Universidad de las Fuerzas Armadas ESPE

Departamento de Ciencias de la Computación

Carrera: Ingeniería en Tecnologías de la Información

Modalidad en Línea

Programación Orientada a Objetos

Peer Reviewing 2

Grupo: 3

Integrantes:

Keyla Lucero Marcillo Flores

Issac Miguel Morales Alcoser

Barbara Camila Rodas Paredes

Rosa Silvana Quito Soto

Kevin Daniel Shagñay Arequipa

Fecha de entrega: 14/06/2024

Sangolquí – Ecuador

REVISIÓN POR PARES (PEER REVIEWING)

• Título y nombre del grupo del trabajo revisado.

Estudio de caso 2 (revisión grupo 1)

Tema: Software de Gestión Restaurante

• Nombre completo de cada persona del grupo revisado.

Nancy Maribel Defaz Almachi, Camila Nicole Shiguango Valverde, Marco Paul Cañarejo Rodríguez, José Eduardo Angulo Lucas

• Evaluación de la implementación de POO

Para la evaluación nos basados desde la descripción de cada clase, incluido el diagrama de clases y resumen de la tarea:

En esta evaluación nos encontramos con un sistema muy bien estructura, en donde demostraron todo el conocimiento adquirido, esto incluye la creación de clases e interfaces, métodos, variables, etc.

Los integrantes Implementan un código estructurado en Java de manera adecuada, siguiendo las buenas prácticas de programación y utilizando estructuras y funciones apropiadas. y presentan el código de manera ordenada y legible.

Los integrantes demuestran un buen entendimiento de Java y son capaces de utilizar correctamente la aplicación, proporcionan una documentación adecuada y une buen resumen del código de la aplicación, aunque puede haber áreas que podrían mejorarse en cuanto a la claridad o concisión.

Los atributos y los principios de poo están muy bien declarados, nos muestran un código limpio en donde se implementa correctamente los principios de la programación orienta a objetos, así como los niveles de acceso a los atributos son adecuados.

En esta evaluación podemos añadir algunos consejos que nos facilitarán la implementación de un buen sistema.

- Los nombres de clase e interfaz deben ser sustantivos, comenzando con una letra mayúscula.
- Los nombres de las variables deben ser sustantivos, comenzando con una letra minúscula.
- Los nombres de los métodos deben ser verbos, comenzando con una letra minúscula.
- Los nombres constantes deben tener todas las letras mayúsculas y las palabras deben estar separadas por guiones bajos.

Los principios de POO:

Herencia:

Se observa que las clases mesa, cliente, plato, reserva y restaurante, cada una de estas clases están diseñada por separado, es decir, no hay ninguna clase que herede características de la otra.

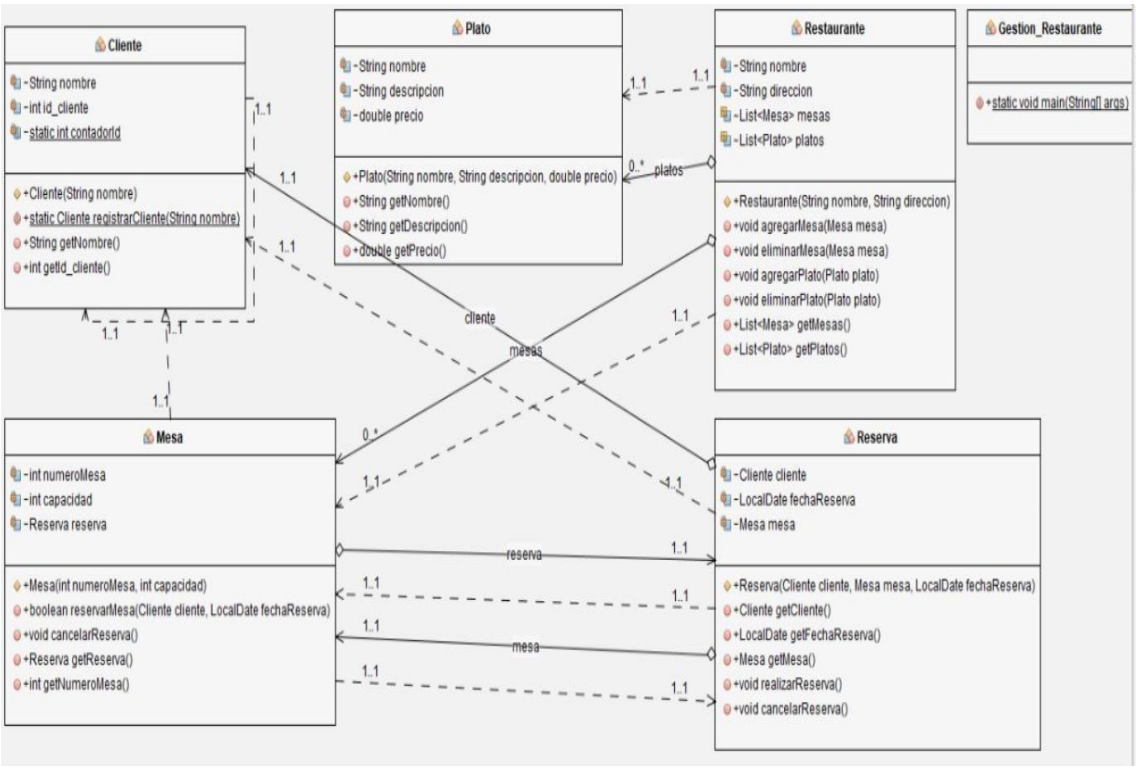
Se recomienda, que se podría usar la herencia para hacer el código más eficiente. Por ejemplo, si varias clases tienen atributos o métodos similares, podríamos crear una clase base, llamada Entidad, que tenga esos atributos o métodos comunes. Luego, las otras clases pueden heredar de Entidad, lo que significaría que automáticamente tendrán esos atributos o métodos y así evitaríamos la repetición de código.

La Encapsulación: Se observa que la encapsulación se maneja bien con los atributos privados y los métodos públicos, asegurándose de que todos los atributos sensibles estén correctamente encapsulados y accesibles solo mediante métodos getter y setter.

Una posible recomendación seria que se aseguren de que todos los datos sensibles (como id_cliente) estén protegidos usando esta técnica. Esto garantiza que no se puedan modificar directamente desde fuera de la clase, lo que ayuda a mantener la integridad de los datos.

- Las clases y sus relaciones están bien definidas en el documento revisado, pero también se podría considerar usar la herencia si se encuentra atributos o métodos comunes.

Evaluación del Diagrama UML:



Vamos a realizar un análisis detallado de todas las clases para asegurarnos de que cumplen con cada uno de los requerimientos necesarios:

La clase cliente representa a los clientes del restaurante, tiene atributos como nombre e id_cliente. El atributo id_cliente es privado, como se requiere en el enunciado.

La clase mesa representa las mesas del restaurante, tiene atributos como numeroMesa y capacidad. También tiene una relación con la clase Cliente a través del atributo id_cliente.

La clase plato: Representa los platos del menú. Tiene atributos como nombre y descripcion.

La clase reserva: representa las reservas de mesas, tiene atributos como fechaReserva, horaReserva y una relación con las clases Cliente y Mesa.

La clase gestión_Restaurante: Esta clase gestiona las listas de clientes, mesas, platos y reservas. Contiene métodos como agregarPlato(), registrarCliente(), reservarMesa() y cancelarReserva().

Sobre las relaciones se observa que hay una relación uno a muchos entre Cliente y Reserva, lo que significa que un cliente puede tener varias reservas. También hay una relación uno a uno entre Mesa y Reserva, indicando que cada mesa puede tener solo una reserva a la vez.

- Después de realizar el análisis de todas las clases se puede decir que el diagrama cumple con todos los requisitos especificados en el enunciado . La privacidad del atributo id_cliente en la clase Mesa se ha implementado correctamente, asegurando que los datos sensibles estén protegidos. Además, los métodos mencionados en el enunciado están presentes en las clases correspondientes, lo que indica que las funcionalidades requeridas se han tenido en cuenta. En conclusión, el diagrama está bien diseñado y representa de manera adecuada las entidades y las relaciones dentro del sistema de gestión de restaurantes, proporcionando una visión clara y organizada de cómo interactúan los diferentes componentes del sistema.

Claridad y complejitud del UML

En cuanto a claridad del UML está claro ya que presenta su estructura correcta con título, atributos y métodos de cada clase, visualmente es complejo ya que observamos que las relaciones se cruzan entre ellas.

Solución: Se podría mejorar al brindar una mejor estructura de este UML para una mejor comprensión del usuario y del código.

Algoritmo

No está muy claro el algoritmo ya que no presenta una correcta estructura, los temas de “Nuevorestaurante()”, “leer nombre_direccion” y “Devolver nuevo restaurante” no están unidas a ninguna parte de la estructura principal del algoritmo, falta el final, al ingresar los datos se debe hacer por medio de un paralelogramo, no un rectángulo, el

inicio y fin debe estar colocado en óvalos no en círculos, esto resulta en un algoritmo un poco confuso y complejo.

Solución: Se puede colocar las diferentes estructuras del algoritmo correctamente, comenzando con el inicio y fin los cuales deben ir en óvalos, el ingreso de datos se representa con un paralelogramo, las relaciones están correctas sin embargo tres estructuras no están conectadas, y se debe procurar disminuir la cantidad de opciones en la estructura de decisión.

Evaluación de las funcionalidades del sistema (relevancia, efectividad y posibles mejoras)

Relevancia:

El sistema desarrollado para la gestión de un restaurante es altamente relevante, ya que aborda varios aspectos cruciales de la operación diaria de un establecimiento gastronómico. Las funcionalidades implementadas, como la gestión de mesas, platos y reservas son esenciales para el manejo eficiente de un restaurante. La capacidad de registrar y gestionar reservas permite a los clientes asegurar una mesa, lo que mejora la experiencia del cliente y optimiza el uso del espacio del restaurante

Estas funcionalidades son relevantes y esenciales para la operación diaria de un establecimiento de este tipo.

Efectividad:

El código parece ser efectivo en cuanto a la implementación de las funcionalidades mencionadas. Se definen claramente las clases y sus respectivos métodos para manejar las operaciones relacionadas con mesas, platos, clientes y reservas. Además, se incluyen ejemplos de uso en el método main, lo que demuestra la efectividad del sistema para realizar tareas como agregar mesas y platos, registrar clientes, hacer y cancelar reservas, y mostrar el menú de platos.

Gestión de Mesas: La capacidad de agregar, eliminar y reservar mesas garantiza una administración flexible y eficiente del espacio disponible.

Gestión de Platos: El registro de platos con sus descripciones y precios facilita la actualización y la gestión del menú, permitiendo mantener información precisa y accesible para el personal y los clientes

Registro de Clientes y Reservas: El sistema permite el registro de clientes y la gestión de reservas mediante la asociación de clientes a mesas en fechas específicas, lo que asegura un seguimiento adecuado de las reservas y mejora la satisfacción del cliente

Posibles mejoras:

Notificaciones y Recordatorios: Integrar un sistema de notificaciones que envíe recordatorios a los clientes sobre sus reservas podría reducir el número de ausencias y mejorar la experiencia del cliente.

```
import java.time.LocalDate;
import java.time.LocalTime;

public class NotificacionReserva implements Runnable {
   private Reserva reserva;

   public NotificacionReserva(Reserva reserva) {
      this.reserva = reserva;
   }
   @Override
   public void run() {
      LocalTime notificacionHora = LocalTime.of(9, 0); // Notificación a las 9:00 AM
      while (true) {
         if (LocalDate.now().equals(reserva.getFechaReserva().minusDays(1)) &&
LocalTime.now().isAfter(notificacionHora)) {
            System.out.println("Recordatorio: Tienes una reserva mañana para " + reserva.getCliente().getNombre());
            break;
         }
      }
   }
   public static void main(String[] args) {
      Cliente cliente = new Cliente("Juan Pérez");
      Mesa mesa = new Mesa(1, 4);
      Reserva reserva = new Reserva(cliente, mesa, LocalDate.now().plusDays(1));
      Thread notificacionThread = new Thread(new NotificacionReserva(reserva));
      notificacionThread.start();
   }
}
```

Validación de entrada: Sería recomendable agregar validaciones para garantizar que los datos ingresados sean correctos y evitar errores o comportamientos inesperados.

```
public static boolean esNombreValido(String nombre) {
   return nombre != null && !nombre.isEmpty() && nombre.matches("[a-zA-Z ]+");
}
```

```
public static boolean esPrecioValido(double precio) {
    return precio > 0;
}
// Usar estas funciones antes de procesar los datos
if (esNombreValido(nombre) && esPrecioValido(precio)) {
    // Procesar datos
} else {
    // Mostrar mensaje de error
}
```

Otra posible mejora seria En la clase Restaurante, puedes agregar un método para obtener el plato por su nombre, lo que facilitaría la búsqueda y manipulación de platos específicos. Por ejemplo:

```
public class Restaurante {
    // ...
    public Plato obtenerPlatoPorNombre(String nombre) {
        for (Plato plato : platos) {
            if (plato.getNombre().equalsIgnoreCase(nombre)) {
                return plato;
            }
        }
        return null; // Si no se encuentra el plato
    }
    // ...
}
```

Luego, en el método main o en cualquier otra parte del código donde necesites acceder a un plato específico, puedes utilizar este nuevo método:

```
Restaurante restaurante = new Restaurante("El Sabor Casero", "Av. Principal #123");
// ...
Plato encebollado = restaurante.obtenerPlatoPorNombre("Encebollado");
if (encebollado != null) {
    System.out.println("Descripción del encebollado: " + encebollado.getDescripcion());
    // Realizar otras operaciones con el plato
} else {
    System.out.println("No se encontró el plato 'Encebollado' en el menú.");
}
```

Esta pequeña mejora permite buscar y obtener un plato específico por su nombre de manera más conveniente, en lugar de tener que recorrer manualmente la lista de platos.

Comentarios:

- La estructura general de este trabajo es buena, se puede observar de una manera clara la separación de responsabilidades entre las diferentes clases (Mesa, Cliente, Plato, Reserva, Restaurante).
- El uso de encapsulación a través de modificadores de acceso privados y métodos getter/setter es correcto.
- La implementación de relaciones entre clases (como la agregación entre Restaurante y Mesa/Plato) es correcta y muestra adecuadamente la estructura del sistema.
- El código está bien comentado, lo que facilita su comprensión.
- Han utilizado correctamente los casos de reserva exitosa y fallida, así como la cancelación de reservas.

Sugerencias:

- En el manejo de excepciones, se podría considerar la implementación de manejo de excepciones para casos como intentar cancelar una reserva inexistente o hacer una reserva en una fecha pasada.
- Implementar más funcionalidades para el manejo de errores. Por ejemplo, si un cliente intenta reservar una mesa que no existe, el sistema debería manejar esa situación con un mensaje de error claro. Considera agregar funcionalidades adicionales como la edición de reservas existentes o la búsqueda de mesas disponibles en un rango de fechas.
- También podrían explorar cómo se podría utilizar la herencia. Por ejemplo, si en el futuro quisieran tener diferentes tipos de mesas (interior, exterior, VIP) con comportamientos ligeramente diferentes.

CRITERIOS DE EVALUACIÓN	PUNTAJE	TOTAL
Implementación de POO (6 puntos)		
Correcto uso de atributos y métodos (2 puntos)	2	6
Adecuado encapsulamiento y acceso a los datos (2 puntos)	2	
Aplicación correcta de la herencia (2 puntos)	2	
Diagrama UML (6 puntos)		
Precisión y corrección de las relaciones entre clases (2 puntos)	2	4
Claridad y legibilidad del diagrama (2 puntos)	1	
Inclusión de todos los elementos necesarios (2 puntos)	1	
Funcionalidades del Sistema (6 puntos)		
Relevancia y utilidad de las funcionalidades propuestas (2 puntos)	4	6
Implementación efectiva de las funcionalidades (2 puntos)	2	
Sugerencias para posibles mejoras (2 puntos)		2
Comentarios y Sugerencias Adicionales (2 puntos)	2	
CALIFICACIÓN FINAL		**18/20**