**Desarrollo de un Sistema de Gestión de Reservas de Eventos**

**Objetivo:** Desarrollar un sistema que permita la gestión de reservas de eventos como conferencias, bodas, y conciertos, implementando conceptos fundamentales de la programación orientada a objetos como herencia, encapsulación y métodos personalizados.

**REQUISITOS**

1. **Diagrama UML de las Clases**
   * **Clases, atributos y métodos:** Definir las clases necesarias, sus atributos y métodos.
   * **Herencia:** Mostrar la relación de herencia entre las clases.
2. **Crear un Paquete para las Clases**
   * Agrupar todas las clases dentro de un paquete Java.
3. **Realizar al Menos 3-4 Niveles de Herencia**
   * Definir una estructura de herencia multinivel con al menos tres niveles.
4. **Implementación de Clases**
   * **Constructores**
   * **Setters y Getters**
   * **Métodos Personalizados:** Crear métodos para calcular el costo de la reserva basado en el tipo de evento y los servicios seleccionados.
5. **Crear al Menos 6 Objetos Diferentes**
   * Utilizar las clases definidas para crear al menos 10 objetos diferentes.
6. **Repositorio en GitHub**
   * Realizar varios commits.

**ESPECIFICACIONES DEL SISTEMA**

**Tipos de Eventos**

* **Conferencia**
* **Boda**
* **Concierto**

**Servicios Adicionales**

1. **Conferencia**
   * Servicio de catering: $200 adicionales.
   * Equipo audiovisual: $100 adicionales.
2. **Boda**
   * Decoración floral: $500 adicionales.
   * Música en vivo: $300 adicionales.
3. **Concierto**
   * Seguridad adicional: $400 adicionales.
   * Stand de comida: $150 adicionales.

**PASOS PARA DESARROLLAR EL SISTEMA**

1. **Diseño del Diagrama UML**
   * Incluir todas las clases necesarias, sus atributos y métodos.
   * Mostrar las relaciones de herencia.
2. **Definir las Clases y el Paquete**
   * Crear un paquete en Java para todas las clases.
3. **Implementar las Clases**
   * **Clase base:** Reserva
   * **Clases derivadas:** Conferencia, Boda, Concierto
   * **Otras clases:** Evento, Servicio, Cliente
   * Implementar constructores, setters, getters y métodos personalizados.
4. **Implementar Herencia Multinivel**
   * Crear una clase base y al menos dos niveles de herencia adicionales.
5. **Métodos Personalizados**
   * Calcular el costo de la reserva según el tipo de evento y los servicios adicionales.
6. **Crear Objetos**
   * Instanciar al menos 6 objetos diferentes para probar el sistema.
7. **Uso de GitHub**

**Cálculos con Datos Quemados**

* **Reserva de Conferencia con Servicio de Catering:**
  + Costo base de Conferencia: $1000
  + Costo adicional por servicio de catering: $200
  + **Total:** $1200
* **Reserva de Boda con Decoración Floral y Música en Vivo:**
  + Costo base de Boda: $2000
  + Costo adicional por decoración floral: $500
  + Costo adicional por música en vivo: $300
  + **Total:** $2800

**1. Diseño del Diagrama UML**

**Clases y Relaciones:**

* **Reserva (Clase Base)**
  + Atributos: id, cliente, fecha
  + Métodos: calcularCosto()
* **Evento (Subclase de Reserva)**
  + Atributos: tipoEvento, costoBase
  + Métodos: calcularCosto()
* **Conferencia (Subclase de Evento)**
  + Métodos: calcularCosto()
* **Boda (Subclase de Evento)**
  + Métodos: calcularCosto()
* **Concierto (Subclase de Evento)**
  + Métodos: calcularCosto()
* **Servicio**
  + Atributos: nombre, costo
* **Cliente**
  + Atributos: nombre, email

El override se aplica cuando una subclase proporciona una implementación específica de un método que ya está definido en su clase base. En el contexto del sistema de reservas de eventos, el método calcularCosto se define en la clase base Reserva como un método abstracto, y luego se proporciona una implementación específica en cada una de las subclases (Conferencia, Boda, Concierto).

**Explicación de @Override**

1. **Evento**: La clase Evento extiende Reserva y declara un método abstracto calcularCosto, aplicando @Override.
2. **Conferencia**: La clase Conferencia extiende Evento y proporciona una implementación concreta de calcularCosto, aplicando @Override.
3. **Boda**: La clase Boda extiende Evento y proporciona una implementación concreta de calcularCosto, aplicando @Override.
4. **Concierto**: La clase Concierto extiende Evento y proporciona una implementación concreta de calcularCosto, aplicando @Override.