre() +
        } else {
            System.out.println("Producto no encontrado.");
        }
    }
}
```

En este ejemplo, se crea un nuevo producto, se guarda en la base de datos, se actualiza su stock y se busca un producto por su nombre en la base de datos.

Practica

Implementa los ejercicios vistos.

Conclusiones:

- La combinación de la programación orientada a objetos con bases de datos en Java permite crear aplicaciones robustas, mantenibles y eficientes.
- Utilizando herramientas como JDBC, ORM y JPA, los desarrolladores pueden manipular datos de manera intuitiva y centrarse en la lógica del negocio, mejorando la productividad y la calidad del software.



Cierre

¿Qué hemos aprendido hoy?

Bibliografía

- MORENO PÉREZ, J. “Programación orientada a objetos”. RA-MA Editorial.
<https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=31933>
- Vélez Serrano, José. “Diseñar y programar, todo es empezar: una introducción a la Programación Orientada a Objetos usando UML y Java”.
Dykinson. <https://tubiblioteca.utp.edu.pe/cgi-bin/koha/opac-detail.pl?biblionumber=36368>