Implementación de Transacciones en Aplicaciones Java utilizando Hibernate

Tema: Garantizando la Consistencia de Múltiples Operaciones con Hibernate

Este proyecto demuestra cómo manejar transacciones complejas en Java, integrando operaciones dependientes (registro de cliente, verificación y actualización de disponibilidad, y registro de reserva) en una única transacción. Se implementa utilizando Hibernate para aprovechar sus facilidades en el manejo de transacciones y persistencia de datos.

Tecnologías

- Java SE 8+
- Hibernate ORM
- MySQL

Estructura del Proyecto

```
reserva-canchas/
├── src/
│   └── com/
│       └── centrodeportivo/
│           ├── model/
│           │   ├── Cliente.java
│           │   ├── Cancha.java
│           │   └── Reserva.java
│           ├── dao/
│           │   └── ReservaDAO.java
│           ├── service/
│           │   └── ReservaService.java
│           └── Main.java
├── resources/
│   └── hibernate.cfg.xml
└── database/
    └── schema.sql
```

Base de Datos (`schema.sql`)

```sql
CREATE DATABASE IF NOT EXISTS centro_deportivo;
USE centro_deportivo;

CREATE TABLE IF NOT EXISTS Cliente (
```

```
    id INT PRIMARY KEY AUTO_INCREMENT,
    nombre VARCHAR(100) NOT NULL,
    telefono VARCHAR(20) NOT NULL
);

CREATE TABLE IF NOT EXISTS Cancha (
    id INT PRIMARY KEY AUTO_INCREMENT,
    tipo VARCHAR(50) NOT NULL,
    disponible BOOLEAN NOT NULL
);

CREATE TABLE IF NOT EXISTS Reserva (
    id INT PRIMARY KEY AUTO_INCREMENT,
    cliente_id INT,
    cancha_id INT,
    fecha DATETIME,
    FOREIGN KEY (cliente_id) REFERENCES Cliente(id),
    FOREIGN KEY (cancha_id) REFERENCES Cancha(id)
);

INSERT INTO Cancha (tipo, disponible) VALUES
('Fútbol', TRUE),
('Vóley', TRUE),
('Básquet', TRUE);
```

Configuración Hibernate (`hibernate.cfg.xml`)

```
<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD
3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/centro_deportivo</prop
erty>
        <property name="hibernate.connection.username">root</property>
        <property name="hibernate.connection.password"></property>
        <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="com.centrodeportivo.model.Cliente"/>
        <mapping class="com.centrodeportivo.model.Cancha"/>
        <mapping class="com.centrodeportivo.model.Reserva"/>
    </session-factory>
</hibernate-configuration>
```

Modelos (Clases Java con Anotaciones Hibernate)

Cliente.java

```
package com.centrodeportivo.model;

import javax.persistence.*;

@Entity
@Table(name = "Cliente")
public class Cliente {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false)
    private String nombre;

    @Column(nullable = false)
    private String telefono;

    // Getters y setters
    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 getTelefono() { return telefono; }
    public void setTelefono(String telefono) { this.telefono = telefono; }
}
```

Cancha.java

```
package com.centrodeportivo.model;

import javax.persistence.*;

@Entity
@Table(name = "Cancha")
public class Cancha {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(nullable = false)
    private String tipo;
```

```
    @Column(nullable = false)
    private boolean disponible;

    // Getters y setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getTipo() { return tipo; }
    public void setTipo(String tipo) { this.tipo = tipo; }

    public boolean isDisponible() { return disponible; }
    public void setDisponible(boolean disponible) { this.disponible = disponible;
}
}
```

Reserva.java

```
package com.centrodeportivo.model;

import javax.persistence.*;
import java.util.Date;

@Entity
@Table(name = "Reserva")
public class Reserva {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @ManyToOne
    @JoinColumn(name = "cliente_id")
    private Cliente cliente;

    @ManyToOne
    @JoinColumn(name = "cancha_id")
    private Cancha cancha;

    @Temporal(TemporalType.TIMESTAMP)
    private Date fecha;

    // Getters y setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public Cliente getCliente() { return cliente; }
    public void setCliente(Cliente cliente) { this.cliente = cliente; }

    public Cancha getCancha() { return cancha; }
    public void setCancha(Cancha cancha) { this.cancha = cancha; }
```

```
    public Date getFecha() { return fecha; }
    public void setFecha(Date fecha) { this.fecha = fecha; }
}
```

DAO

ReservaDAO.java

```
package com.centrodeportivo.dao;

import com.centrodeportivo.model.*;
import org.hibernate.Session;
import org.hibernate.Transaction;

import java.util.Date;

public class ReservaDAO {

    public void registrarReserva(Session session, Cliente cliente, int canchaId) {
        Transaction tx = session.beginTransaction();
        try {
            // Guardar cliente
            session.save(cliente);

            // Verificar cancha disponible
            Cancha cancha = session.get(Cancha.class, canchaId);
            if (!cancha.isDisponible()) {
                throw new RuntimeException("La cancha no está disponible");
            }

            // Actualizar disponibilidad
            cancha.setDisponible(false);
            session.update(cancha);

            // Crear reserva
            Reserva reserva = new Reserva();
            reserva.setCliente(cliente);
            reserva.setCancha(cancha);
            reserva.setFecha(new Date());
            session.save(reserva);

            tx.commit();
            System.out.println("Reserva realizada con éxito.");
        } catch (Exception e) {
            tx.rollback();
            throw e;
        }
    }
}
```

Servicio

ReservaService.java

```
package com.centrodeportivo.service;

import com.centrodeportivo.dao.ReservaDAO;
import com.centrodeportivo.model.Cliente;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ReservaService {

    private SessionFactory sessionFactory;

    public ReservaService() {
        sessionFactory = new Configuration().configure().buildSessionFactory();
    }

    public void realizarReserva(String nombre, String telefono, int canchaId) {
        try (Session session = sessionFactory.openSession()) {
            ReservaDAO reservaDAO = new ReservaDAO();
            Cliente cliente = new Cliente();
            cliente.setNombre(nombre);
            cliente.setTelefono(telefono);
            reservaDAO.registrarReserva(session, cliente, canchaId);
        }
    }
}
```

Main

```
package com.centrodeportivo;

import com.centrodeportivo.service.ReservaService;

public class Main {
    public static void main(String[] args) {
        ReservaService reservaService = new ReservaService();
        reservaService.realizarReserva("Luis Pérez", "987654321", 1);
    }
}
```

Explicación

1. **Transacciones declarativas con Hibernate**: Hibernate se encarga de manejar `commit` y `rollback` mediante la clase `Transaction`.
2. **Uso de anotaciones JPA**: Las entidades están mapeadas con `@Entity`, facilitando el trabajo con la base de datos.
3. **Integración limpia y sencilla**: Menos código JDBC manual y más enfoque en la lógica del negocio.

Conclusión

Con este proyecto, garantizamos consistencia y controlamos transacciones de forma efectiva utilizando Hibernate. Aprovechamos las ventajas de un ORM (Mapeo Objeto-Relacional) para un desarrollo más limpio y eficiente.