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?

Universidad Tecnológica del Perú

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.

Universidad Tecnológica del Perú

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.

Universidad Tecnológica del Perú

Ejercicio 1

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

- Xammp
- Conector MYSQL
- Base de datos:nombre_base_de_datos

Recordando

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 con NetBeans, BueJ u otro IDE.

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

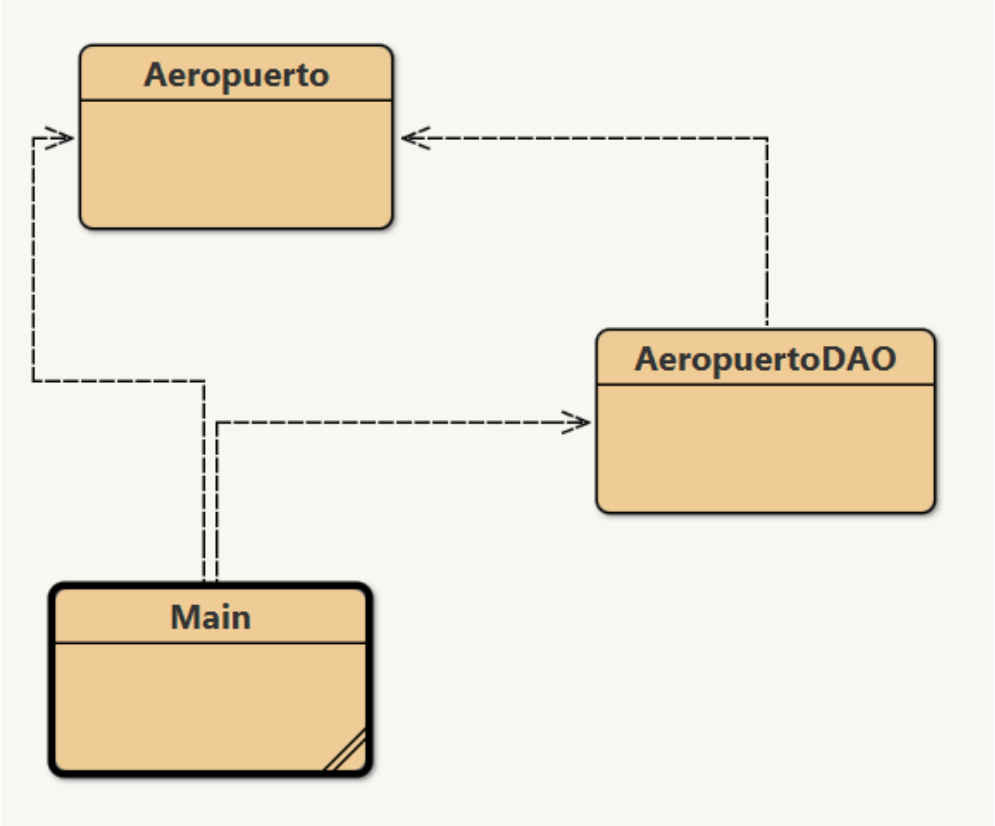
Recordando

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";
```

```
CREATE TABLE aeropuertos (    id INT AUTO_INCREMENT
PRIMARY KEY,    nombre VARCHAR(255) NOT NULL,    ciudad
VARCHAR(255) NOT NULL);
```

Ejercicio 1-Clase Aeropuerto



Universidad Tecnológica del Perú

Ejercicio 1-Clase Aeropuerto

```
public class Aeropuerto {
    private int id;
    private String nombre;
    private String ciudad;

    public Aeropuerto(int id, String nombre, String ciudad) {
        this.id = id;
        this.nombre = nombre;
        this.ciudad = ciudad;
    }

    public Aeropuerto() {
    }
```

Ejercicio 1
Clase Aeropuerto

```
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 String getCiudad() {
        return ciudad;
    }

    public void setCiudad(String ciudad) {
        this.ciudad = ciudad;
    }

}
```

Clase AeropuertoDAO

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class AeropuertoDAO {
    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 Connection getConnection() throws SQLException {
        return DriverManager.getConnection(URL, USER, PASSWORD);
    }
```

Clase AeropuertoDAO-método agregarAeropuerto

```
public void agregarAeropuerto(Aeropuerto aeropuerto) {
        String sql = "INSERT INTO aeropuertos(nombre, ciudad) VALUES (?, ?)";
        try (Connection con = getConnection();
             PreparedStatement pstmt = con.prepareStatement(sql)) {
            pstmt.setString(1, aeropuerto.getNombre());
            pstmt.setString(2, aeropuerto.getCiudad());
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
```

Clase AeropuertoDAO-método listarAeropuertos

```
public List<Aeropuerto> listarAeropuertos() {
        List<Aeropuerto> aeropuertos = new ArrayList<>();
        String sql = "SELECT * FROM aeropuertos";
        try (Connection con = getConnection();
             PreparedStatement pstmt = con.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                Aeropuerto aeropuerto = new Aeropuerto();
                aeropuerto.setId(rs.getInt("id"));
                aeropuerto.setNombre(rs.getString("nombre"));
                aeropuerto.setCiudad(rs.getString("ciudad"));
                aeropuertos.add(aeropuerto);
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        return aeropuertos;
    }
```

Clase AeropuertoDAO-método actualizarAeropuerto

```
// Método para actualizar un aeropuerto
  public void actualizarAeropuerto(Aeropuerto aeropuerto) {
     String sql = "UPDATE aeropuertos SET nombre = ?, ciudad = ? WHERE id = ?";
     try (Connection con = getConnection();
          PreparedStatement pstmt = con.prepareStatement(sql)) {
        pstmt.setString(1, aeropuerto.getNombre());
        pstmt.setString(2, aeropuerto.getCiudad());
        pstmt.setInt(3, aeropuerto.getId());
        pstmt.executeUpdate();
     } catch (SQLException ex) {
        ex.printStackTrace();
     }
  }
```

Clase AeropuertoDAO-método eliminarAeropuerto

```
// Método para eliminar un aeropuerto
    public void eliminarAeropuerto(int id) {
        String sql = "DELETE FROM aeropuertos WHERE id = ?";
        try (Connection con = getConnection();
             PreparedStatement pstmt = con.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            pstmt.executeUpdate();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }
}
```

Clase Principal (para probar las operaciones)

```
import java.util.List;

public class Main {
    public static void main(String[] args) {
        AeropuertoDAO aeropuertoDAO = new AeropuertoDAO();

        // Agregar un nuevo aeropuerto
        Aeropuerto nuevoAeropuerto = new Aeropuerto(1, "Aeropuerto
Internacional", "Ejemplo");
        aeropuertoDAO.agregarAeropuerto(nuevoAeropuerto);

        // Listar todos los aeropuertos
        List<Aeropuerto> aeropuertos = aeropuertoDAO.listarAeropuertos();
        for (Aeropuerto aeropuerto : aeropuertos) {
            System.out.println("ID: " + aeropuerto.getId() + ", Nombre: " +
aeropuerto.getNombre() + ", Ciudad: " + aeropuerto.getCiudad());
        }
```

Clase Principal (para probar las operaciones)

```
// Actualizar un aeropuerto
        nuevoAeropuerto.setNombre("Nombre Aeropuerto Actualizado");
        nuevoAeropuerto.setCiudad("Ciudad Aeropuerto Actualizada");
        aeropuertoDAO.actualizarAeropuerto(nuevoAeropuerto);

        // Eliminar un aeropuerto
        aeropuertoDAO.eliminarAeropuerto(nuevoAeropuerto.getId());

        // Listar nuevamente todos los aeropuertos después de las operaciones
        aeropuertos = aeropuertoDAO.listarAeropuertos();
        for (Aeropuerto aeropuerto : aeropuertos) {
            System.out.println("ID: " + aeropuerto.getId() + ", Nombre: " +
aeropuerto.getNombre() + ", Ciudad: " + aeropuerto.getCiudad());
        }
    }
}
```

Explicación del codigo

Este código incluye:

- Un constructor completo en la clase Aeropuerto para simplificar la creación de instancias.
- Un método actualizarAeropuerto en AeropuertoDAO para actualizar los registros.
- Un método eliminarAeropuerto en AeropuertoDAO para eliminar registros.
- Uso adecuado de try-with-resources para manejar la conexión a la base de datos.
- Ejemplos de cómo agregar, actualizar y eliminar aeropuertos en la clase Main.

Practica

Implementa el ejercicio visto.

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.

Universidad Tecnológica del Perú

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

Universidad Tecnológica del Perú