Universidad Nacional
ARTURO JAURETCHE

INSTITUTO
INGENIERÍA Y
AGRONOMÍA

19-08-2024

Metodologías de la Programación II
Trabajo Final - Informe

Docente:
Claudia Cappelletti

Integrantes:
Lautaro Duarte
Ian Galarza
Franco Giordano
Jonathan Lekander
Tomas Rippa

Grupo N°4

Índice

Introducción

En el presente informe, haremos uso y aplicación de diversos conceptos y metodologías abordadas durante la cursada de la materia *Metodologías de la Programación II.* Para el actual caso, la elaboración de este informe gira en torno al desarrollo de una aplicación en lenguaje SmallTalk

Primeramente, vamos a dar una breve descripción del contexto sobre el cual parte la idea del desarrollo. A su vez, detallaremos la especificación e implementación de cada clase involucrada en el entorno planteado, como también la codificación de la aplicación que involucra a estas clases. Posteriormente, explicaremos las metodologías de trabajo empleadas por los integrantes de este equipo, tratando particularmente con aquellas que sean consideradas prácticas ágiles.

Desarrollo

La aplicación desarrollada para la elaboración de este informe consiste de un sistema de reservas de hotel el cual incorpora funcionalidades aprendidas durante la cursada de la materia. Para llevar a cabo su implementación, fue necesario definir claramente un problema a resolver:

Problema

Una persona busca realizar una reserva en la recepción de un hotel, en el cual se accede al sistema de reservas del mismo, y se recibe los datos por parte de la persona (número de cliente, tipo de habitación deseada, fecha de inicio y fecha de fin, y la cantidad de huéspedes).

Con los datos ya recibidos, se verifica en el sistema que la persona sea un cliente registrado, en la cual en caso de que no esté registrado se notificará en la pantalla, para luego proceder a verificar que haya una habitación disponible del tipo solicitado. Una vez se realizan todas las verificaciones, se procede a ingresar un número de reserva, para luego crearla, y se modifica el estado de la habitación para indicar que está ocupada, notificando en pantalla que la reserva fue realizada con éxito.

Diagrama UML

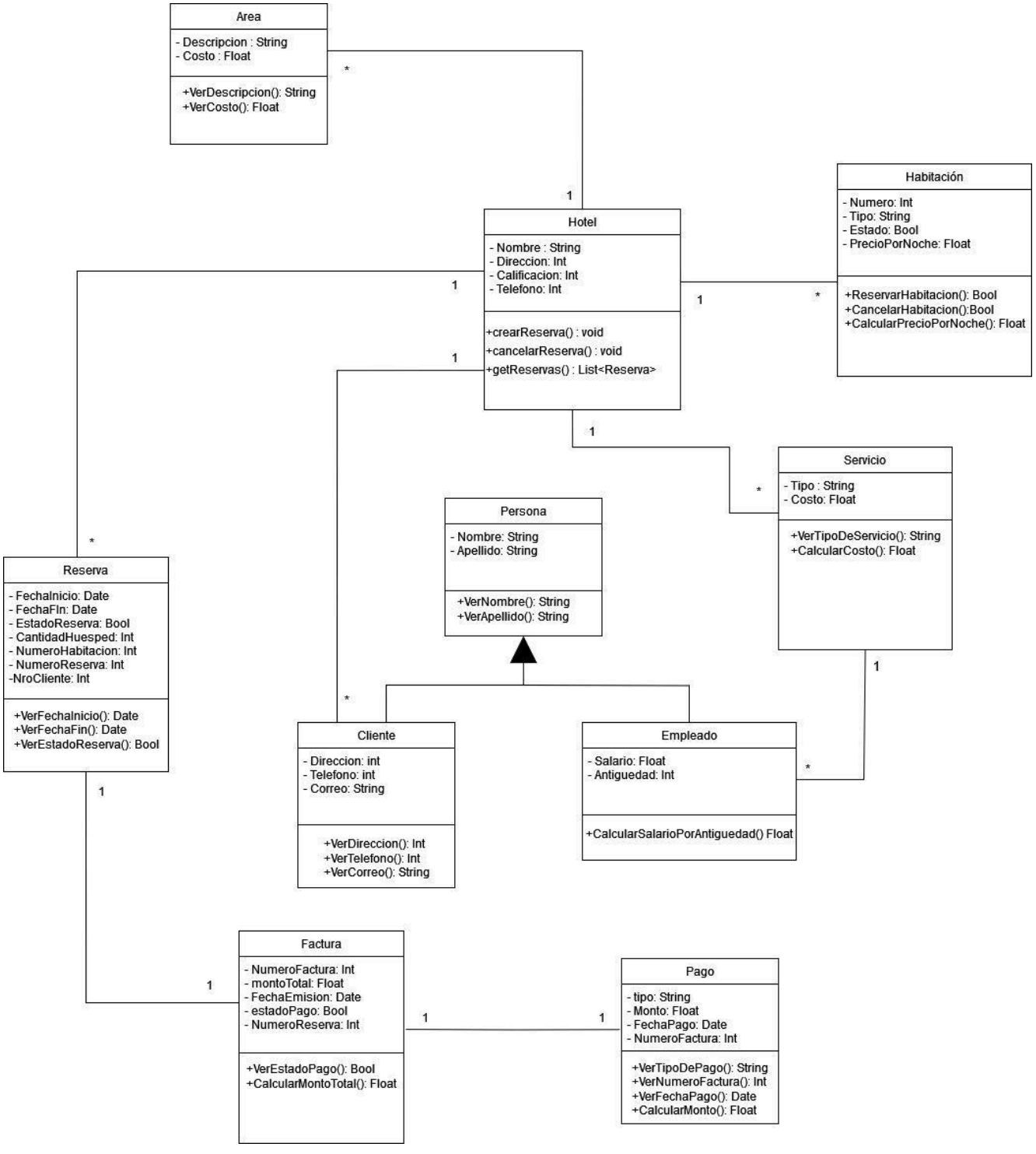
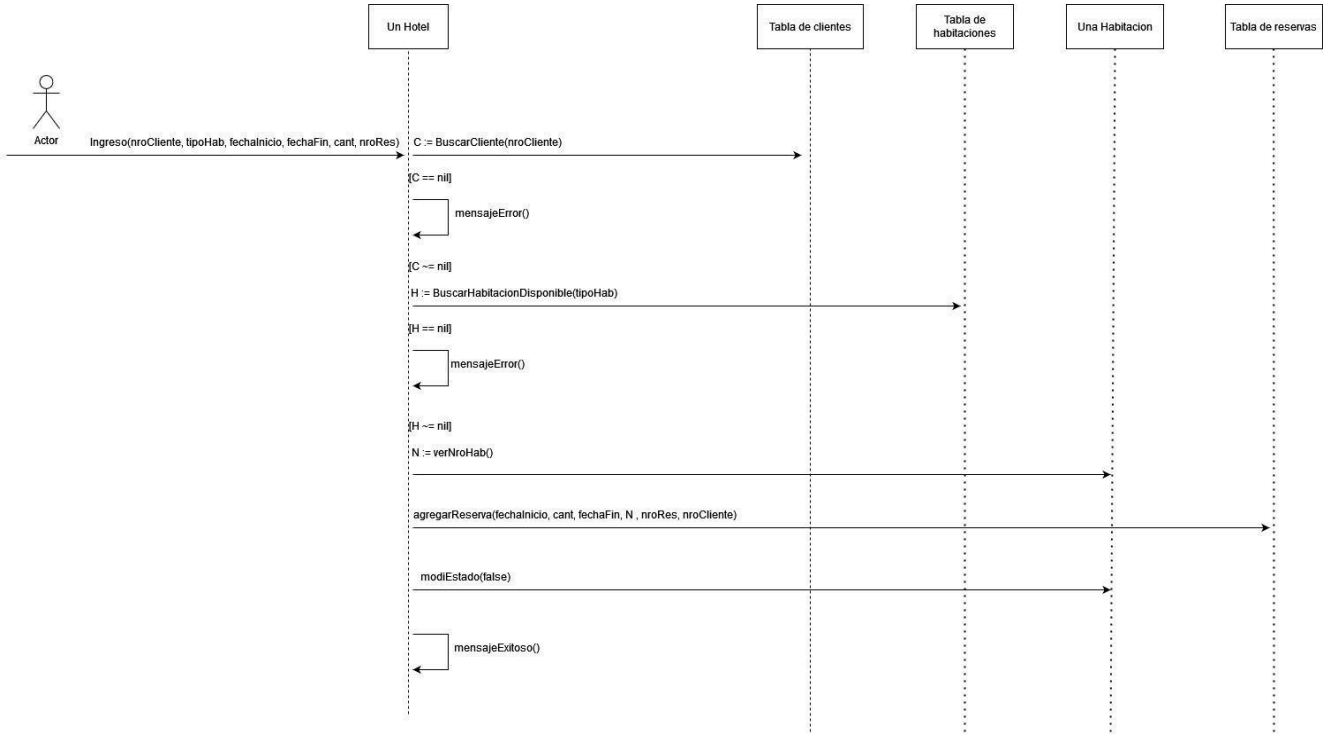


Diagrama de secuencias



Implementación y especificación de clases utilizadas

A continuación se detallarán las clases desarrolladas en nuestro sistema de reservas para hoteles.

Especificaciones

Clase	Atributos	Métodos principales
Hotel	nombre dirección calificación teléfono áreas reservas habitaciones servicios clientes	initHotelnom:dir:cal:tel: agregarReserva: unaRes agregarHabitacion: unaHab agregarCliente: unClient todasLasReservas todasLasHabitaciones todosLosClientes buscarCliente:unNum buscarHabitacionDisponibleTipo: unTipo buscarHabitacionPorNumero: unNum reservasVigentes verNombre

Reserva	fechaInicio fechaFin estadoReserva cantidadHuesped numeroHabitacion numeroReserva numeroCliente	initHabitacionnro:tipo:precioPorNoche: esVigente: unaFecha cantHuesped modiEstadoReserva: unEstado modiFechaFin: unaFecha modiFechaInicio: unaFecha modiNumeroHabitacion: unNum verEstadoReserva verFechaFin verFechaInicio verNumeroCliente verNumeroReserva verNumeroHabitacion
Habitación	nroHab tipo estado precioPorNoche	initHabitacionnro:tipo:precioPorNoche: verEstado verNroHab verTipo verPrecioPorNoche modiEstado: unEstado modiprecioPorNoche: unPrecio
Cliente	nombre apellido dirección correo teléfono nroCliente	initClientDireccion:correo:telefono:nombr e:apellido: verNroCliente verCorreo verDireccion verTelefono
Persona	nombre apellido	initPersonaNombre:apellido: verApellido verNombre
Empleado	nombre apellido salario antigüedad	initEmpleadoSalario:antigüedad:nombre: apellido: modiAntiguedad: verAntiguedad verSalario
Factura	nroFactura montoTotal fechaEmision estadoPago nroReserva	calcularMontoTotal verEstadoPago initFacturanroFactura:montoTotal:fechaE mision:estadoPago:nroReserva:
Servicio	tipo costo	initServiciotipo:costo: modiCosto: unCosto modiTipo: unTipo verCosto verTipo

Implementaciones clases utilizadas

Hotel

Métodos de Clase:

- **crearHotelnom:dir:cal:tel:**

Permite crear una instancia del objeto Hotel inicializando sus atributos.

```
crearHotelnom:unNom dir:unaDir cal:unaCal tel:unTel
^(self new) initHotelnom:unNom dir:unaDir cal:unaCal tel:unTel.
```

Métodos de Instancia:

- **InitHotelnom:dir:cal:tel:**

Método para inicializar un hotel dentro del sistema. En el mismo se le debe asignar sus 4 atributos, además de la creación de las colecciones dentro del hotel.

```
initHotelnom:unNom dir:unaDir cal:unaCal tel:unTel
nombre:= unNom.
direccion:= unaDir.
calificacion:= unaCal.
telefono:= unTel.
areas:= OrderedCollection new.
reservas:= OrderedCollection new.
habitaciones:= OrderedCollection new.
servicios:= OrderedCollection new.
clientes:= OrderedCollection new.
```

- **verNombre**

Retorna el atributo nombre del Hotel.

```
verNombre
^nombre.
```

- **agregarHabitacion:**

Permite agregar una instancia de Habitación a la colección de habitaciones del Hotel.

```
agregarHabitacion:unaHab
habitaciones add: unaHab.
```

- **agregarCliente:**

Permite agregar una instancia de Cliente a la colección de clientes del Hotel.

```
agregarCliente:unClient
clientes add: unClient .
```

- **agregarReserva:**

Permite agregar una instancia de Reserva a la colección de reservas del Hotel.

```
agregarReserva:unaRes
reservas add: unaRes.
```

- **buscarCliente:**

Detecta y retorna el primer objeto Cliente en la colección de clientes del Hotel que coincida con el número de cliente proporcionado. En caso de no encontrar ningún resultado, retorna nil.

```
buscarCliente: unNum
 ^ clientes detect: [:client |  client verNroCliente = unNum ] ifNone: [nil].
```

- **buscarHabitacionDisponibleTipo:**

Detecta y retorna el primer objeto Habitación en la colección de habitaciones del Hotel que coincida con el tipo proporcionado, y tenga estado True (Disponible). En caso de no encontrar ningún resultado, retorna nil.

```
buscarHabitacionDisponibleTipo: unTipo
   ^ habitaciones detect: [:hab | (hab verEstado = true and: [hab verTipo = unTipo])] ifNone: [nil].
```

- **buscarHabitacionPorNumero:**

Detecta y retorna el primer objeto Habitacion en la colección de habitaciones del hotel que coincida con el número de Habitación ingresado. En caso de no encontrar ningún resultado, retorna nil.

```
buscarHabitacionPorNumero: unNum
^ habitaciones detect: [:hab | hab verNroHab = unNum] ifNone: [nil].
```

- **todasLasHabitaciones**

Retorna la colección de habitaciones del Hotel.

```
todasLasHabitaciones
^ habitaciones.
```

- **todasLasReservas**

Retorna la colección de reservas del Hotel.

```
todasLasReservas
^ reservas .
```

- **todosLosClientes**

Retorna la colección de clientes del Hotel.

```
todosLosClientes
^ clientes .
```

- **reservasVigentes**

DESCRIPCION

IMAGEN

Habitación

Métodos de Clase:

- **crearHabitacionNro:tipo:precioPorNoche:**

Permite crear una instancia del objeto Habitacion inicializando sus atributos.

```
crearHabitacionNro:unNum tipo:unTipo precioPorNoche:unPrecioPorNoche
^(self new)initHabitacionnro: unNum tipo:unTipo  precioPorNoche: unPrecioPorNoche.
```

Métodos de Instancia:

- **initHabitacionnro:tipo:precioPorNoche:**

Método para inicializar una habitación dentro del sistema. En el mismo se le debe asignar sus 3 atributos, además de la asignación 'true' al atributo estado.

```
initHabitacionnro: unNum tipo:unTipo  precioPorNoche: unPrecioPorNoche
nroHab:=unNum.
tipo:=unTipo.
estado:= true.
precioPorNoche:=unPrecioPorNoche.
```

- **modiEstado:**

Modifica el atributo estado de la Habitación.

```
modiEstado:unEstado
estado:=unEstado.
```

- **verEstado**

Retorna el atributo estado de la Habitación.

```
verEstado
^estado.
```

- **verNroHab**

Retorna el atributo nroHab de la Habitación.

```
verNroHab
^nroHab.
```

- **verTipo**

Retorna el atributo tipo de la Habitación.

```
verTipo
^tipo.
```

- **verPrecioPorNoche**

Retorna el atributo precioPorNoche de la Habitación

```
verPrecioPorNoche
^precioPorNoche.
```

Reserva

Métodos de Clase:

- **crearReservaIni:Fin:cantHuesp:numHab:nroRes:nroCliente:**

Permite crear una instancia del objeto Reserva inicializando sus atributos.

```
crearReservaIni:unaFechaIni Fin:unaFechaFin cantHuesp:unaCant numHab:unNroHab nroRes:unNroRes nroCliente:unNroCliente
^(self new) initReservaIni: unaFechaIni Fin: unaFechaFin cantHuesp: unaCant numHab: unNroHab nroRes: unNroRes nroCliente: unNroCliente.
```

Métodos de Instancia:

- **InitReservaIni:Fin:cantHuesp:numHab:nroRes:nroCliente:**

Método para inicializar una reserva dentro del sistema. En el mismo se le debe asignar sus 6 atributos, además de la creación de la asignación ‘true’ al atributo estadoReserva.

```
initReservaIni:unaFechaIni Fin:unaFechaFin cantHuesp:unaCant numHab:unNroHab nroRes:unNroRes nroCliente:unNroCliente
fechaInicio := Date fromString: unaFechaIni format: 'dd-mm-yyyy'.
fechaFin := Date fromString: unaFechaFin format: 'dd-mm-yyyy'.
estadoReserva := true.
cantidadHuesped := unaCant.
numeroReserva := unNroRes.
numeroHabitacion := unNroHab.
numeroCliente := unNroCliente.
```

- **verFechaInicio**

Retorna la fecha de inicio de la Reserva

```
verFechaInicio
    ^fechaInicio.
```

- **verFechaFin**

Retorna la fecha de fin de la Reserva

```
verFechaFin
    ^fechaFin
```

- **verEstadoReserva**

Retorna el estado de la Reserva

```
verEstadoReserva
    ^estadoReserva
```

- **cantHuesped**

Retorna la cantidad de huéspedes de la Reserva

```
cantHuesped
    ^cantidadHuesped.
```

- **verNumeroHabitacion**

Retorna el número de habitación de la Reserva

```
verNumeroHabitacion
    ^numeroHabitacion
```

- **verNumeroCliente**

Retorna el número de cliente de la Reserva

```
verNumeroCliente
    ^numeroCliente.
```

- **verNumeroReserva**

Retorna el número propio de la Reserva

```
verNumeroReserva
    ^numeroReserva.
```

Cliente → Persona

Métodos de Clase:

- **crearClienteNombre:apellido:dirección:teléfono:correo:nroCliente:**

Permite crear una instancia del objeto Cliente inicializando sus atributos.

```
crearClienteNombre: unNombre apellido: unApellido direccion:unaDireccion telefono: unTelefono correo: unCorreo nroCliente: unNum
^(self new) initClientDireccion: unaDireccion correo: unCorreo telefono: unTelefono nombre: unNombre apellido: unApellido nroCliente: unNum.
```

Métodos de Instancia:

- **initClientDireccion:correo:telefono:nombre:apellido:nroCliente:**

Método para inicializar un cliente dentro del sistema. En el mismo se le debe asignar sus 6 atributos, 2 de los cuales pertenecen a su superclase (Persona).

```
initClientDireccion: unaDireccion correo: unCorreo telefono: unTelefono nombre: unNombre apellido: unApellido nroCliente: unNum
    super initPersonaNombre: unNombre apellido: unApellido.
    direccion := unaDireccion.
    correo := unCorreo.
    telefono := unTelefono.
    nroCliente := unNum.
```

- **verNroCliente**

Retorna el atributo nroCliente del Cliente.

```
verNroCliente
    ^nroCliente .
```

- **verDireccion**

Retorna el atributo dirección del Cliente

```
verDireccion
    ^direccion.
```

- **verTelefono**

Retorna el atributo teléfono del Cliente

```
verTelefono
    ^telefono.
```

- **verCorreo**

Retorna el atributo correo del Cliente

```
verCorreo
	^correo.
```

Persona

Métodos de Clase:

- **crearPersonaNombre:apellido:**

Permite crear una instancia del objeto Persona inicializando sus atributos.

```
crearPersonaNombre: unNombre apellido: unApellido
^(self new) initPersonaNombre: unNombre apellido: unApellido.
```

Métodos de Instancia:

- **initPersonaNombre:apellido:**

Método para inicializar una persona dentro del sistema. En el mismo se le debe asignar sus 2 atributos.

```
initPersonaNombre: unNombre apellido: unApellido
	nombre := unNombre.
	apellido := unApellido.
```

- **verNombre**

Retorna el atributo nombre de la Persona.

```
verNombre
	^nombre.
```

- **verApellido**

Retorna el atributo apellido de la Persona.

```
verApellido
	^apellido.
```

Implementación de la aplicación en Smalltalk

REVISAR

El proyecto desarrollado consiste de un menú sencillo el cual nos permite llevar a cabo algunas de las funciones básicas, que consideramos que debería tener un hotel.
Podemos encontrar opciones tales como:
-La creación de:
	-Un nuevo Cliente
	-Una nueva Reserva
	-Habitaciones

-La visualización de:
    -Todas las Habitaciones
    -Las Habitaciones según su Tipo
    -Las Habitaciones no disponibles
    -La cantidad de reservas por cantidad de Huéspedes
    -Los Clientes
    -Todas las reservaciones, por fecha
    -Cantidad de reservaciones por Cliente
-La modificación de:
    -El estado de las Habitaciones

Para la creacion de los objetos mencionados anteriormente, se utilizo el Prompter para que uno pueda ingresar los datos que componen a los objetos en cuestion. En el caso de las Reservaciones

(Yo pondria imagenes del codigo de la aplicacion + la explicacion de lo qué hace dicha imagen)

Metodología utilizada en el proyecto

REVISAR

En nuestro proyecto, adoptamos la metodología Scrumban, que combina las mejores prácticas de Scrum y Kanban. Elegimos esta metodología debido a su flexibilidad, ya que no requiere roles estrictos. En lugar de tener roles definidos, adoptamos un enfoque colaborativo donde todos los miembros del equipo son responsables del progreso y la ejecución del proyecto.
Durante todo el proceso del desarrollo, organizamos nuestras tareas utilizando un Tablero Scrumban, que nos permitió visualizar el estado de cada tarea. Cada vez que se completaba una tarea, por ejemplo, “Crear la clase Hotel “ (Finalizada), se actualizaba el tablero. Las tareas pendientes, por ejemplo, “Crear la clase Reserva”, se mantenían claramente visibles, lo que nos ayudaba a priorizar lo que se debería hacer a continuación.
Además, al final de cada presentación, realizamos una breve daily de 15 minutos para organizarnos respecto a las próximas entregas. Uno de los pilares más importantes en nuestra metodología fue la comunicación constante “cara a cara”,lo que nos permitió tomar decisiones rápidamente.Cada miembro asumió la responsabilidad de las tareas según sus habilidades, fomentando un ambiente colaborativo.

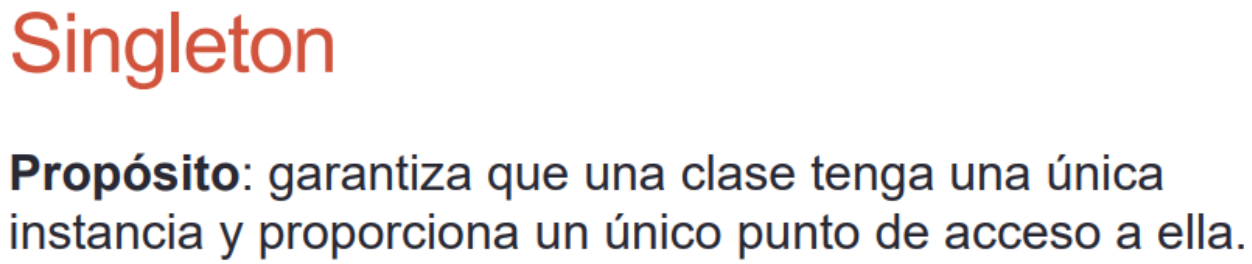
(Elegi esta metodología porque me parecio sencillo de explicar, pero facilmente lo podriamos cambiar por otra metodología)
fuente: Utilice los tps que entregamos durante la cursada

Frameworks para desarrollar el proyecto en POO

Casos de uso de Patrones

REVISAR

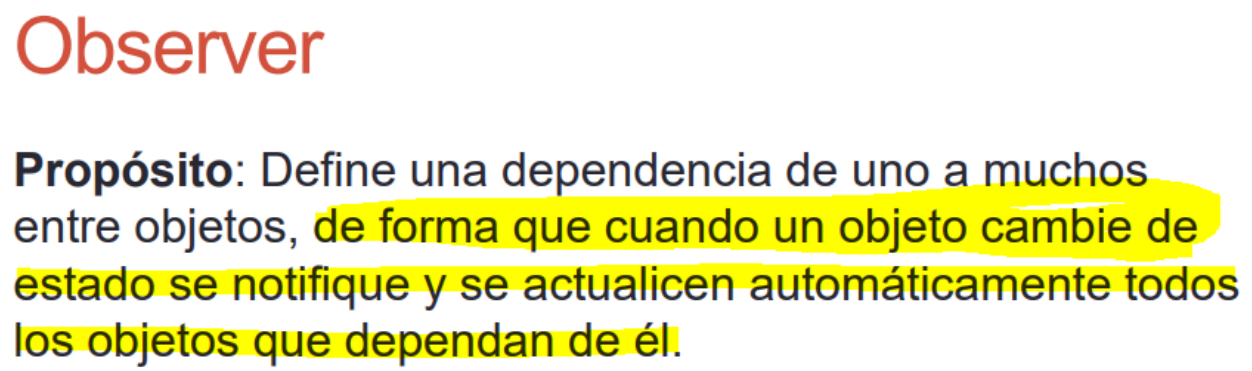
Fuente: Los pdf de Metodologia 1



Patrón creacional Singleton

El patrón Singleton es un patrón creacional que asegura que una clase tenga una única instancia durante toda la ejecución de la aplicación y proporciona un único punto de acceso a dicha instancia.
Dado que nuestro informe se basa en un sistema de reservas de hoteles,uno de los problemas que puede llegar a presentarse en este contexto es la creación de múltiples instancias de la clase Hotel, esto podría generar una gestión desorganizada y propensa a errores en los recursos. Donde cada instancia de Hotel maneja su propio conjunto de Habitaciones,Clientes y Reservas, lo que generaría problemas de duplicidad, provocando una ineficiencia en el sistema.
Como solución a este problema, se implementaría el patrón Singleton, garantizando que solo exista una única instancia de la clase Hotel.Con este patrón, evitamos la creación de instancias adicionales y aseguramos que las operaciones sean coherentes durante la ejecución del sistema.

(Yo pondria foto donde podría agregar el singleton o donde surgiría el problema)



Patrón de comportamiento Observer

El patrón Observer define una dependencia de uno a muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y se actualicen automáticamente todos los objetos que dependan de él.
En nuestro sistema de reservas de hoteles, uno de los problemas que podría surgir es la necesidad de mantener “actualizada” la información. Supongamos que varios clientes están consultando

simultáneamente la disponibilidad de habitaciones y se realizan cambios (reservar, cancelar, consultar, etc). En este caso los demás clientes deberían ser notificados automáticamente para ver dicha información “actualizada”.
(En esta parte lo podriamos dejar como esta o sugerir un problema hipotetico, capaz que al no implementar el observer hace que la experiencia del usuario sea mala)

(podríamos agregar otro patrón, pero no se me ocurre otro)

Repositorios utilizados en el desarrollo del software

Bibliografía