Título de la práctica: Unidad 1 Paradigma Orientado a objetos (Examen)		
Asignatura: Optativa	**Fecha de inicio: 04 de octubre 25**	**Hoja:** 1 **de 2**
Unidad temática: Unidad I	**Fecha de entrega: 10 de octubre 25**	**Grupo:**
No. de participantes recomendados: 1 integrante	**Elaboró:** ISC Adriana Y. Contreras Álvarez	
Lugar Asignado: Laboratorio	**Revisó:**	
Alumno:	**López Guerrero Andrea Sarahi**	

Caso: Sistema de Reservaciones para Hotel

La empresa “Hotel Le Villa” desea implementar un sistema informático que permita gestionar de manera eficiente las reservaciones, habitaciones, clientes, servicios adicionales y pagos del establecimiento. El sistema debe representar a las distintas entidades que forman parte de las operaciones del hotel, considerando que cada una cumple un rol diferente dentro del proceso de administración y atención al cliente. Además, se requiere llevar un registro de fechas importantes, como la fecha de ingreso y salida de cada cliente, la fecha de realización de una reserva o la fecha de pago.

El equipo de desarrollo ha establecido que el sistema debe cumplir con las siguientes condiciones:

- Todas las habitaciones comparten atributos básicos, como el número de habitación, tipo (sencilla, doble, suite) y estado (disponible, ocupada, en mantenimiento).
- Los clientes poseen información particular, como su nombre, número de identificación, número de contacto y el historial de reservaciones realizadas.
- Cada reserva debe incluir los datos del cliente asociado, la habitación asignada, las fechas de entrada y salida, así como los servicios adicionales solicitados.
- Los servicios adicionales (como restaurante, spa o transporte) deben poder registrarse y asociarse a una reserva específica.
- Los pagos deben reflejar el monto total, método de pago y fecha de transacción.
- Todas las entidades deben poder mostrar su información en un formato legible y estructurado.

Descripción general

El presente proyecto tiene como propósito desarrollar un sistema de reservaciones para un hotel, que permita automatizar y optimizar la gestión de habitaciones, clientes, servicios y pagos.

Objetivos

Objetivo general:
Automatizar la gestión de reservaciones del hotel para mejorar la eficiencia operativa y la satisfacción del cliente.

Objetivos específicos:

- Optimizar la asignación y disponibilidad de habitaciones en tiempo real.
- Registrar y administrar los datos de los clientes y sus reservaciones.
- Controlar los servicios adicionales y pagos asociados a cada estancia.
- Reducir errores manuales y aumentar la trazabilidad de las operaciones.
- Proveer reportes claros sobre ocupación, ingresos y actividad del hotel.

Actores del Sistema

Actores principales

- **Cliente:**
  Es el usuario que realiza reservaciones en el hotel. Puede consultar disponibilidad de habitaciones, registrar una reserva, solicitar servicios adicionales y efectuar pagos por su estancia.
- **Recepcionista:**
  Es el encargado de gestionar las operaciones diarias del sistema. Registra nuevas reservaciones, verifica el estado de las habitaciones, gestiona el check-in y check-out de los clientes, y actualiza la información de pagos o servicios solicitados.

Actores secundarios

- **Botones de servicio (personal de apoyo):**
  Representan al personal operativo encargado de atender las solicitudes del cliente dentro del hotel (por ejemplo, servicio a la habitación, limpieza o asistencia). Su interacción con el sistema se limita a registrar o confirmar la prestación de servicios adicionales.
- **Servicio (módulo interno del sistema):**
  Representa el componente que gestiona los servicios adicionales ofrecidos por el hotel, como restaurante, spa, transporte o lavandería. Permite su registro, disponibilidad y asociación con una reserva o cliente específico.

Actores externos

- **Sistema de pago:**
  Es una plataforma externa (por ejemplo, PayPal, Stripe o terminal bancaria) encargada de procesar los pagos electrónicos de los clientes. Se comunica con el sistema del hotel para confirmar o rechazar las transacciones de manera segura.

Principios de POO

- Abstracción: La abstracción consiste en identificar los elementos esenciales del sistema, dejando de lado los detalles innecesarios, para representar cada entidad mediante una clase con sus atributos y comportamientos más relevantes.

  En este caso:

  - Se abstraen las entidades principales del dominio del hotel: Cliente (Customer), Empleado (Employee), Recepcionista (Receptionist), Botones (Bellboy), Habitación (Room), Reserva (Reservation), Servicio (Service) y Pago (Payment).
  - Cada una refleja características propias del mundo real (por ejemplo, el cliente tiene nombre, teléfono y correo; una habitación tiene tipo, costo y estado).
  - Además, se definen las acciones esenciales que cada entidad puede realizar, como crear una reservación, registrar un cliente o procesar un pago.
- Encapsulamiento: El encapsulamiento busca proteger los datos de cada objeto, evitando que sean modificados directamente desde fuera de la clase. En lugar de eso, el acceso y modificación se hace mediante métodos definidos.

  Aplicación en el sistema:

  - Los datos de cada clase (por ejemplo, id, status, cost, email) se consideran atributos privados o protegidos, de modo que solo pueden ser accedidos mediante métodos específicos.
  - Por ejemplo, la clase Room no permite modificar directamente su estado (“disponible”, “ocupada” o “mantenimiento”) desde fuera; solo puede hacerse mediante métodos como assignCustomer() o releaseRoom().
  - Esto garantiza la integridad de la información, evitando inconsistencias o accesos indebidos.
- Herencia: La herencia permite reutilizar atributos y métodos de una clase base (superclase) en otras clases derivadas (subclases), reduciendo la redundancia y mejorando la organización.

  Aplicación en el sistema:

  - La clase Employee actúa como una superclase, ya que define los atributos y comportamientos comunes de todos los empleados del hotel (nombre, CURP, teléfono, correo, etc.).

    - De ella heredan las clases Receptionist y Bellboy, que además de heredar esos atributos, agregan funcionalidades específicas:
        - El Recepcionista puede crear, modificar o cancelar reservaciones.
        - El Botones (Bellboy) puede entregar llaves y notificar el estado de las habitaciones.
    - Gracias a la herencia, ambas subclases comparten la estructura general de un empleado pero se especializan en sus funciones concretas.
- Polimorfismo: El polimorfismo permite que una misma acción pueda tener diferentes comportamientos según el tipo de objeto que la ejecute.

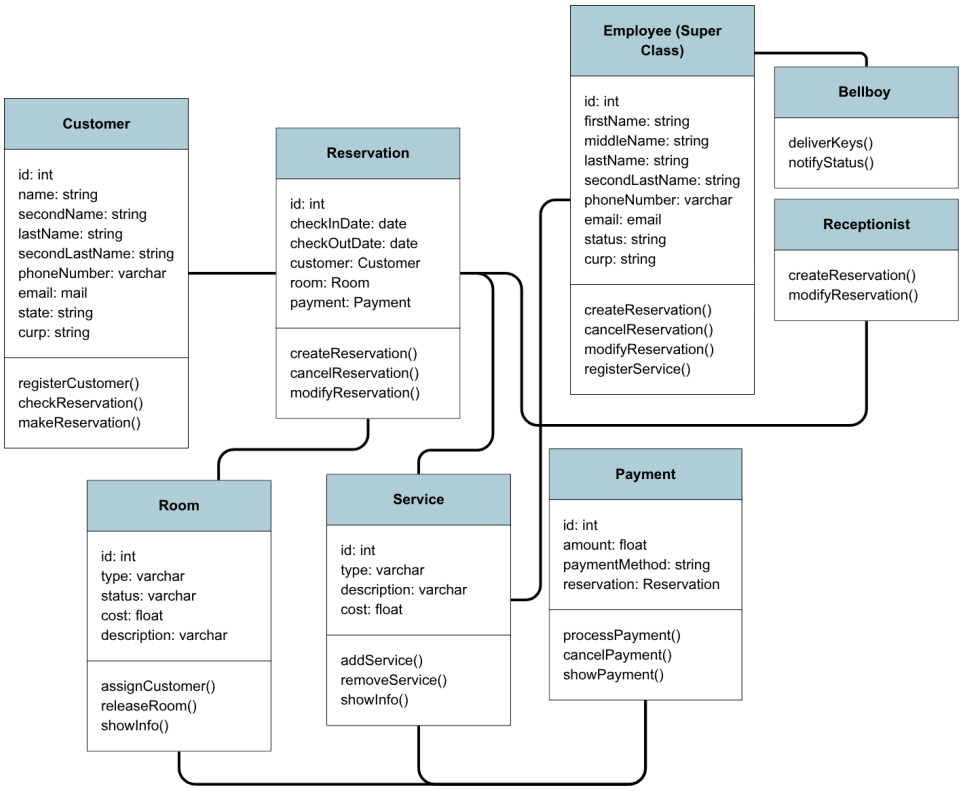
  Aplicación en el sistema:

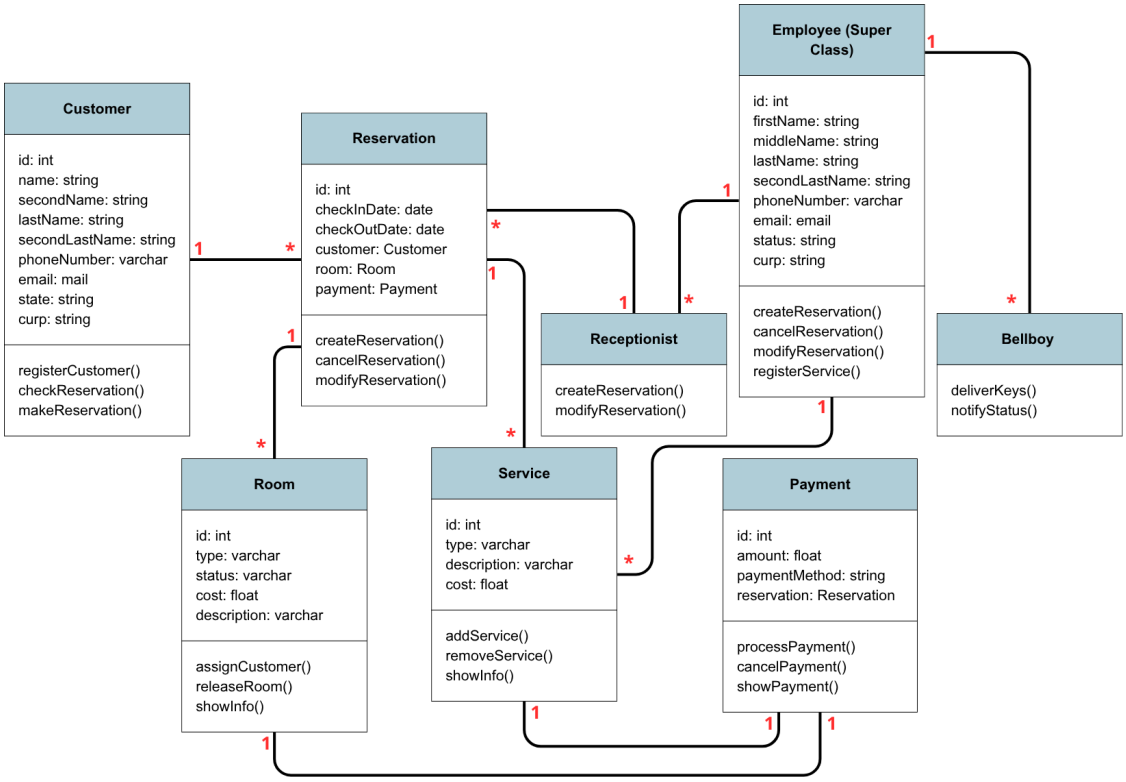
    - La superclase Employee define métodos generales como createReservation() o modifyReservation().
    - Sin embargo, las subclases Receptionist y Bellboy pueden redefinir (sobrescribir) esos métodos para adaptarlos a su rol.
        - Por ejemplo, el Recepcionista usa createReservation() para registrar una nueva reserva en el sistema.
        - Mientras que el Botones (Bellboy) podría usar un método con el mismo nombre, pero orientado a confirmar el estado de entrega o asistencia al cliente.
    - De este modo, el sistema puede invocar el mismo método en distintos objetos sin preocuparse por su tipo exacto, ya que cada uno ejecutará su propia versión del comportamiento.

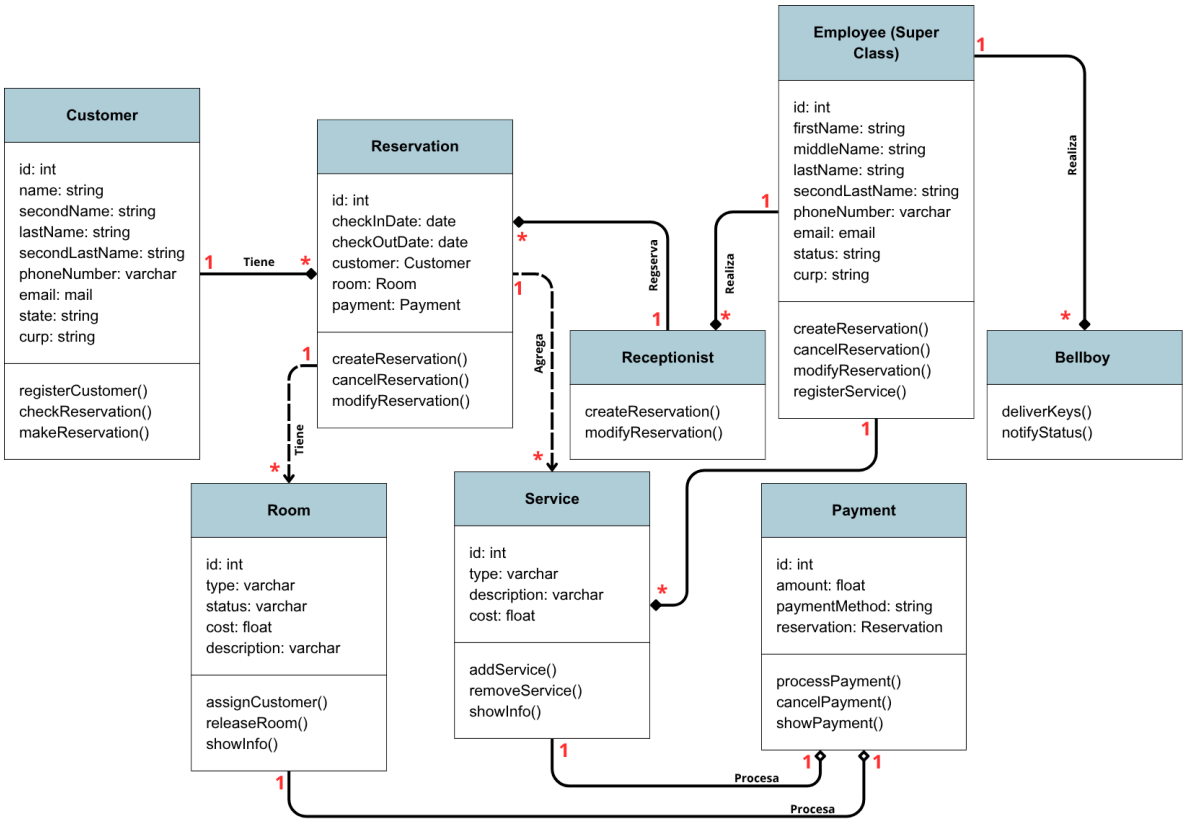
Modelado UML

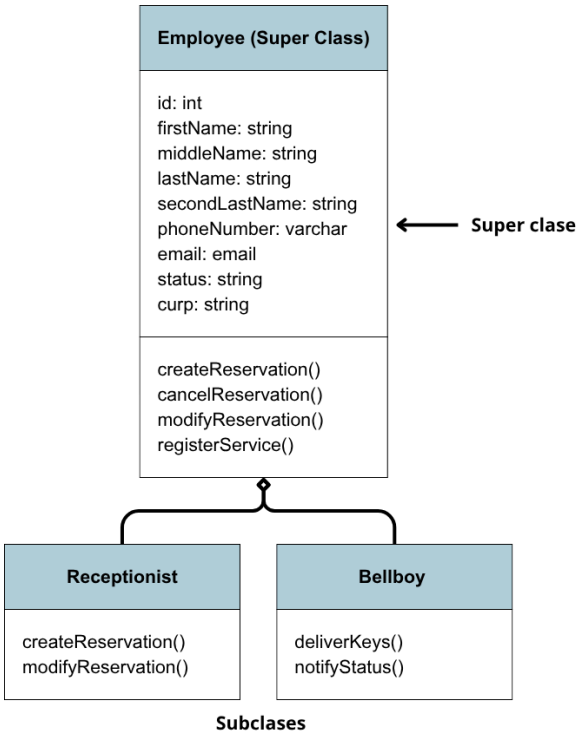
Orden de los diagramas:

1. Diagrama de clases
2. Asociaciones
3. Agregación y composición
4. Herencia
5. Secuencia
6. Casos de uso
7. Especificación de caso de uso

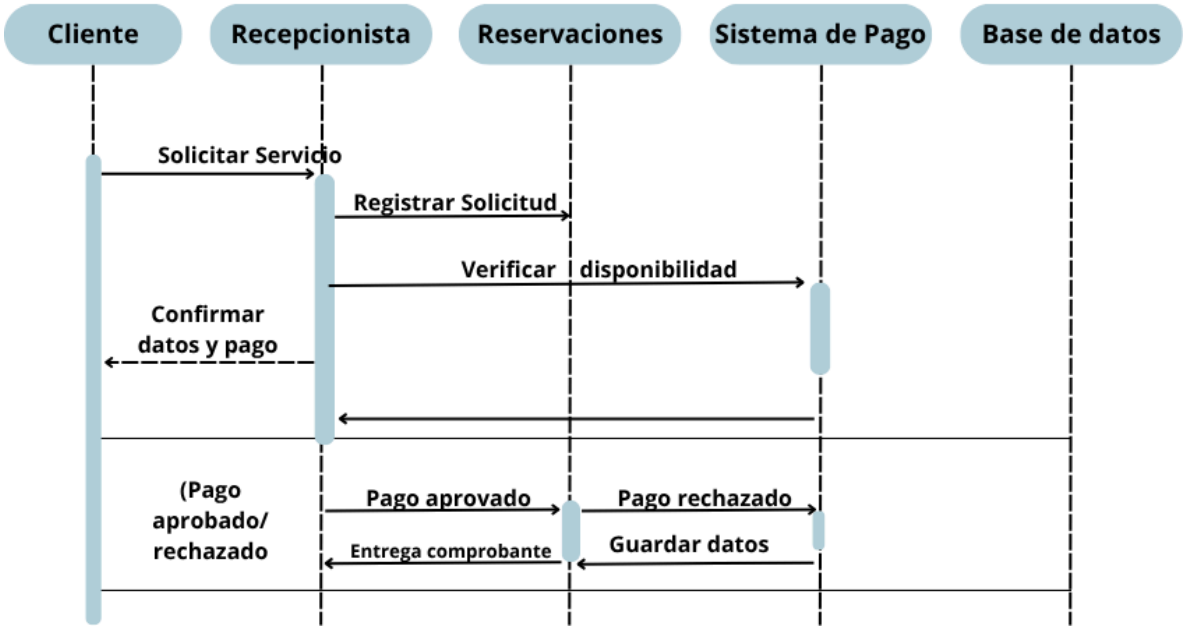

Customer
id: int
name: string
secondName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: mail
state: string
curp: string
registerCustomer()
checkReservation()
makeReservation()
Reservation
id: int
checkInDate: date
checkOutDate: date
customer: Customer
room: Room
payment: Payment
createReservation()
cancelReservation()
modifyReservation()
Employee (Super Class)
id: int
firstName: string
middleName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: email
status: string
curp: string
createReservation()
cancelReservation()
modifyReservation()
registerService()
Bellboy
deliverKeys()
notifyStatus()
Receptionist
createReservation()
modifyReservation()
Room
id: int
type: varchar
status: varchar
cost: float
description: varchar
assignCustomer()
releaseRoom()
showInfo()
Service
id: int
type: varchar
description: varchar
cost: float
addService()
removeService()
showInfo()
Payment
id: int
amount: float
paymentMethod: string
reservation: Reservation
processPayment()
cancelPayment()
showPayment()

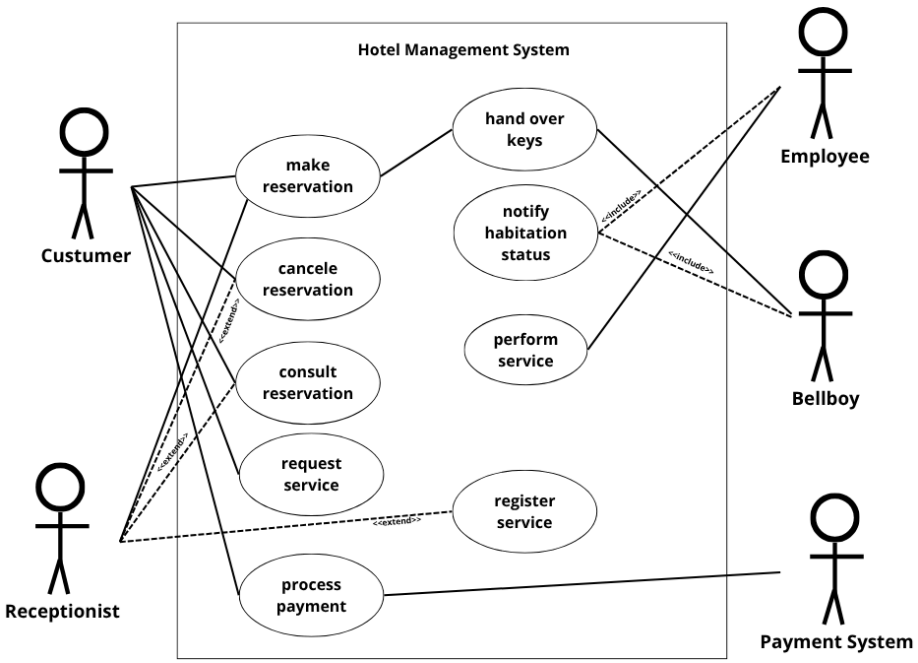

Customer
id: int
name: string
secondName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: mail
state: string
curp: string
registerCustomer()
checkReservation()
makeReservation()
Reservation
id: int
checkInDate: date
checkOutDate: date
customer: Customer
room: Room
payment: Payment
createReservation()
cancelReservation()
modifyReservation()
Employee (Super Class)
id: int
firstName: string
middleName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: email
status: string
curp: string
createReservation()
cancelReservation()
modifyReservation()
registerService()
Receptionist
createReservation()
modifyReservation()
Bellboy
deliverKeys()
notifyStatus()
Room
id: int
type: varchar
status: varchar
cost: float
description: varchar
assignCustomer()
releaseRoom()
showInfo()
Service
id: int
type: varchar
description: varchar
cost: float
addService()
removeService()
showInfo()
Payment
id: int
amount: float
paymentMethod: string
reservation: Reservation
processPayment()
cancelPayment()
showPayment()
1
*


Customer
id: int
name: string
secondName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: mail
state: string
curp: string
registerCustomer()
checkReservation()
makeReservation()
Reservation
id: int
checkInDate: date
checkOutDate: date
customer: Customer
room: Room
payment: Payment
createReservation()
cancelReservation()
modifyReservation()
Employee (Super Class)
id: int
firstName: string
middleName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: email
status: string
curp: string
createReservation()
cancelReservation()
modifyReservation()
registerService()
Receptionist
createReservation()
modifyReservation()
Bellboy
deliverKeys()
notifyStatus()
Room
id: int
type: varchar
status: varchar
cost: float
description: varchar
assignCustomer()
releaseRoom()
showInfo()
Service
id: int
type: varchar
description: varchar
cost: float
addService()
removeService()
showInfo()
Payment
id: int
amount: float
paymentMethod: string
reservation: Reservation
processPayment()
cancelPayment()
showPayment()
Tiene
Regserva
Realiza
Realiza
Agrega
Tiene
Procesa
Procesa


Employee (Super Class)
id: int
firstName: string
middleName: string
lastName: string
secondLastName: string
phoneNumber: varchar
email: email
status: string
curp: string
createReservation()
cancelReservation()
modifyReservation()
registerService()
Super clase
Receptionist
createReservation()
modifyReservation()
Bellboy
deliverKeys()
notifyStatus()
Subclases

Sistema de secuencia





Clase: Customer (Cliente)	
Caso de uso:	Registrar cliente
Id:	CU-01
Breve descripción:	El recepcionista registra los datos de un nuevo cliente en el sistema
Actores primarios:	Recepcionista
Actores secundarios:	Cliente
Precondiciones	1. El cliente no debe estar previamente registrado. 2. El sistema debe tener acceso a la base de datos.
Flujo principal:	1. El recepcionista selecciona “Registrar cliente”. 2. Ingresa los datos personales del cliente (nombre, CURP, teléfono, correo, etc.). 3. El sistema valida la información. 4. Se guarda el registro del cliente en la base de datos. 5. El sistema confirma el registro exitoso.
Postcondiciones	El cliente queda registrado y disponible para realizar reservaciones.

Clase: Employee (Empleado – Superclase)	
Caso de uso:	Administrar operaciones del hotel
Id:	CU-02
Breve descripción:	Representa las funciones comunes de los empleados, como gestionar reservaciones y servicios.
Actores primarios:	Empleado (Recepcionista o Bellboy)
Precondiciones	1. El empleado debe estar registrado y activo. 2. Debe tener los permisos correspondientes según su rol.
Flujo principal:	1. El empleado inicia sesión. 2. Accede a las opciones disponibles según su rol. 3. Realiza acciones como crear, modificar o cancelar reservaciones, o registrar servicios. 4. El sistema registra las operaciones realizadas.
Postcondiciones	Se actualiza el registro de actividades del empleado y las operaciones quedan guardadas.

Subclase: Receptionist (Recepcionista)	
Caso de uso:	Crear reservación
Id:	CU-03
Breve descripción:	El recepcionista genera una nueva reservación para un cliente, asignando habitación y pago.
Actores primarios:	Recepcionista
Actores secundarios:	Cliente, Sistema de pago
Precondiciones	1. El cliente debe estar registrado. 2. Debe haber habitaciones disponibles.
Flujo principal:	1. El recepcionista busca al cliente. 2. Selecciona habitación, fechas y servicios. 3. El sistema calcula el costo total. 4. Se procesa el pago y se confirma la reservación.
Postcondiciones	La reservación y el pago quedan registrados y la habitación pasa a estado “reservada”.

Subclase: Bellboy (Botones)	
Caso de uso:	Entregar llaves y notificar estado
Id:	CU-04
Breve descripción:	El botones entrega llaves a los huéspedes y actualiza el estado de las habitaciones
Actores primarios:	Botones
Actores secundarios:	Recepcionista
Precondiciones	1. El cliente debe tener una reservación activa. 2. La habitación debe estar lista para ocupación.
Flujo principal:	1. El botones consulta la habitación asignada al cliente. 2. Entrega las llaves al huésped. 3. Actualiza el estado de la habitación a “ocupada”. 4. Notifica al recepcionista que el cliente ingresó.
Postcondiciones	El sistema refleja la habitación como ocupada y se registra la entrega de llaves.

Clase: Reservation (Reservación)	
Caso de uso:	Gestionar reservación
Id:	CU-05
Breve descripción:	Permite crear, modificar o cancelar una reservación en el sistema.
Actores primarios:	Recepcionista
Actores secundarios:	Cliente, Sistema de pago
Precondiciones	1. El cliente debe estar registrado. 2. Debe haber habitaciones disponibles.
Flujo principal:	1. Se selecciona el cliente. 2. Se elige habitación y fechas. 3. El sistema calcula el total. 4. Se procesa el pago. 5. El sistema guarda la reservación.
Postcondiciones	La reservación queda activa y asociada al cliente y al pago.

Clase: Room (Habitación)	
Caso de uso:	Asignar habitación
Id:	CU-06
Breve descripción:	Asigna una habitación a una reservación y controla su estado.
Actores primarios:	Recepcionista
Actores secundarios:	Cliente
Precondiciones	1. La habitación debe estar disponible.
Flujo principal:	1. El recepcionista busca habitaciones disponibles. 2. Asigna una habitación a la reservación. 3. El sistema cambia el estado a “reservada”.
Postcondiciones	La habitación queda ocupada o reservada según corresponda.

Clase: Service (Servicio)	
Caso de uso:	Agregar servicio adicional
Id:	CU-07
Breve descripción:	El recepcionista o el cliente agrega servicios adicionales a una reservación (spa, restaurante, transporte, etc.).
Actores primarios:	Recepcionista/Cliente
Precondiciones	1. Debe existir una reservación activa. 2. El servicio debe estar disponible.
Flujo principal:	. Se selecciona el servicio deseado. 2. El sistema añade el costo al total de la reservación. 3. Se actualiza la información de la reserva.
Postcondiciones	El servicio queda registrado y asociado a la reservación del cliente

Clase: Payment (Pago)	
Caso de uso:	Procesar pago
Id:	CU-08
Breve descripción:	Permite registrar, validar y confirmar el pago de una reservación.
Actores primarios:	Recepcionista / Cliente
Actores secundarios:	Sistema de pago externo
Precondiciones	1. Debe existir una reservación generada. 2. El sistema de pago debe estar disponible.
Flujo principal:	1. El usuario selecciona método de pago (tarjeta, efectivo, etc.). 2. El sistema envía la solicitud al servicio de pago. 3. Se valida la transacción. 4. El sistema confirma el pago y actualiza la reservación.
Postcondiciones	El pago queda registrado y asociado a la reservación correspondiente.

Clases y Objetos

1. Clase Cliente

Descripción: Representa a la persona que realiza la reservación.

Atributos:

- idCliente : int
- nombre : str
- telefono : str
- email : str
- direccion : str

Métodos:

- registrarCliente() – Guarda los datos del cliente en el sistema.
- actualizarDatos() – Permite modificar la información personal.
- consultarReservaciones() – Muestra el historial de reservaciones del cliente.

Relaciones:

- Un Cliente puede tener muchas Reservaciones.
  → Relación 1 a N con Reservacion.

2. Clase Reservacion

Descripción: Contiene la información principal de una reserva.

Atributos:

- idReservacion : int
- fechaInicio : date
- fechaFin : date
- numPersonas : int
- estado : str (Ej. “Pendiente”, “Pagada”, “Cancelada”)

Métodos:

- crearReservacion() – Crea una nueva reservación.
- cancelarReservacion() – Cambia el estado a “Cancelada”.
- consultarEstado() – Devuelve el estado actual de la reservación.

Relaciones:

- Una Reservacion pertenece a un Cliente.
- Una Reservacion tiene una Habitación asignada.

- Una Reservacion puede tener un Pago asociado.

3. Clase Habitación

Descripción: Representa una habitación disponible en el hotel.

Atributos:

- idHabitacion : int
- numero : int
- tipo : str (Ej. “Sencilla”, “Doble”, “Suite”)
- precioPorNoche : float
- estado : str (Ej. “Disponible”, “Ocupada”, “Mantenimiento”)

Métodos:

- verificarDisponibilidad(fechaInicio, fechaFin) – Comprueba si la habitación está libre.
- actualizarEstado(nuevoEstado) – Cambia el estado actual de la habitación.

Relaciones:

- Una Habitación puede estar asociada a muchas Reservaciones a lo largo del tiempo.
  → Relación 1 a N con Reservacion.

4. Clase Pago

Descripción: Registra la información del pago de una reservación.

Atributos:

- idPago : int
- monto : float
- metodo : str (Ej. “Tarjeta”, “Efectivo”, “Transferencia”)
- fechaPago : date
- estado : str (Ej. “Aprobado”, “Rechazado”)

Métodos:

- procesarPago() – Realiza el cobro y actualiza el estado.
- validarPago() – Verifica que la transacción sea válida.
- generarRecibo() – Devuelve un comprobante de pago.

Relaciones:

- Un Pago está asociado a una sola Reservación.
  → Relación 1 a 1 con Reservacion.

5. Clase Empleado (Recepcionista)

Descripción: Representa a los trabajadores del hotel que gestionan las reservaciones.

Atributos:

- idEmpleado : int
- nombre : str
- puesto : str
- turno : str

Métodos:

- registrarReservacion(cliente, habitacion) – Inicia el proceso de registro.
- consultarDisponibilidad() – Muestra habitaciones disponibles.
- confirmarPago() – Verifica si un pago fue exitoso.

Relaciones:

- Un Empleado puede gestionar varias Reservaciones.
  → Relación 1 a N con Reservacion.

Códigos

- Bellboy:

```
from employee import Employee

class Bellboy(Employee):
    def deliverKeys(self, room):
        print(f"Delivery key for room {room.getId()} delivered to the customer.")

    def notifyStatus(self, room):
        print(f"Notifying status of room {room.getId()}: {room.getStatus()}")
```

- Receptionist:

```
from employee import Employee

class Receptionist(Employee):
    def createReservation(self, reservation):
```

```
                print(f"Receptionist creating reservation
#{reservation.getId()}...")
        super().createReservation(reservation)

    def modifyReservation(self, reservation, newCheckIn=None,
newCheckOut=None):
                print(f"Receptionist modifying reservation
#{reservation.getId()}...")
            super().modifyReservation(reservation, newCheckIn,
newCheckOut)
```

- Customer:

```
from typing import List

class Customer:
    def __init__(self, id, name, secondName, lastName, secondLastName,
phone, email, state, curp):
        self.__id = id
        self.__name = name
        self.__secondName = secondName
        self.__lastName = lastName
        self.__secondLastName = secondLastName
        self.__phone = phone
        self.__email = email
        self.__state = state
        self.__curp = curp
        self.__reservations: List = []

    # -------- Getters and Setters --------
    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getName(self): return self.__name
    def setName(self, name): self.__name = name

    def getSecondName(self): return self.__secondName
    def setSecondName(self, secondName): self.__secondName =
secondName

    def getLastName(self): return self.__lastName
```

```
    def setLastName(self, lastName): self.__lastName = lastName

    def getSecondLastName(self): return self.__secondLastName
    def setSecondLastName(self, secondLastName): self.__secondLastName
= secondLastName

    def getPhone(self): return self.__phone
    def setPhone(self, phone): self.__phone = phone

    def getEmail(self): return self.__email
    def setEmail(self, email): self.__email = email

    def getState(self): return self.__state
    def setState(self, state): self.__state = state

    def getCurp(self): return self.__curp
    def setCurp(self, curp): self.__curp = curp

    def getReservations(self): return self.__reservations

    # -------- Métodos --------
    def registerCustomer(self):
        print(f"Customer {self.__name} has been registered.")

    def checkReservation(self):
        if not self.__reservations:
            print(f"Customer {self.__name} has no reservations.")
        else:
                                    print(f"Customer  {self.__name}  has
{len(self.__reservations)} reservation(s).")

    def makeReservation(self, reservation):
        self.__reservations.append(reservation)
                print(f"Reservation  made  by  {self.__name}  for  room
{reservation.getRoom().getId()}.")
```

- Room:

```
class Room:
```

```
        def __init__(self, id, type, status="Available", cost=0.0,
description=""):
        self.__id = id
        self.__type = type
        self.__status = status
        self.__cost = cost
        self.__description = description

    # -------- Getters and Setters --------
    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getType(self): return self.__type
    def setType(self, type): self.__type = type

    def getStatus(self): return self.__status
    def setStatus(self, status): self.__status = status

    def getCost(self): return self.__cost
    def setCost(self, cost): self.__cost = cost

    def getDescription(self): return self.__description
        def setDescription(self, description): self.__description =
description

    # -------- Métodos --------
    def assignCustomer(self, customer):
        if self.__status == "Available":
            self.__status = "Not available"
                                print(f"Room {self.__id} assigned to
{customer.getName()}.")
        else:
                        print(f"Room {self.__id} is not available for
{customer.getName()}.")

    def releaseRoom(self):
        self.__status = "Available"
        print(f"Room {self.__id} released.")
```

```
    def showInfo(self):
        print(f"Room {self.__id} - Type: {self.__type} - Status: {self.__status} - Cost: ${self.__cost} - Description: {self.__description}")
```

- Payment:

```
from datetime import date

class Payment:
    def __init__(self, id, amount, paymentMethod, reservation=None):
        self.__id = id
        self.__amount = amount
        self.__paymentMethod = paymentMethod
        self.__reservation = reservation
        self.__date = date.today()

    # -------- Getters and Setters --------
    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getAmount(self): return self.__amount
    def setAmount(self, amount): self.__amount = amount

    def getPaymentMethod(self): return self.__paymentMethod
    def setPaymentMethod(self, method): self.__paymentMethod = method

    def getReservation(self): return self.__reservation
    def setReservation(self, reservation): self.__reservation = reservation

    def getDate(self): return self.__date

    # -------- Métodos --------
    def processPayment(self):
        print(f"Payment processed for ${self.__amount} by {self.__paymentMethod} on {self.__date}.")

    def cancelPayment(self):
```

```
        print(f"Payment #{self.__id} canceled.")

    def showPayment(self):
            print(f"Payment  #{self.__id}:  ${self.__amount},  method:
{self.__paymentMethod}, date: {self.__date}")
```

- Resevation:

```
class Reservation:
    def  __init__(self,  id,  checkIn,  checkOut,  customer,  room,
payment=None):
        self.__id = id
        self.__checkIn = checkIn
        self.__checkOut = checkOut
        self.__customer = customer
        self.__room = room
        self.__payment = payment
        self.__services = []

    # -------- Getters and Setters --------
    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getCheckIn(self): return self.__checkIn
    def setCheckIn(self, checkIn): self.__checkIn = checkIn

    def getCheckOut(self): return self.__checkOut
    def setCheckOut(self, checkOut): self.__checkOut = checkOut

    def getCustomer(self): return self.__customer
    def setCustomer(self, customer): self.__customer = customer

    def getRoom(self): return self.__room
    def setRoom(self, room): self.__room = room

    def getPayment(self): return self.__payment
    def setPayment(self, payment): self.__payment = payment

    def getServices(self): return self.__services

```

```
    # -------- Métodos --------
    def createReservation(self):
        if self.__room.getStatus() == "Available":
            self.__room.assignCustomer(self.__customer)
            self.__customer.makeReservation(self)
             print(f"Reservation #{self.__id} created successfully for
{self.__customer.getName()}")
        else:
                        print(f"Unable to create reservation. Room
{self.__room.getId()} not available.")

    def cancelReservation(self):
        self.__room.releaseRoom()
        print(f"Reservation #{self.__id} cancelled.")

    def modifyReservation(self, newCheckIn=None, newCheckOut=None):
        if newCheckIn:
            self.__checkIn = newCheckIn
        if newCheckOut:
            self.__checkOut = newCheckOut
                print(f"Reservation #{self.__id} updated: from
{self.__checkIn} to {self.__checkOut}.")
```

- Service:

```
class Service:
    def __init__(self, id, type, cost, description):
        self.__id = id
        self.__type = type
        self.__cost = cost
        self.__description = description

    # -------- Getters and Setters --------
    def getId(self): return self.__id
    def setId(self, id): self.__id = id

    def getType(self): return self.__type
    def setType(self, type): self.__type = type

    def getCost(self): return self.__cost
```

```
    def setCost(self, cost): self.__cost = cost

    def getDescription(self): return self.__description
        def setDescription(self, description): self.__description = description

    # -------- Métodos --------
    def addService(self):
        print(f"Service '{self.__type}' added. Cost: ${self.__cost}")

    def removeService(self):
                print(f"Service '{self.__type}' removed. Cost: ${self.__cost}")

    def showInfo(self):
            print(f"Service: {self.__type} - {self.__description} - ${self.__cost}")
```

- Main:

```
from datetime import date
from customer import Customer
from room import Room
from payment import Payment
from reservation import Reservation
from receptionist import Receptionist
from bellboy import Bellboy
from service import Service

# ---------- Crear objetos base ----------
customer1 = Customer(
    1, "Andrea", "Sarahi", "Lopez", "Guerrero",
    "4420000000", "andrea@mail.com", "Querétaro", "LOGA001122QRO"
)

room1 = Room(
    101, "Suite", "Available", 1500.0, "Sea view"
)

payment1 = Payment(
```

```
    1, 1500.0, "Credit Card"
)

# ---------- Crear reserva ----------
reservation1 = Reservation(
    1, date(2025, 10, 10), date(2025, 10, 15),
    customer1, room1, payment1
)

# ---------- Crear empleados ----------
receptionist = Receptionist(
    1, "Brigitte", "", "Herrera", "Rodriguez",
    "4421111111", "brigitte@mail.com", "Active", "HERB001122QRO"
)

bellboy = Bellboy(
    2, "Esmeralda", "", "Vazquez", "Garcia",
    "4422222222", "esme@mail.com", "Active", "VAGE001122QRO"
)

# ---------- Simulación de flujo de trabajo del hotel ----------
print("\n---- Registro de cliente ----")
customer1.registerCustomer()

print("\n---- Creación de reserva ----")
receptionist.createReservation(reservation1)

print("\n---- Procesando pago ----")
payment1.processPayment()

print("\n---- Entrega de llaves ----")
bellboy.deliverKeys(room1)

print("\n---- Estado de la habitación ----")
bellboy.notifyStatus(room1)

print("\n---- Agregar servicio extra ----")
```

```
service1 = Service(1, "Spa", 300, "Full body massage")
receptionist.registerService(service1)

print("\n---- Modificación de la reserva ----")
receptionist.modifyReservation(reservation1,    newCheckIn=date(2025,
10, 11))

print("\n---- Cancelación de la reserva ----")
receptionist.cancelReservation(reservation1)

print("\n---- Liberar habitación ----")
room1.releaseRoom()

print("\n---- Mostrar información ----")
room1.showInfo()
service1.showInfo()
payment1.showPayment()
```