

# **“UNIVERSIDAD LAICA ELOY ALFARO DE MANABÍ”**

Título:

**Informe Sistema de Gestión de Hoteles**

Autor:

**Jama Guamán Roosevelt Jair**

**Rugel Fajardo Dustin Isaac**

**Vélez Álvarez Cristopher Bryan**

Tutor:

**Ing. Anthony Legarda**

Curso:

**3er Nivel “A”**

Ciudad:

**Manta**

**2024**

## Tabla de contenido

1.	Objetivos principales:	5
2.	Enfoque del proyecto:	5
3.	Diagrama de Clases	6
3.1	Clases y sus relaciones	7
3.1.1	Clase Huéspedes	7
3.1.2	Clase Empleado	7
3.1.3	Clase Reserva	7
3.1.4	Clase Facturación	8
3.1.5	Clase Inventario	8
3.1.6	Clase Evento	8
3.2	Relaciones Clave:	9
3.3	Notas Adicionales:	9
4.	Documentación de clases principales del sistema Hotelero	10
4.1	Clase Huéspedes	10
4.1.1	Descripción	10
4.1.2	Atributos	10
4.1.3	Métodos principales	10
4.2	Clase Empleado (Hereda de Huespedes)	13
4.2.1	Descripcion	13
4.2.2	Atributos	13
4.2.3	Métodos Principales	14

4.3	Clase Inventario .....	16
4.3.1	Descripción.....	16
4.3.2	Atributos.....	16
4.3.3	Métodos principales.....	17
4.4	Clase Evento .....	19
4.4.1	Descripción.....	19
4.4.2	Métodos Principales .....	19
4.5	Clase Main (main_admin.py).....	20
4.5.1	Descripción.....	20
4.5.2	Componentes Principales .....	20
4.5.3	Funcionalidad del Menú Principal .....	21
5.	Clase Main (Main_usuario.py).....	22
6.	Clase Facturacion (Facturacion.py) .....	25
7.	Clase Reservas (Reservas.py).....	29
8.	Clases de Verificación (verificaciones.py).....	33
8.1	Clase VerificadorFecha .....	33
8.2	Clase VerificacionDatos .....	34
8.3	Sistema Principal - Hotel Manabita (Main.py).....	34
8.3.1	Descripción.....	34
8.3.2	Componentes Principales .....	35
•	Gestores inicializados .....	35

○	Componentes PrincipalesGestor de Usuario (user_main): Maneja todas las operaciones relacionadas con el sistema de usuario .....	35
○	Gestor de Administrador (admin_main): Maneja todas las operaciones relacionadas con el sistema administrativo .....	35
•	Funcionalidades base .....	35
○	Sistema de menús interactivo.....	35
○	Control de acceso a subsistemas .....	35
○	Gestión de salida del programa .....	35
8.3.3	Funcionalidad del Menú Principal .....	35
8.3.4	Aspectos técnicos .....	37
9.	Conclusión .....	37
10.	Análisis y Conclusión - Sistema de Gestión Hotel Manabita .....	38
10.1	Análisis .....	38
10.1.1	Arquitectura Modular.....	38
10.1.2	Jerarquía de Clases.....	38
10.1.3	Gestión de Datos .....	38

## **1. Objetivos principales:**

- Implementar un sistema de reservas: Permitir a los usuarios realizar, modificar y cancelar reservas de manera eficiente.
- Mejorar la atención al cliente: Proporcionar una interfaz para que el personal del hotel pueda gestionar las necesidades de los huéspedes de manera efectiva.
- Centralizar la gestión: Crear un sistema unificado que maneje tanto las operaciones administrativas como las interacciones con los usuarios.
- Aplicar principios de POO: Utilizar las ventajas de la programación orientada a objetos para crear un código modular y escalable.

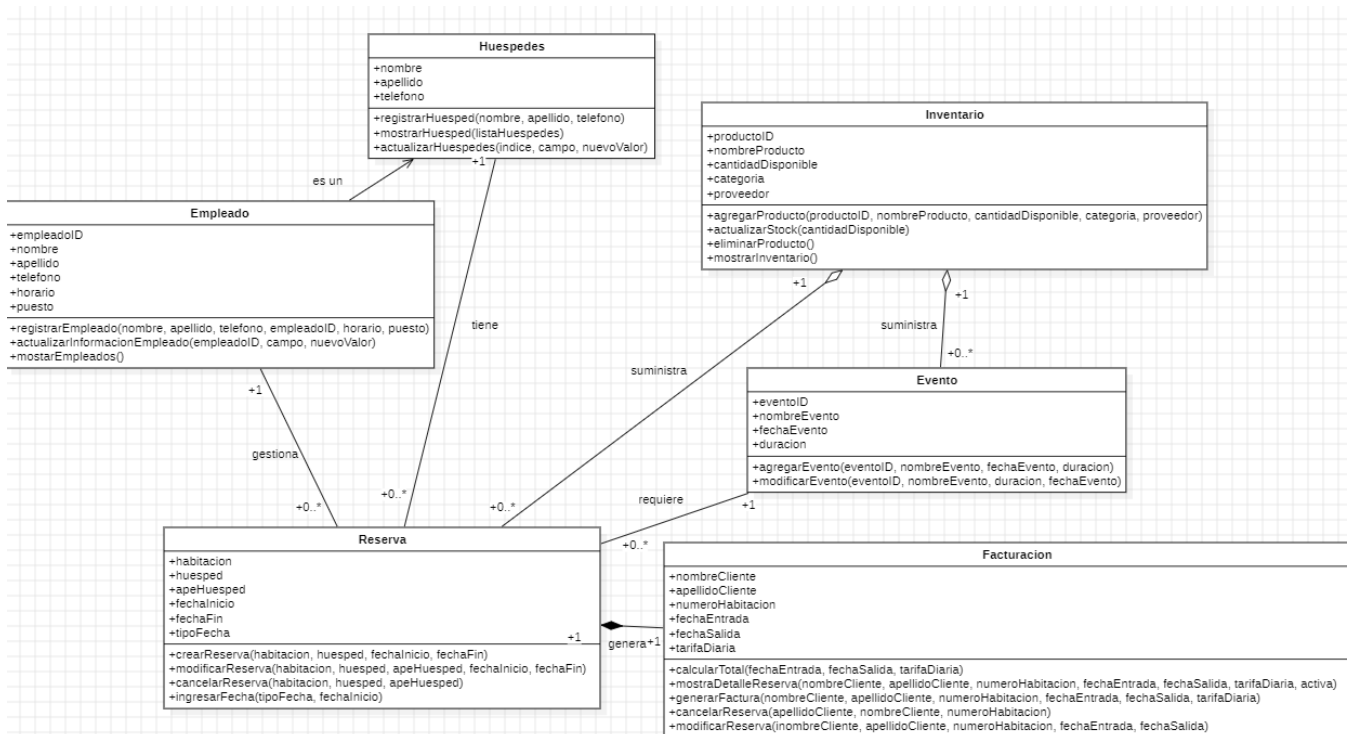
## **2. Enfoque del proyecto:**

- Estructura modular: El proyecto se organiza en torno a una clase principal (main) que engloba dos componentes principales:
  - Main Admin: Gestiona las funcionalidades y accesos para el personal administrativo del hotel.
  - Main Usuario: Maneja las interacciones y servicios para los huéspedes o clientes del hotel.
- Programación Orientada a Objetos: Utilizar clases y objetos para representar entidades como habitaciones, reservas, clientes y servicios del hotel.
- Separación de responsabilidades: Dividir claramente las funcionalidades administrativas de las interacciones con el usuario para mejorar la organización del código y la seguridad.
- Interfaz de usuario: Desarrollar interfaces separadas para administradores y usuarios, adaptadas a sus necesidades específicas.
- Gestión de datos: Implementar un sistema para almacenar y recuperar información sobre reservas, clientes y servicios del hotel.
- Flujo de trabajo de reservas: Crear un proceso intuitivo para que los usuarios puedan buscar disponibilidad, hacer reservas y gestionar sus estancias.

- Funcionalidades administrativas: Incluir herramientas para que el personal pueda gestionar habitaciones, ver ocupación, manejar check-ins y check-outs, y generar informes.

Este enfoque permite crear un sistema completo que atiende tanto las necesidades operativas del hotel como las experiencias de los huéspedes, todo ello aprovechando las ventajas de la programación orientada a objetos en Python.

### 3. Diagrama de Clases



### 3.1 Clases y sus relaciones

#### 3.1.1 Clase Huéspedes

#### 3.1.2 Clase Empleado

- **Atributos:**
  - empleadoID, nombre, apellido, telefono, horario, puesto
- **Métodos:**
  - Métodos para registrar, modificar y actualizar empleados.
- **Relaciones:**
  - Está relacionado con la clase **Huéspedes** (los empleados "atienden" a los huéspedes).
  - También está relacionado con la clase **Reserva**. Esta relación sugiere que los empleados están a cargo de gestionar o crear reservas.

#### 3.1.3 Clase Reserva

- **Atributos:**
  - habitacion, huesped, fechaInicio, fechaFinal, especificacion
- **Métodos:**
  - Métodos para ingresar, consultar y modificar reservas.
- **Relaciones:**
  - Está directamente vinculada con la clase **Huéspedes**, ya que las reservas son realizadas por huéspedes.
  - También tiene una relación con **Empleado**, lo que sugiere que los empleados son los que procesan o gestionan las reservas.
  - Además, genera una interacción con la clase **Facturación**, ya que una reserva genera una factura.

### 3.1.4 Clase Facturación

- **Atributos:**
  - nombreCliente, numeroReserva, fechaEntrada, fechaSalida, habitacion, tarifaTotal, estadoPago
- **Métodos:**
  - Métodos para crear y modificar facturas, así como cambiar el estado de pago.
- **Relaciones:**
  - Está asociada con la clase **Reserva**, ya que se genera una factura por cada reserva realizada.
  - También está vinculada al proceso de gestión de **Huéspedes**, dado que la facturación incluye los detalles del cliente (huésped) y su reserva.

### 3.1.5 Clase Inventario

- **Atributos:**
  - productoID, nombreProducto, cantidadDisponible, categoria, proveedor
- **Métodos:**
  - Métodos para actualizar y gestionar el inventario del hotel.
- **Relaciones:**
  - Suministra productos a los **Eventos**, que requieren el uso de materiales o suministros del inventario.

### 3.1.6 Clase Evento

- **Atributos:**
  - eventoID, nombreEvento, fechaInicio, fechaFin, duracion
- **Métodos:**
  - Métodos para registrar y modificar los eventos realizados en el hotel.



- **Relaciones:**

- Tiene una relación con el **Inventario**, ya que un evento puede requerir productos del inventario (suministros, comida, etc.).

### 3.2 Relaciones Clave:

- **Empleado y Huéspedes:** Los empleados están encargados de gestionar las reservas y atender a los huéspedes.
- **Reserva y Facturación:** Cada reserva está relacionada con una factura que contiene los detalles financieros del huésped y su estancia.
- **Inventario y Evento:** Los eventos requieren productos del inventario del hotel, creando una dependencia entre ambas clases.

### 3.3 Notas Adicionales:

- El diagrama también muestra las multiplicidades de las relaciones. Por ejemplo, la relación entre **Huésped** y **Reserva** es de "1 a muchos", lo que significa que un huésped puede realizar múltiples reservas.
- La relación entre **Evento** e **Inventario** es de "muchos a muchos", indicando que múltiples eventos pueden requerir múltiples productos del inventario.
- Este diagrama ilustra las principales funcionalidades de un sistema de gestión hotelera, modelando las relaciones entre los huéspedes, empleados, reservas, inventario y otros aspectos clave del hotel.
- Capturas del código fuente organizado y jerárquico empezando por la clase padre y/o de mayor relevancia.

## 4. Documentación de clases principales del sistema Hotelero

### 4.1 Clase Huéspedes

```
1 class Huespedes:
2     def __init__(self, nombre, apellido, telefono):
3         self._nombre = nombre
4         self._apellido = apellido
5         self._telefono = telefono
6         self.lista_huespedes = []
7
```

#### 4.1.1 Descripción

La clase Huéspedes es la base para la gestión de huéspedes del hotel. Implementa un sistema de registro y administración de la información básica de los huéspedes.

#### 4.1.2 Atributos

- `_nombre`: Nombre del huésped
- `_apellido`: Apellido del huésped
- `_telefono`: Número telefónico del huésped
- `lista_huespedes`: Lista que almacena todos los huéspedes registrados

#### 4.1.3 Métodos principales

##### 1. registrar\_huesped(nombre, apellido, telefono)

```
1 def registrar_huesped(self, nombre, apellido, telefono):
2     nuevo_huesped = Huespedes(nombre, apellido, telefono)
3     self.lista_huespedes.append(nuevo_huesped)
4     print(f"Huésped registrado: {nuevo_huesped.nombre} {nuevo_huesped.apellido}, Teléfono: {nuevo_huesped.telefono}")
```

- Registra un nuevo huésped en el sistema
- Almacena la información en la lista\_huespedes
- Imprime confirmación del registro

## 2. mostrar\_huespedes()

```
1 def mostrar_empleados(self):
2     if len(self.lista_empleados) > 0:
3         print("\nEmpleados registrados:")
4         for i, empleado in enumerate(self.lista_empleados, 1):
5             print(f"{i}. {empleado.nombre} {empleado.apellido}")
6             print(f"    ID: {empleado.IdEmpleado}")
7             print(f"    Teléfono: {empleado.telefono}")
8             print(f"    Puesto: {empleado.puesto}")
9             print(f"    Horario: {empleado.Horario}")
10        return True
11    else:
12        print("No hay empleados registrados.")
13        return False
```

- Muestra todos los huéspedes registrados
- Retorna True si hay huéspedes, False si no hay

### 3. actualizar\_huesped(indice, campo, nuevo\_valor)

```
1  def actualizar_informacion_empleado(self, IdEmpleado, campo, nuevo_valor):
2      try:
3          empleado = next((e for e in self.lista_empleados if e.IdEmpleado == IdEmpleado), None)
4          if empleado:
5              if campo == 1:
6                  empleado._nombre = nuevo_valor
7              elif campo == 2:
8                  empleado._apellido = nuevo_valor
9              elif campo == 3:
10                 empleado._telefono = nuevo_valor
11             elif campo == 4:
12                 empleado._puesto = nuevo_valor
13
14                 print(f"\nEmpleado actualizado:")
15                 print(f"Nombre: {empleado.nombre} {empleado.apellido}")
16                 print(f"ID: {empleado.IdEmpleado}")
17                 print(f"Teléfono: {empleado.telefono}")
18                 print(f"Puesto: {empleado.puesto}")
19                 return True
20             else:
21                 print("\nEmpleado no encontrado")
22                 return False
23         except Exception as e:
24             print(f"\nError al actualizar empleado: {str(e)}")
25             return False
```

- Actualiza la información de un huésped específico
- Campos actualizables: nombre, apellido, teléfono

## 4.2 Clase Empleado (Hereda de Huespedes)

```
1 class Empleado(Huespedes):
2     def __init__(self, nombre, apellido, telefono, IdEmpleado, Horario, puesto):
3         super().__init__(nombre, apellido, telefono)
4         self._IdEmpleado = IdEmpleado
5         self._puesto = puesto
6         self._Horario = Horario
7         self.lista_empleados = []
```

### 4.2.1 Descripción

La clase Empleado hereda de Huespedes y agrega funcionalidades específicas para la gestión del personal del hotel.

### 4.2.2 Atributos

- `_IdEmpleado`: Identificador único del empleado
- `_puesto`: Cargo o posición del empleado
- `_Horario`: Horario de trabajo asignado
- `lista_empleados`: Lista que almacena todos los empleados

### 4.2.3 Métodos Principales

#### 1. registrar\_empleado(nombre, apellido, telefono, IdEmpleado, Horario, puesto)

```
1 def registrar_empleado(self, nombre, apellido, telefono, IdEmpleado, Horario, puesto):
2     nuevo_empleado = Empleado(nombre, apellido, telefono, IdEmpleado, Horario, puesto)
3     self.lista_empleados.append(nuevo_empleado)
4     print(f"El Empleado registrado es {nuevo_empleado.nombre} {nuevo_empleado.apellido},
5           Teléfono: {nuevo_empleado.telefono}, ID= {nuevo_empleado.IdEmpleado}")
```

- Registra un nuevo empleado con toda su información laboral
- Almacena en lista\_empleados
- Imprime confirmación

#### 2. mostrar\_empleados()

```
1 def mostrar_empleados(self):
2     if len(self.lista_empleados) > 0:
3         print("\nEmpleados registrados:")
4         for i, empleado in enumerate(self.lista_empleados, 1):
5             print(f"{i}. {empleado.nombre} {empleado.apellido}")
6             print(f"    ID: {empleado.IdEmpleado}")
7             print(f"    Teléfono: {empleado.telefono}")
8             print(f"    Puesto: {empleado.puesto}")
9             print(f"    Horario: {empleado.Horario}")
10        return True
11    else:
12        print("No hay empleados registrados.")
13        return False
```

- Muestra la lista completa de empleados con sus detalles
- Incluye ID, puesto y horario

### 3. actualizar\_informacion\_empleado(IdEmpleado, campo, nuevo\_valor)

```
1 def actualizar_informacion_empleado(self, IdEmpleado, campo, nuevo_valor):
2     try:
3         empleado = next((e for e in self.lista_empleados if e.IdEmpleado == IdEmpleado), None)
4         if empleado:
5             if campo == 1:
6                 empleado._nombre = nuevo_valor
7             elif campo == 2:
8                 empleado._apellido = nuevo_valor
9             elif campo == 3:
10                 empleado._telefono = nuevo_valor
11             elif campo == 4:
12                 empleado._puesto = nuevo_valor
13
14             print(f"\nEmpleado actualizado:")
15             print(f"Nombre: {empleado.nombre} {empleado.apellido}")
16             print(f"ID: {empleado.IdEmpleado}")
17             print(f"Teléfono: {empleado.telefono}")
18             print(f"Puesto: {empleado.puesto}")
19             return True
20         else:
21             print("\nEmpleado no encontrado")
22             return False
23     except Exception as e:
24         print(f"\nError al actualizar empleado: {str(e)}")
25         return False
```

- Actualiza información específica del empleado
- Permite modificar datos personales y laborales

## 4.3 Clase Inventario

```
1 class Inventario:
2     def __init__(self, productoID, nombreProducto, cantidadDisponible, categoria, proveedor):
3         self._productoID = productoID
4         self._nombreProducto = nombreProducto
5         self._cantidadDisponible = cantidadDisponible
6         self._categoria = categoria
7         self._proveedor = proveedor
8         self.listaProductos = [
9             {"productoID": 1, "nombreProducto": "Toalla", "cantidadDisponible": 100, "categoria": "Baño", "proveedor": "TextilesHotel"},
10            {"productoID": 2, "nombreProducto": "Almohada", "cantidadDisponible": 50, "categoria": "Habitación", "proveedor": "SueñosFelices"},
11            {"productoID": 3, "nombreProducto": "Jabón", "cantidadDisponible": 200, "categoria": "Amenidades", "proveedor": "LimpiezaTotal"},
12            {"productoID": 4, "nombreProducto": "Secador de pelo", "cantidadDisponible": 30, "categoria": "Electrónica", "proveedor": "ElectroHotel"},
13            {"productoID": 5, "nombreProducto": "Minibar", "cantidadDisponible": 20, "categoria": "Alimentación", "proveedor": "BebidasFrescas"}
14        ]
15        self.reservas = [] # Agregación con Reservas
16        self.eventos = [] # Agregación con Eventos
```

### 4.3.1 Descripción

La clase Inventario gestiona el stock de productos y suministros del hotel.

### 4.3.2 Atributos

- `_productoID`: Identificador único del producto
- `_nombreProducto`: Nombre del producto
- `_cantidadDisponible`: Cantidad en stock
- `_categoria`: Categoría del producto
- `_proveedor`: Proveedor del producto
- `listaProductos`: Lista con productos predefinidos



### 4.3.3 Métodos principales

#### 1. agregarProducto(productoID, nombreProducto, cantidadDisponible, categoria, proveedor)

```
1 def agregarProducto(self, productoID, nombreProducto, cantidadDisponible, categoria, proveedor):
2
3     print(f"El producto agregado es {productoID}:{nombreProducto}, la cantidad es
4           {cantidadDisponible}")
5     self.listaProductos.append({
6         "productoID": productoID,
7         "nombreProducto": nombreProducto,
8         "cantidadDisponible": cantidadDisponible,
9         "categoria": categoria,
10        "proveedor": proveedor
11    })
```

- Agrega nuevo producto al inventario
- Actualiza la listaProductos

#### 2. actualizarStock(numero)

```
1 def actualizarStock(self, numero, productoID, nombreProducto):
2     self.productoID= productoID
3     self.nombreProducto = nombreProducto
4
5
6     print(f"El producto {self.productoID}:{self.nombreProducto} ha sido actualizado con
7           la cantidad de {numero}")
8     for producto in self.listaProductos:
9         if producto["productoID"] == self.productoID:
10            producto["cantidadDisponible"] = numero
11            break
```

- Actualiza la cantidad disponible de un producto
- Modifica el registro existente

### 3. eliminarProducto()

```
1 def eliminarProducto(self):
2     print(f"El producto eliminado es {self.productoID}:{self.nombreProducto}")
3     self.listaProductos = [producto for producto in self.listaProductos if producto
4                             ["productoID"] != self.productoID]
```

- Elimina un producto del inventario
- Actualiza la listaProductos

### 4. mostrar\_inventario()

```
1 def mostrar_inventario(self):
2     if not self.listaProductos:
3         print("El inventario está vacío.")
4     else:
5         print("\nInventario actual:")
6         for producto in self.listaProductos:
7             print(f"ID: {producto['productoID']}, Producto: {producto['nombreProducto']},
8                   Cantidad: {producto['cantidadDisponible']}, Categoría:
9                   {producto['categoria']}, Proveedor: {producto['proveedor']}")
10
```

- Muestra todos los productos en inventario
- Incluye detalles completos de cada producto

## 4.4 Clase Evento

```
1 class Evento:
2     def __init__(self, nombre_evento, fecha_evento, duracion, evento_ID):
3         self._fecha_evento = fecha_evento
4         self._duracion = duracion
5         self._nombre_evento = nombre_evento
6         self._evento_id = evento_ID
```

### 4.4.1 Descripción

La clase Evento maneja la programación y gestión de eventos en el hotel.

- `_fecha_evento`: Fecha programada del evento
- `_duracion`: Duración del evento
- `_nombre_evento`: Nombre o título del evento
- `_evento_id`: Identificador único del evento

### 4.4.2 Métodos Principales

#### 1. agregar\_evento()

```
1 def agregar_evento(self, evento_id, nombre_evento, duracion, fecha_evento):
2     self.evento_id = evento_id
3     self.nombre_evento = nombre_evento
4     self.duracion = duracion
5     self.fecha_evento = fecha_evento
6     print(f"El evento agregado es {self.evento_id}: {self.nombre_evento}
7         con una duración de {self.duracion} para la fecha {self.fecha_evento}.")
```

- Registra un nuevo evento
- Imprime confirmación

## 2. modificar\_evento()

```
1 def modificar_evento(self, evento_id, nombre_evento, duracion, fecha_evento):
2     self.evento_id = evento_id
3     self.nombre_evento = nombre_evento
4     self.duracion = duracion
5     self.fecha_evento = fecha_evento
6     print(f"El evento modificado ahora es {self.evento_id}: {self.nombre_evento}
7         con una duración de {self.duracion} para la fecha {self.fecha_evento}.")
```

- Permite modificar los detalles de un evento existente
- Actualiza la información

## 4.5 Clase Main (main\_admin.py)

```
1 class Main:
2     def __init__(self):
3         self.huespedes = Huespedes("", "", "")
4         self.empleado_manager = Empleado(0,0,0,0,0,0)
5         self.eventos_registrados = []
6         self.inventario = Inventario(0, "", 0, "", "")
7
8         self.gestor_huespedes = GestionHuespedes(self.huespedes)
9         self.gestor_empleados = GestionEmpleados(self.empleado_manager)
10        self.gestor_eventos = GestionEventos(self.eventos_registrados)
11        self.gestor_inventario = GestionInventario(self.inventario)
```

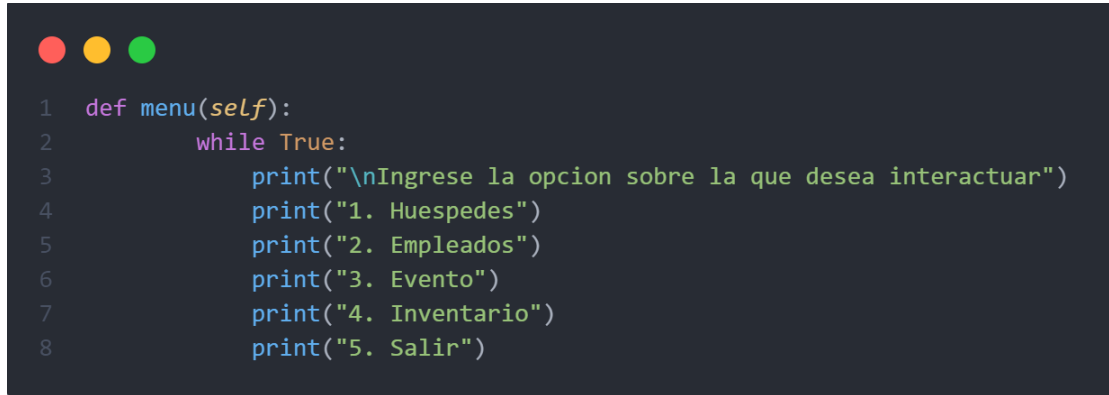
### 4.5.1 Descripción

La clase Main es el punto de entrada del sistema y coordina todas las operaciones.

### 4.5.2 Componentes Principales

- Inicializa todos los gestores
- Proporciona el menú principal
- Coordina la interacción entre módulos

### 4.5.3 Funcionalidad del Menú Principal

A terminal window with a dark background and three colored window control buttons (red, yellow, green) in the top-left corner. It displays a Python function named 'menu' that takes 'self' as an argument. Inside the function, there is a 'while True:' loop that prints a prompt and a list of five options: '1. Huespedes', '2. Empleados', '3. Evento', '4. Inventario', and '5. Salir'.

```
1 def menu(self):  
2     while True:  
3         print("\nIngrese la opcion sobre la que desea interactuar")  
4         print("1. Huespedes")  
5         print("2. Empleados")  
6         print("3. Evento")  
7         print("4. Inventario")  
8         print("5. Salir")
```

- Esta estructura permite:
  1. Navegación intuitiva entre módulos
  2. Acceso a todas las funcionalidades
  3. Gestión centralizada del sistema
  4. Salida controlada del programa

La clase Main actúa como controlador principal, gestionando la interacción entre todos los componentes del sistema y proporcionando una interfaz unificada para el usuario. </antArtifact>

Esta documentación proporciona una visión detallada de las clases principales del sistema, incluyendo sus atributos, métodos y relaciones. ¿Necesitas que profundice en algún aspecto específico de alguna de las clases?

## 5. Clase Main (Main\_usuario.py)

La clase Main es el punto de entrada del programa y maneja la interfaz de usuario principal.

### Paso 1: Inicialización

```
class Main:
    def __init__(self):
        self.reservas = []
```

- Se crea una lista vacía para almacenar las reservas.

### Paso 2: Mostrar el menú principal

```
def mostrar_menu(self):
    print("HOTEL MANABITA")
    print("1. Crear nueva reserva")
    print("2. Modificar Reserva")
    print("3. Cancelar Reserva")
    print("4. Detalles de mis reservas")
    print("5. Salir")
```

- Muestra las opciones disponibles para el usuario.

### Paso 3: Crear una nueva reserva

```

def crear_reserva(self):
    verificador = VerificacionDatos()

    habitacion_disponible = False
    while not habitacion_disponible:
        habitacion = verificador.verificar_entero("Ingrese el número de su habitación: ")
        if VerificadorReservas.habitacion_disponible(habitacion, self.reservas):
            habitacion_disponible = True
        else:
            print("La habitación ya está reservada por otro usuario. Por favor, elija otra habitación.")

    nombre = verificador.verificar_cadena("Ingrese su nombre: ")
    apellido = verificador.verificar_cadena("Ingrese su apellido: ")
    validador = VerificadorFecha()
    fecha_entrada = validador.solicitar_fecha("Ingrese la fecha de entrada")
    fecha_salida = validador.solicitar_fecha("Ingrese la fecha de salida")

    print("-----")
    reserva = Facturacion(nombre, apellido, habitacion, fecha_entrada, fecha_salida, tarifa_diaria=20)
    total = reserva.calcular_total()
    print(f"Total calculado: ${total:.2f}")
    print(reserva.generar_factura())
    self.reservas.append(reserva)

```

- Solicita y valida los datos de la reserva.
- Crea una nueva instancia de Facturacion y la añade a la lista de reservas.

#### Paso 4: Modificar una reserva existente

```

def modificar_reserva(self):
    numHabitacion = int(input("Ingrese el número de habitación reservada que desea modificar: "))
    reserva_encontrada = False
    for reserva in self.reservas:
        if reserva.numero_habitacion == numHabitacion and reserva.activa:
            resultado = reserva.modificar_reserva()
            print(resultado)
            reserva_encontrada = True
            break
    if not reserva_encontrada:
        print("Reserva no encontrada o ya cancelada.")

```

- Busca una reserva por número de habitación y permite modificarla.

#### Paso 5: Cancelar una reserva

```
def cancelar_reserva(self):
    numHabitacion = int(input("Ingrese el número de habitación de la reserva que desea cancelar: "))
    reserva_encontrada = False
    for reserva in self.reservas:
        if reserva.numero_habitacion == numHabitacion and reserva.activa:
            resultado = reserva.cancelar_reserva()
            print(resultado)
            reserva_encontrada = True
            break
    if not reserva_encontrada:
        print("Reserva no encontrada o ya cancelada.")
```

- Busca una reserva por número de habitación y la cancela si está activa.

#### Paso 6: Mostrar detalles de las reservas

```
def mostrar_detalle_reservas(self):
    if self.reservas:
        for i, reserva in enumerate(self.reservas, 1):
            if reserva.activa:
                print(f"\nReserva {i}:")
                print(reserva.mostrar_detalle_reserva())
    else:
        print("No hay reservas registradas.")
```

- Muestra los detalles de todas las reservas activas.

#### Paso 7: Ejecutar el programa principal



```

def ejecutar(self):
    op = None
    while op != 5:
        self.mostrar_menu()
        op = int(input("Escoja una opción: "))

        if op == 1:
            self.crear_reserva()
        elif op == 2:
            self.modificar_reserva()
        elif op == 3:
            self.cancelar_reserva()
        elif op == 4:
            self.mostrar_detalle_reservas()
        elif op == 5:
            print("Gracias por usar nuestro sistema de reservas. ¡Hasta pronto!")
        else:
            print("Opción inválida. Por favor, intente de nuevo.")

if __name__ == "__main__":
    main = Main()
    main.ejecutar()

```

- Implementa el bucle principal del programa, manejando las opciones del menú.

## 6. Clase Facturacion (Facturacion.py)

La clase Facturacion maneja los aspectos financieros y de facturación de una reserva.

### Paso 1: Inicialización

```

class Facturacion:
    def __init__(self, nombre_cliente, apellido_cliente, numero_habitacion, fecha_entrada, fecha_salida, tarifa_diaria):
        self.nombre_cliente = nombre_cliente
        self.apellido_cliente = apellido_cliente
        self.numero_habitacion = numero_habitacion
        self.fecha_entrada = fecha_entrada
        self.fecha_salida = fecha_salida
        self.tarifa_diaria = tarifa_diaria
        self.activa = True
        self.reserva = Reservas(numero_habitacion, nombre_cliente, apellido_cliente, fecha_entrada, fecha_salida)

```

- Inicializa los atributos de la facturación y crea una instancia de Reservas.

### Paso 2: Métodos getter y setter

```
#Getters
```

```
def get_nombre_cliente(self):  
    return self._nombre_cliente
```

```
def get_apellido_cliente(self):  
    return self._apellido_cliente
```

```
def get_numero_habitacion(self):  
    return self._numero_habitacion
```

```
def get_fecha_entrada(self):  
    return self._fecha_entrada
```

```
def get_fecha_salida(self):  
    return self._fecha_salida
```

```
def get_tarifa_diaria(self):  
    return self._tarifa_diaria
```

```
def is_activa(self):  
    return self._activa
```

```

# Setters
def set_nombre_cliente(self, nombre_cliente):
    self._nombre_cliente = nombre_cliente

def set_apellido_cliente(self, apellido_cliente):
    self._apellido_cliente = apellido_cliente

def set_numero_habitacion(self, numero_habitacion):
    self._numero_habitacion = numero_habitacion

def set_fecha_entrada(self, fecha_entrada):
    self._fecha_entrada = fecha_entrada

def set_fecha_salida(self, fecha_salida):
    self._fecha_salida = fecha_salida

def set_tarifa_diaria(self, tarifa_diaria):
    self._tarifa_diaria = tarifa_diaria

def set_activa(self, estado):
    self._activa = estado

```

- Implementa métodos para acceder y modificar los atributos de la clase.

### Paso 3: Calcular el total de la reserva

```

def calcular_total(self, fecha_entrada, fecha_salida, tarifa_diaria):
    dias = (fecha_salida - fecha_entrada).days
    return dias * tarifa_diaria

```

- Calcula el costo total de la estancia.

### Paso 4: Mostrar detalles de la reserva

```
def mostrar_detalle_reserva(self, nombre_cliente, apellido_cliente, numero_habitacion,
                             fecha_entrada, fecha_salida, tarifa_diaria, activa):
    total = self.calcular_total(fecha_entrada, fecha_salida, tarifa_diaria)
    detalle = f"""
    Detalle de la Reserva:
    -----
    Cliente: {nombre_cliente} {apellido_cliente}
    Habitación: {numero_habitacion}
    Fecha de Entrada: {fecha_entrada.strftime('%d/%m/%Y')}
    Fecha de Salida: {fecha_salida.strftime('%d/%m/%Y')}
    Tarifa Diaria: ${tarifa_diaria:.2f}
    Total a Pagar: ${total:.2f}
    Estado: {'Activa' if activa else 'Cancelada'}
    """
    return detalle
```

- Genera un resumen detallado de la reserva.

#### Paso 5: Generar factura

```
def generar_factura(self, nombre_cliente, apellido_cliente, numero_habitacion,
                    fecha_entrada, fecha_salida, tarifa_diaria):
    total = self.calcular_total(fecha_entrada, fecha_salida, tarifa_diaria)
    iva = total * 0.15
    total_con_iva = total + iva
    factura = f"""
    FACTURA
    -----
    Cliente: {nombre_cliente} {apellido_cliente}
    Detalle:
    - Alojamiento en habitación {numero_habitacion}
    - Desde: {fecha_entrada.strftime('%d/%m/%Y')}
    - Hasta: {fecha_salida.strftime('%d/%m/%Y')}

    Subtotal: ${total:.2f}
    IVA (15%): ${iva:.2f}
    Total: ${total_con_iva:.2f}
    """
    return factura
```

- Crea una factura completa incluyendo IVA.

#### Paso 6: Cancelar reserva

```
def cancelar_reserva(self, apellido_cliente, nombre_cliente, numero_habitacion):
    self.activa = False
    return f"La reserva para {apellido_cliente}, {nombre_cliente} en la habitación {numero_habitacion} ha sido cancelada."
```

- Cancela la reserva cambiando su estado a inactivo.

## Paso 7: Modificar reserva

```
def modificar_reserva(self, nombre_cliente, apellido_cliente, numero_habitacion,
                     fecha_entrada, fecha_salida):
    print("¿Qué desea modificar?")
    print("1. Nombre del cliente")
    print("2. Apellido del cliente")
    print("3. Número de habitación")
    print("4. Fecha de entrada")
    print("5. Fecha de salida")
    print("6. Volver al menú principal")

    op_mod = int(input("Escoja una opción: "))

    if op_mod == 1:
        nombre_cliente = input("Ingrese el nuevo nombre: ")
    elif op_mod == 2:
        apellido_cliente = input("Ingrese el nuevo apellido: ")
    elif op_mod == 3:
        numero_habitacion = int(input("Ingrese el nuevo número de habitación: "))
    elif op_mod == 4:
        nueva_fecha = input("Ingrese la nueva fecha de entrada (DD/MM/YYYY): ")
        fecha_entrada = datetime.strptime(nueva_fecha, "%d/%m/%Y")
    elif op_mod == 5:
        nueva_fecha = input("Ingrese la nueva fecha de salida (DD/MM/YYYY): ")
        fecha_salida = datetime.strptime(nueva_fecha, "%d/%m/%Y")
    elif op_mod == 6:
        return "Volviendo al menu principal..."
    else:
        return "Opcion invalida."

    self.nombre_cliente = nombre_cliente
    self.apellido_cliente = apellido_cliente
    self.numero_habitacion = numero_habitacion
    self.fecha_entrada = fecha_entrada
    self.fecha_salida = fecha_salida

    return "Reserva modificada exitosamente."
```

- Permite modificar diferentes aspectos de la reserva.

## 7. Clase Reservas (Reservas.py)

La clase Reservas maneja la información básica de una reserva.

### Paso 1: Inicialización

```
class Reservas():
    def __init__(self, habitacion, huesped, apeHuesped, fecha_inicio, fecha_fin):
        self._habitacion = habitacion
        self._huesped = huesped
        self._apeHuesped = apeHuesped
        self._fecha_inicio = fecha_inicio
        self._fecha_fin = fecha_fin
        self._activa = True
```

- Inicializa los atributos de la reserva.

## Paso 2: Métodos getter y setter

```
# Getters
def get_habitacion(self):
    return self._habitacion

def get_huesped(self):
    return self._huesped

def get_apeHuesped(self):
    return self._apeHuesped

def get_fecha_inicio(self):
    return self._fecha_inicio

def get_fecha_fin(self):
    return self._fecha_fin

def is_activa(self):
    return self._activa
```

```
# Setters
def set_habitacion(self, habitacion):
    self._habitacion = habitacion

def set_huesped(self, huesped):
    self._huesped = huesped

def set_apeHuesped(self, apeHuesped):
    self._apeHuesped = apeHuesped

def set_fecha_inicio(self, fecha_inicio):
    self._fecha_inicio = fecha_inicio

def set_fecha_fin(self, fecha_fin):
    self._fecha_fin = fecha_fin

def set_activa(self, activa):
    self._activa = activa
```

- Implementa métodos para acceder y modificar los atributos de la clase.

### Paso 3: Crear reserva

```
def crear_reserva(self, habitacion, huesped, apeHuesped, fecha_inicio, fecha_fin):
    print(f"Número de habitación: {habitacion}")
    print(f"Huésped: {apeHuesped}, {huesped}")
    print(f"Fecha de entrada: {fecha_inicio.strftime('%d/%m/%Y')}")
    print(f"Fecha de salida: {fecha_fin.strftime('%d/%m/%Y')}")
```

- Muestra los detalles de una nueva reserva.

## Paso 4: Modificar reserva

```
def modificar_reserva(self, habitacion, huesped, apeHuesped, fecha_inicio, fecha_fin):
    print("¿Qué desea modificar?")
    print("1. Nombre del huésped")
    print("2. Apellido del huésped")
    print("3. Número de habitación")
    print("4. Fecha de entrada")
    print("5. Fecha de salida")
    print("6. Menú principal")
    op_mod = None
    while op_mod != 6:
        op_mod = int(input("Escriba una opción: "))
        if op_mod == 1:
            huesped = input("Ingrese el nuevo nombre: ")
            self.set_huesped(huesped)
            print("Nombre del huésped actualizado con éxito.")
        elif op_mod == 2:
            apeHuesped = input("Ingrese el nuevo apellido: ")
            self.set_apeHuesped(apeHuesped)
            print("Apellido del huésped actualizado con éxito.")
        elif op_mod == 3:
            habitacion = int(input("Ingrese el nuevo número de habitación: "))
            self.set_habitacion(habitacion)
            print("Número de habitación actualizado con éxito.")
        elif op_mod == 4:
            fecha_inicio = ingresar_fecha("entrada")
            fecha_fin = ingresar_fecha("salida", fecha_inicio)
            self.set_fecha_inicio(fecha_inicio)
            self.set_fecha_fin(fecha_fin)
            print("Fechas actualizadas.")
        elif op_mod == 5:
            fecha_fin = ingresar_fecha("salida", self.get_fecha_inicio())
            self.set_fecha_fin(fecha_fin)
            print("Fecha de salida actualizada.")
        elif op_mod == 6:
            print("Regresando al menú principal...")
            break
        else:
            print("Opción inválida, por favor intente de nuevo.")
    self.crear_reserva(self.get_habitacion(), self.get_huesped(), self.get_apeHuesped(), self.get_fecha_inicio(), self.get_fecha_fin())
```

- Permite modificar diferentes aspectos de la reserva.

## Paso 5: Cancelar reserva

```
def cancelar_reserva(self, habitacion, huesped, apeHuesped):
    self.set_activa(False)
    print(f"La reserva para {apeHuesped}, {huesped} en la habitación {habitacion} ha sido cancelada.")
```

- Cancela la reserva cambiando su estado a inactivo.



## 8. Clases de Verificación (verificaciones.py)

### Clase VerificadorReservas

```
class VerificadorReservas():  
    @staticmethod  
    def habitacion_disponible(habitacion, reservas):  
        for reserva in reservas:  
            if reserva.numero_habitacion == habitacion and reserva.activa:  
                return False  
        return True
```

- Verifica si una habitación está disponible.

### 8.1 Clase VerificadorFecha

```
class VerificadorFecha:  
    def __init__(self):  
        pass  
  
    def solicitar_fecha(self, mensaje):  
        while True:  
            fecha_str = input(mensaje + " (formato: DD/MM/YYYY): ")  
            try:  
                fecha = datetime.strptime(fecha_str, "%d/%m/%Y")  
                if fecha < datetime.now():  
                    print("La fecha no puede ser en el pasado. Inténtalo de nuevo.")  
            except ValueError:  
                print("Formato de fecha inválido. Inténtalo de nuevo.")  
            else:  
                return fecha
```

- Solicita y valida una fecha ingresada por el usuario.

## 8.2 Clase VerificacionDatos

- Valida entradas numéricas y de texto.

```
class VerificacionDatos:

    def verificar_entero(self, mensaje):
        while True:
            valor = input(mensaje)
            try:
                valor_entero = int(valor)
                return valor_entero
            except ValueError:
                print("Por favor, ingrese un número válido.")

    def verificar_cadena(self, mensaje):
        while True:
            valor = input(mensaje)
            if valor.isalpha():
                return valor
            else:
                print("Por favor, ingrese solo letras.")
```

## 8.3 Sistema Principal - Hotel Manabita (Main.py)

### 8.3.1 Descripción

La clase MainSystem es el punto de entrada principal del sistema del Hotel Manabita que coordina la navegación entre los subsistemas de usuario y administrador. Actúa como un hub central que permite acceder a las diferentes funcionalidades del sistema según el tipo de usuario.

### 8.3.2 Componentes Principales

- **Gestores inicializados**
  - **Componentes PrincipalesGestor de Usuario (user\_main):** Maneja todas las operaciones relacionadas con el sistema de usuario
  - **Gestor de Administrador (admin\_main):** Maneja todas las operaciones relacionadas con el sistema administrativo
- **Funcionalidades base**
  - **Sistema de menús interactivo**
  - **Control de acceso a subsistemas**
  - **Gestión de salida del programa**

### 8.3.3 Funcionalidad del Menú Principal

Opciones Disponibles:

#### 1. Acceso Usuario

- Redirecciona al sistema de usuario
- Ejecuta user\_main.ejecutar()
- Permite acceso a funcionalidades de cliente

#### 2. Acceso Administrador

- Redirecciona al sistema administrativo
- Ejecuta admin\_main.menu()
- Permite acceso a funcionalidades de gestión

#### 3. Salir

- Finaliza la ejecución del programa
- Muestra mensaje de despedida
- Cierre controlado del sistema

Esta estructura permite:

#### **1. Navegación intuitiva entre módulos**

- Menú claro y conciso
- Opciones numeradas
- Retroalimentación inmediata de las acciones

#### **2. Acceso a todas las funcionalidades**

- Separación clara entre sistema usuario y administrador
- Acceso directo a cada subsistema
- Estructura modular y escalable


#### **3. Gestión centralizada del sistema**

- Control unificado de accesos
- Manejo centralizado de la navegación
- Punto único de entrada al sistema

#### **4. Salida controlada del programa**

- Opción específica para salir
- Mensaje de confirmación
- Cierre seguro del sistema

### 8.3.4 Aspectos técnicos



```
1 import sys
2 from vistaUsuario.Main_usuario import Main as UserMain
3 from admin.main_admin import Main as AdminMain
4
```

- **Ciclo de vida**

1. Inicialización de componentes en el constructor
2. Ejecución del bucle principal
3. Gestión de entrada del usuario
4. Redirección a subsistemas
5. Control de salida

La clase MainSystem actúa como controlador principal, gestionando la interacción entre todos los componentes del sistema y proporcionando una interfaz unificada para el usuario. Su diseño modular permite una fácil extensión y mantenimiento del sistema, mientras mantiene una separación clara entre las diferentes áreas funcionales del Hotel Manabita.

## 9. Conclusión

El Sistema de Gestión Hotel Manabita representa una solución integral que cumple efectivamente con los objetivos planteados inicialmente. La implementación demuestra un diseño maduro que:

- Centraliza la gestión hotelera de manera efectiva

- Proporciona interfaces específicas para diferentes tipos de usuarios
- Implementa verificaciones y validaciones robustas
- Mantiene una clara separación de responsabilidades

El uso de POO ha permitido crear un sistema modular, mantenible y escalable, que puede adaptarse fácilmente a futuras necesidades del hotel. La arquitectura elegida facilita tanto la gestión administrativa como la experiencia del usuario final, cumpliendo así con los requerimientos de un sistema hotelero moderno.

## **10. Análisis y Conclusión - Sistema de Gestión Hotel Manabita**

### **10.1 Análisis**

El sistema desarrollado demuestra una arquitectura robusta y bien estructurada que sigue los principios de la Programación Orientada a Objetos (POO). Los aspectos más destacables son:

#### **10.1.1 Arquitectura Modular**

- División clara entre sistema administrativo y de usuario
- Separación efectiva de responsabilidades
- Fácil mantenimiento y escalabilidad

#### **10.1.2 Jerarquía de Clases**

- Herencia bien implementada (ej: Empleado hereda de Huéspedes)
- Relaciones coherentes entre clases (ej: Reserva-Facturación)
- Encapsulamiento adecuado de datos

#### **10.1.3 Gestión de Datos**

- Sistema robusto de verificaciones
- Validación de entrada de datos
- Control de estados en reservas