

# Programación 4

## Informe del Modelo de Diseño - Diagramas de Comunicación -

### Grupo 01

Integrantes

Amorena Diego  
Bello Federico  
Giacometti Danilo  
Rodriguez Mathias  
Olveyra Javier

**CI:** 5.011.502-7  
**CI:** 4.993.837-3  
**CI:** 5.172.582-9  
**CI:** 5.198.420-3  
**CI:** 4.971.974-7

Docente: Martin Prino

<b>Ejercicio 1</b>	<b>3</b>
<b>Ejercicio 2</b>	<b>4</b>
1 Realización de Casos de Uso (Diagramas de comunicación)	4
1.1 Alta de Usuario	4
1.2 Alta de Hostal	6
1.3 Alta de Habitación	7
1.4 Asignar empleado a hostal	9
1.5 Realizar Reserva	11
1.6 Consultar top 3 de hostales	14
1.7 Registrar estadía	15
1.8 Finalizar Estadía	16
1.9 Calificar estadía	18
1.10 Comentar calificación	19
1.11 Consulta de Usuario	20
1.12 Consulta de Hostal	21
1.13 Consulta de Reserva	23
1.14 Consulta de Estadía	24
1.15 Baja de reserva	27
2 Criterios Generales	28
<b>Ejercicio 3</b>	<b>29</b>
<b>Ejercicio 4</b>	<b>31</b>
<b>Ejercicio 5</b>	<b>32</b>

# Ejercicio 1

Con los nuevos cambios y funcionalidades agregados al sistema fue necesario a calificación agregarle un atributo, *suscriptores: Set(Empleado)*, con el objetivo de llevar la cuenta de los empleados suscritos a las notificaciones, para más adelante utilizar el patrón observer (se detallará más adelante). Si fue necesario crear los nuevos DSS de los casos de los siguientes casos de uso: suscribirse a notificaciones, consulta de notificaciones, eliminar suscripción y modificar la fecha del sistema. Si bien existe la posibilidad de agregar un nuevo concepto en el modelo de dominio, notificación, el cual contenga la información de dicha notificación, optamos por no agregar dicho concepto, obteniendo la información necesaria a través de asociaciones ya hechas, para así no agregarle complejidad al modelo.

Luego, se modificaron ciertos DSS de sin memoria a con memoria, con los objetivos de dejar más claros los diagramas, facilitar los diagramas de comunicación y mejorar el pasaje de información. Dichos DSS fueron pasados con anterioridad al tutor vía email, siendo estos: alta de habitación, asignar empleado a hostal, finalizar estadía, calificar estadía, comentar calificación, consulta de reserva, y consulta de estadía.

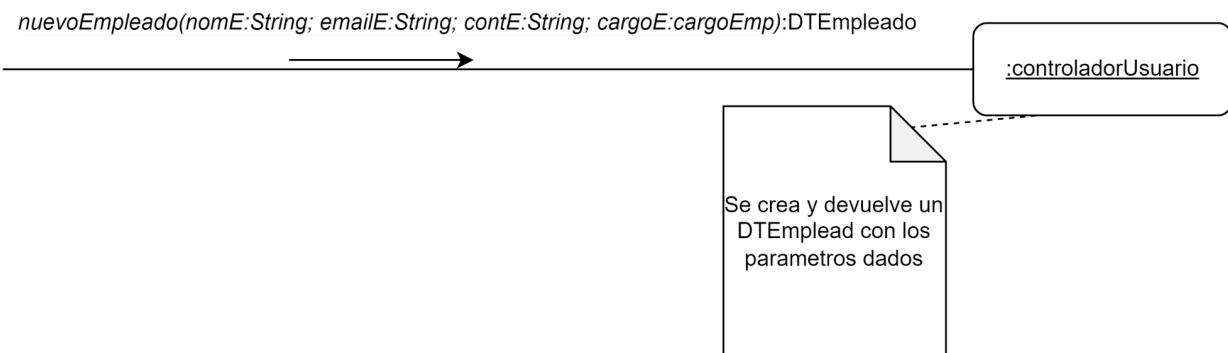
Por último, se realizaron algunos cambios mínimos para mantener cierta consistencia. Por ejemplo, la operación la cual cancela el alta de un hostal, pasó de llamarse *CancelarAlta(DTHostal)* a llamarse *cancelarHostal(DTHostal)* o la operacion *RegistrarEstadia(Integer)* pasó a llamarse *registrarEstadia(Integer)*.

# Ejercicio 2

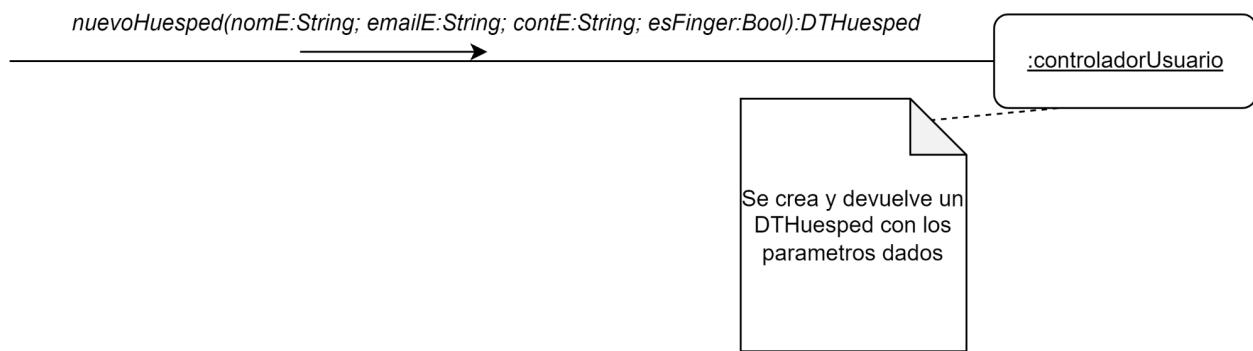
## 1 Realización de Casos de Uso (Diagramas de comunicación)

### 1.1 Alta de Usuario

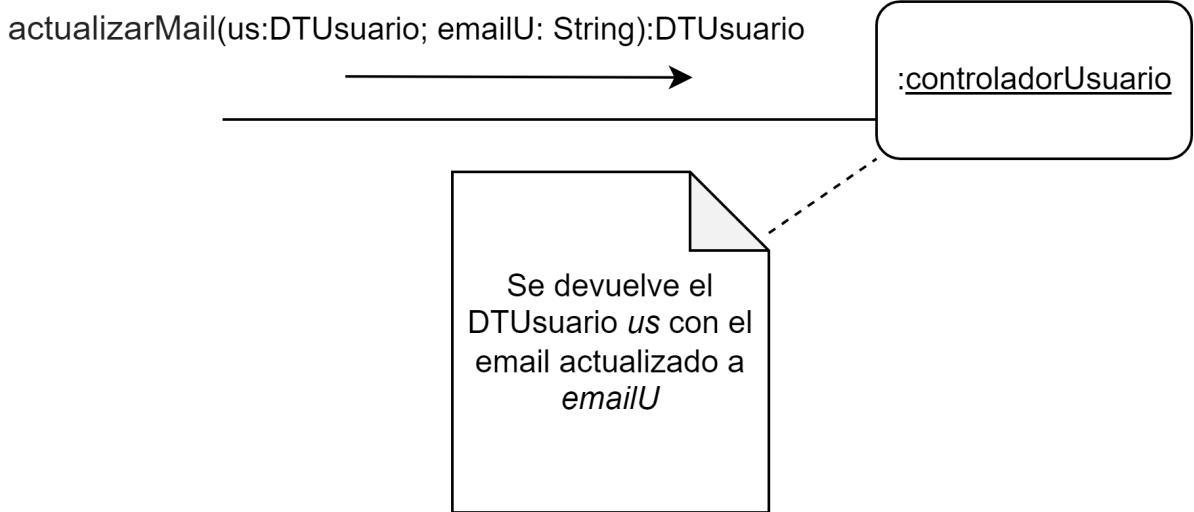
**Diagrama de comunicación de la operación** nuevoEmpleado(String; String; String; cargoEmp):DTEmplead



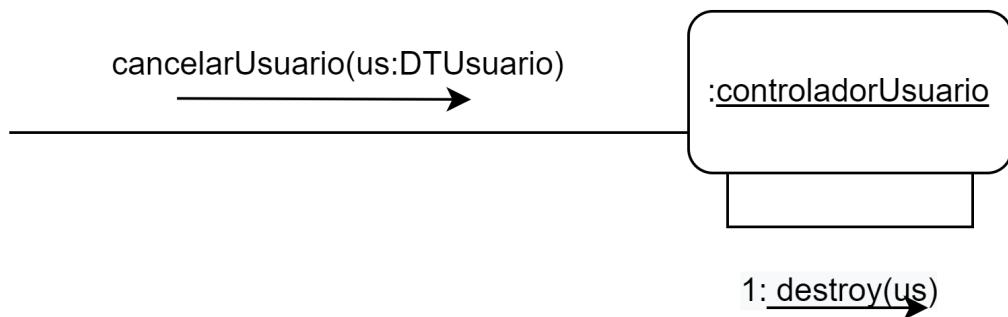
**Diagrama de comunicación de la operación** nuevoHuesped(String; String; String; boolean):DTHuesped



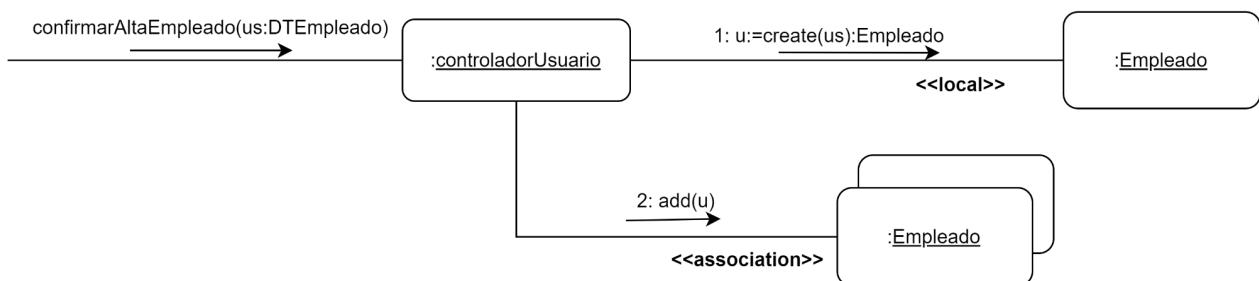
### Diagrama de comunicación de la operación actualizarMail(DTUsuario; String):DTUsuario



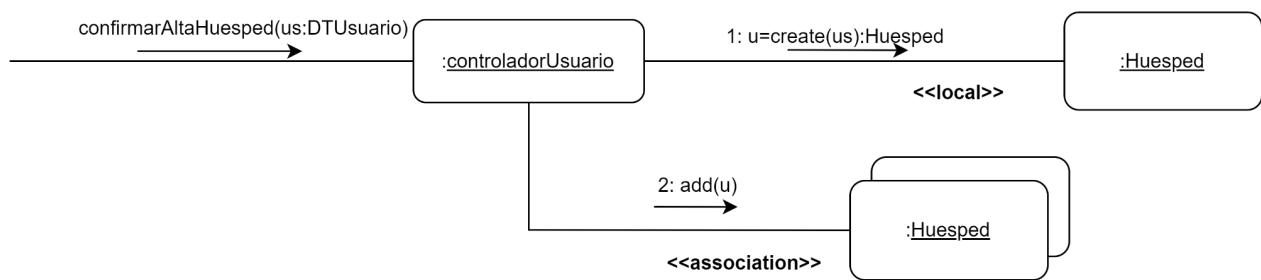
### Diagrama de comunicación de la operación cancelarUsuario(DTUsuario)



### Diagrama de comunicación de la operación confirmarAltaEmpleado(DTEmppleado)

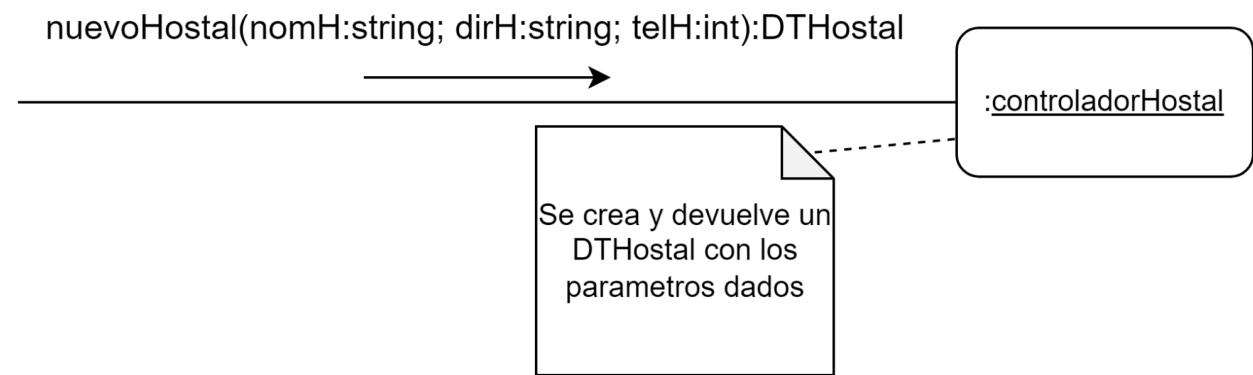


## Diagrama de comunicación de la operación confirmarAltaHuesped(DTHuesped)

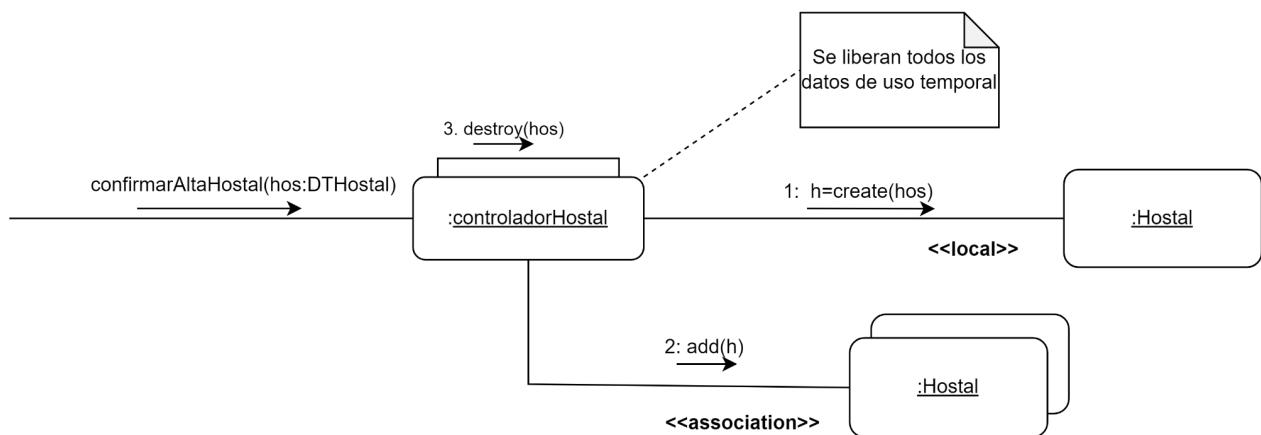


## 1.2 Alta de Hostal

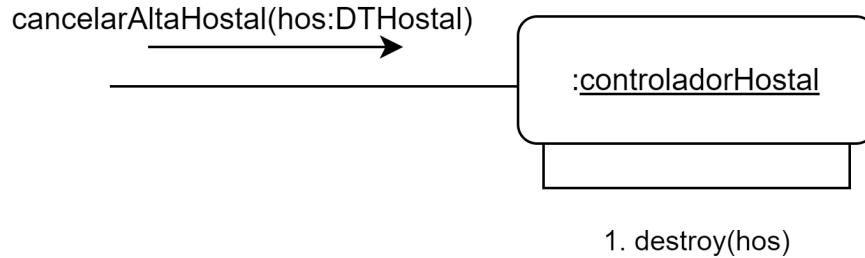
### Diagrama de comunicación de la operación nuevoHostal(String; String; integer):DTHostal



### Diagrama de comunicación de la operación confirmarHostal(DTHostal)

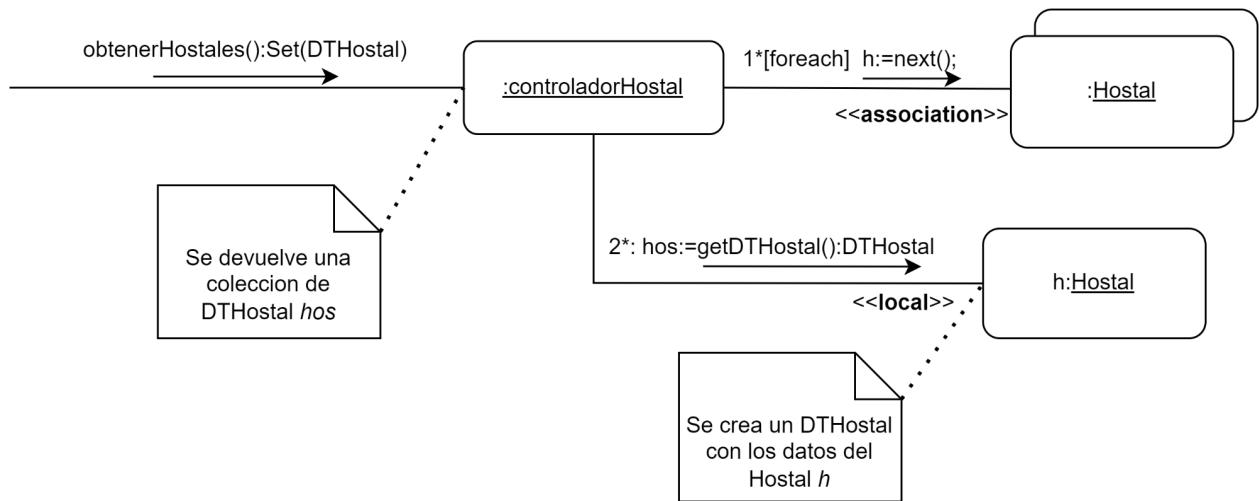


## Diagrama de comunicación de la operación cancelarHostal(DTHostal)

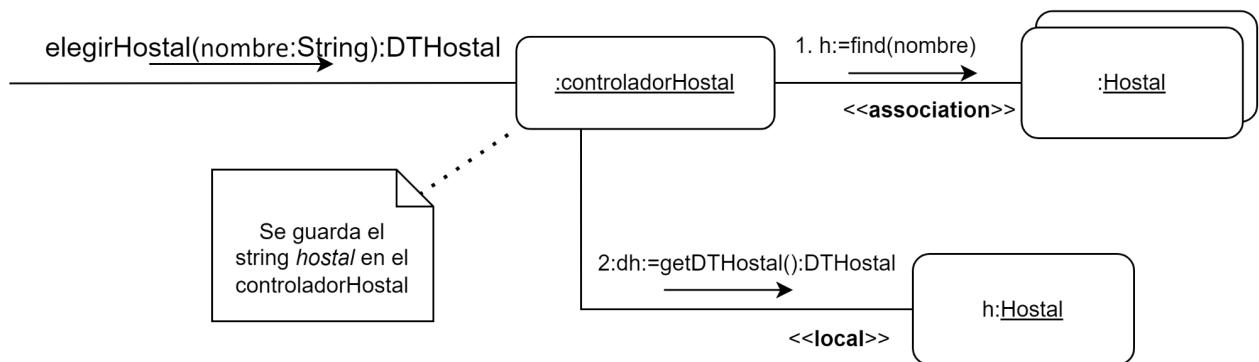


## 1.3 Alta de Habitación

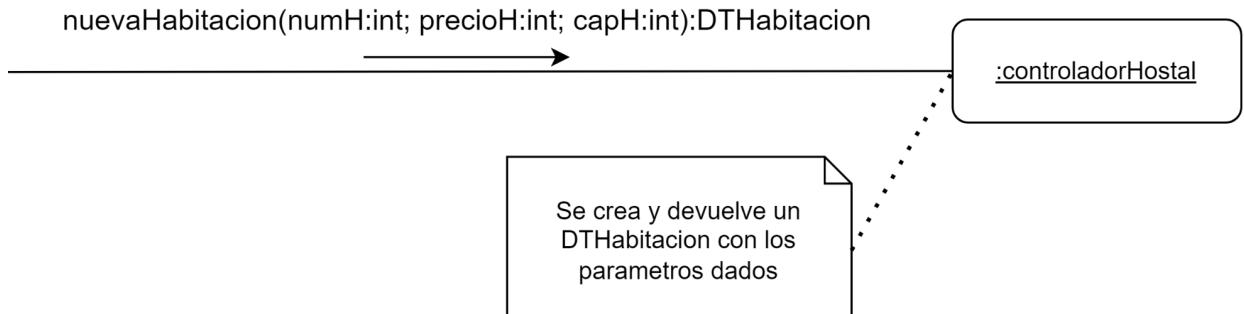
### Diagrama de comunicación de la operación obtenerHostales():Set(DTHostal)



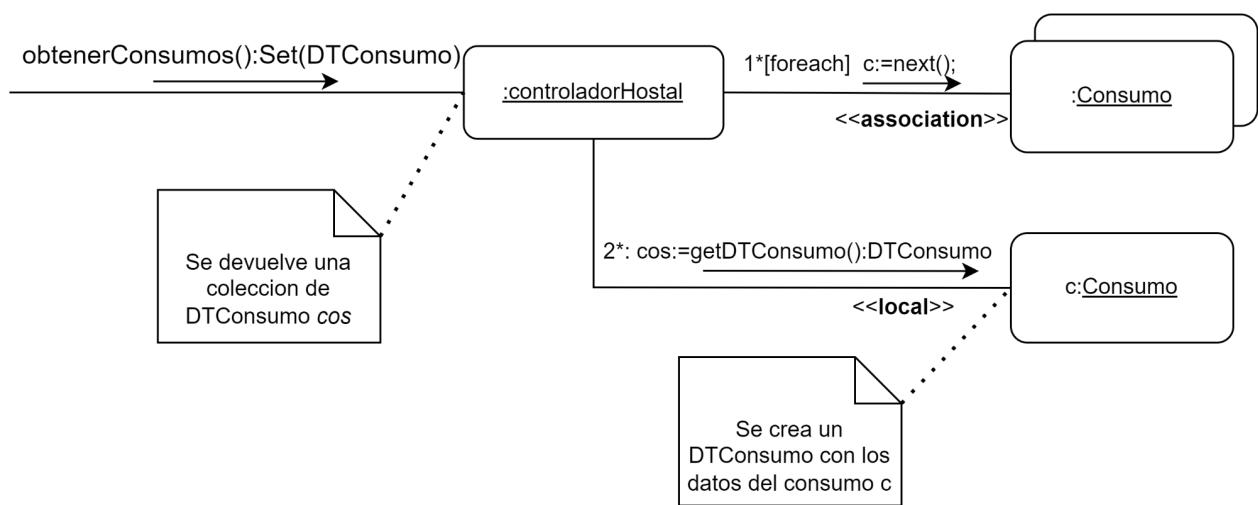
### Diagrama de comunicación de la operación elegirHostal(String):DTHostal



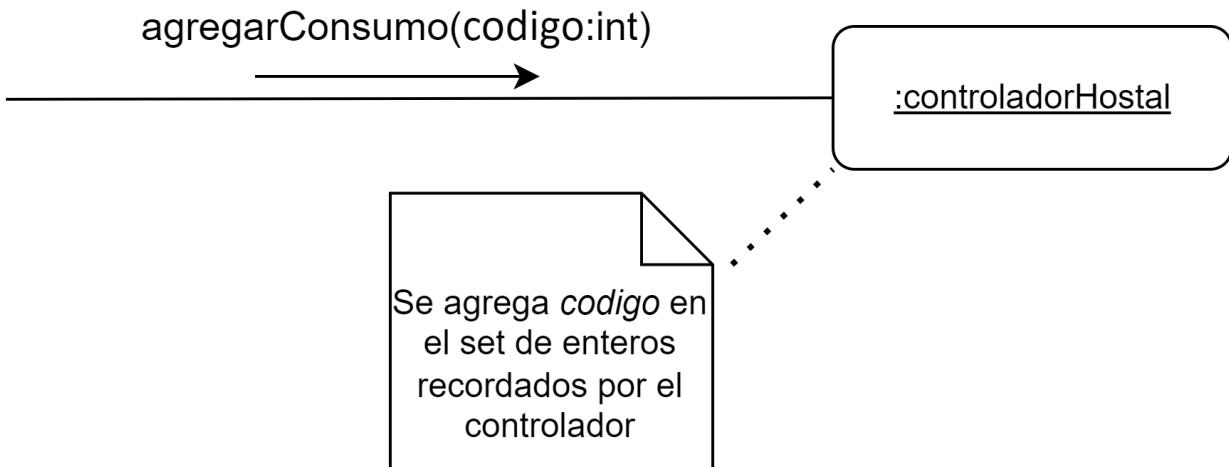
**Diagrama de comunicación de la operación** nuevaHabitacion(integer; integer; integer) : DTHabitacion



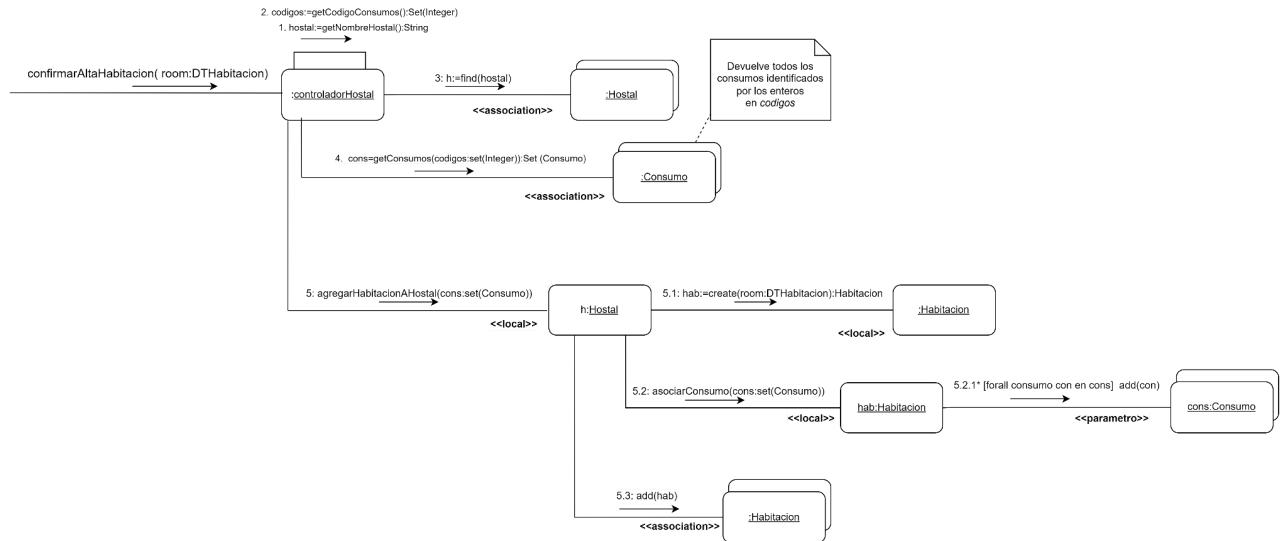
**Diagrama de comunicación de la operación** obtenerConsumos():Set(DTConsumo)



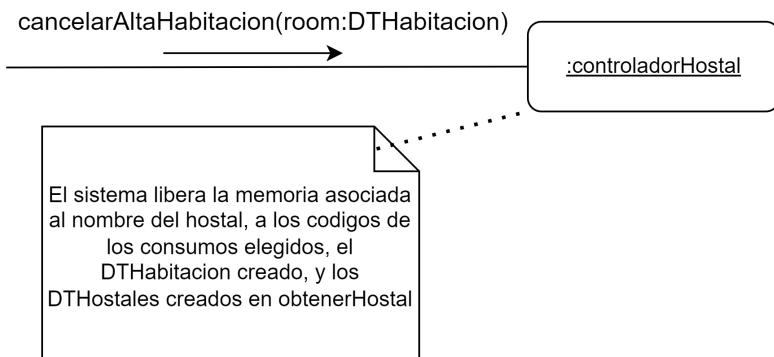
**Diagrama de comunicación de la operación** agregarConsumos(Integer)



## Diagrama de comunicación de la operación confirmarAltaHabitacion(DTHabitacion)

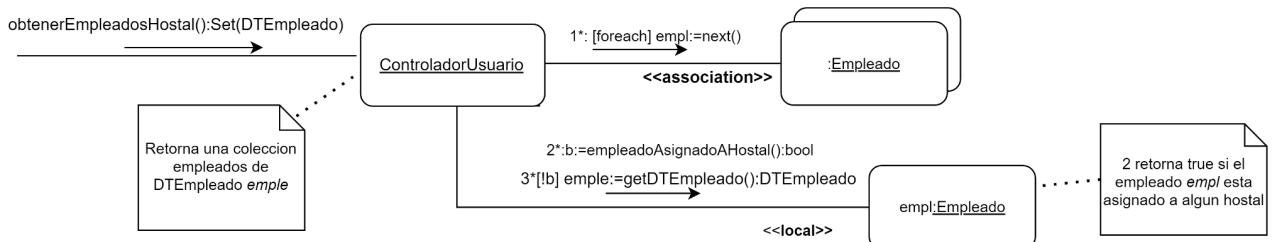


## Diagrama de comunicación de la operación cancelarAltaHabitacion(DTHabitacion)

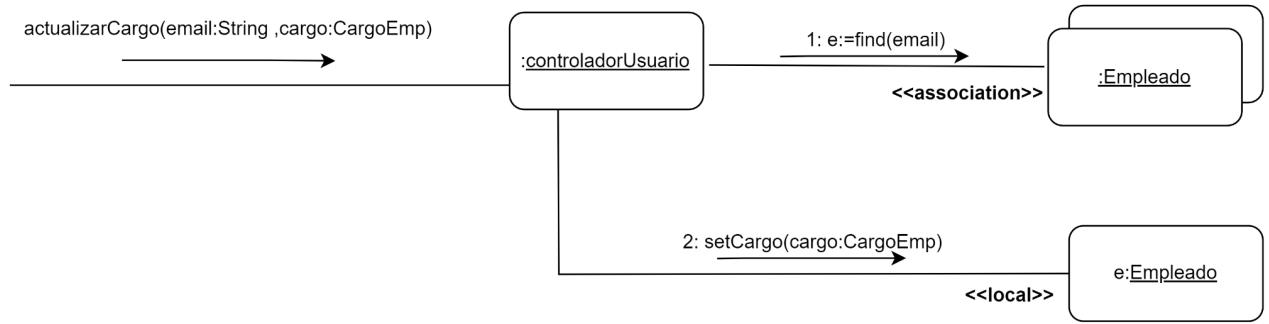


## 1.4 Asignar empleado a hostal

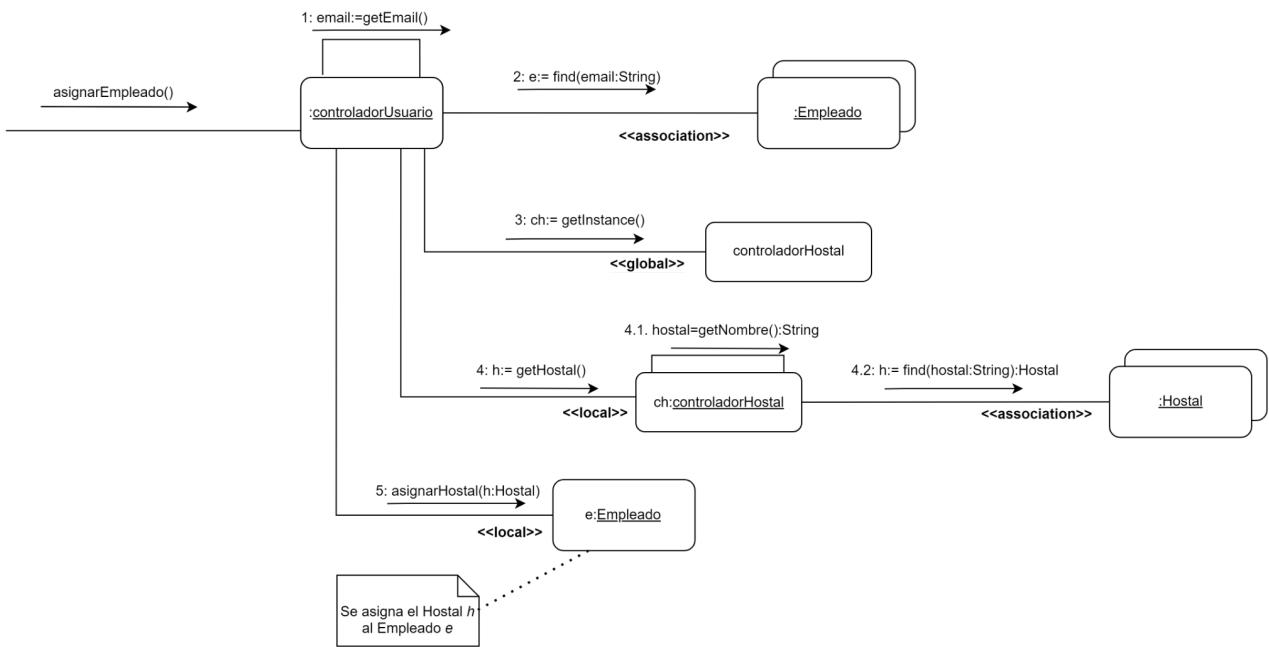
### Diagrama de comunicación de la operación obtenerEmpleadosHostal():Set(DTEmppleado)



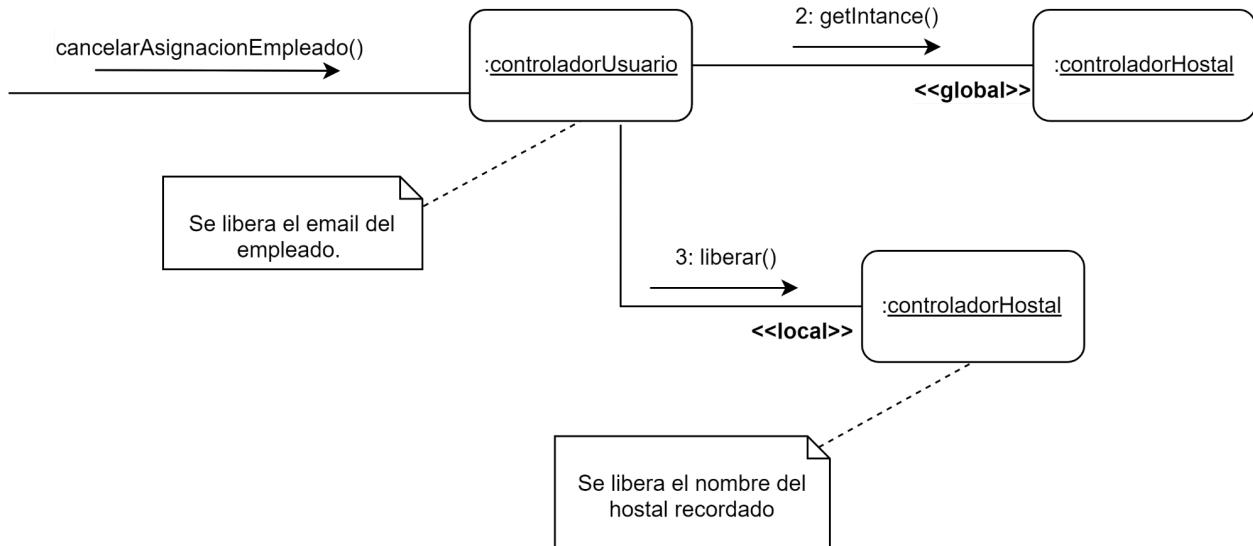
## Diagrama de comunicación de la operación actualizarCargo(String, CargoEmp)



## Diagrama de comunicación de la operación asignarEmpleado()

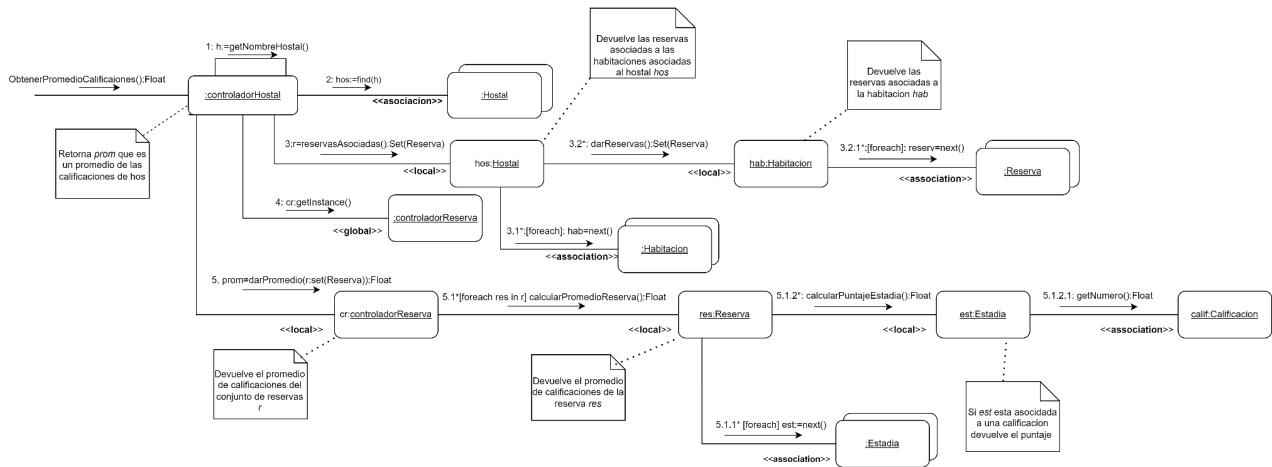


## Diagrama de comunicación de la operación cancelarAsignacionEmpleado()

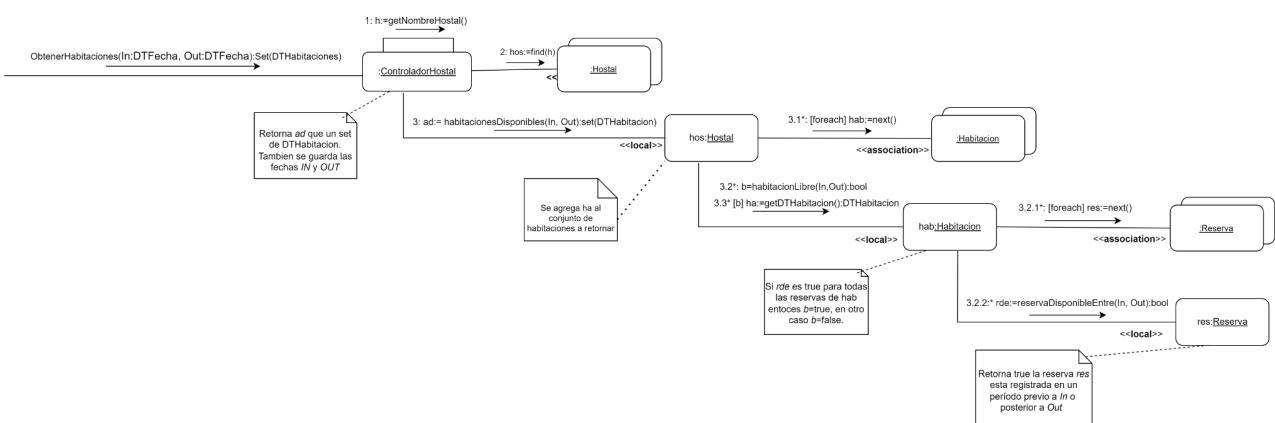


## 1.5 Realizar Reserva

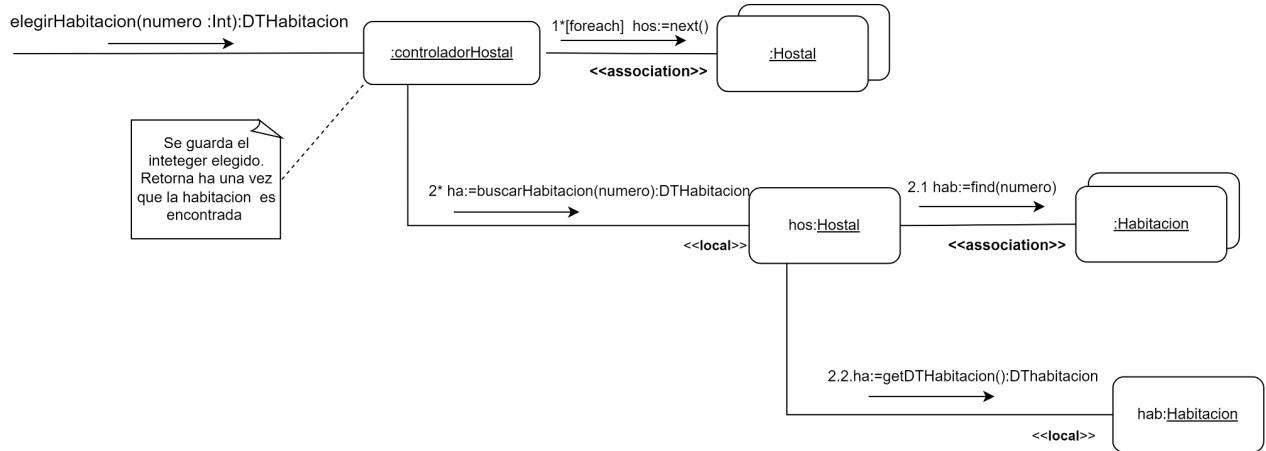
### Diagrama de comunicación de la operación obtenerPromedioCalificaciones():Float



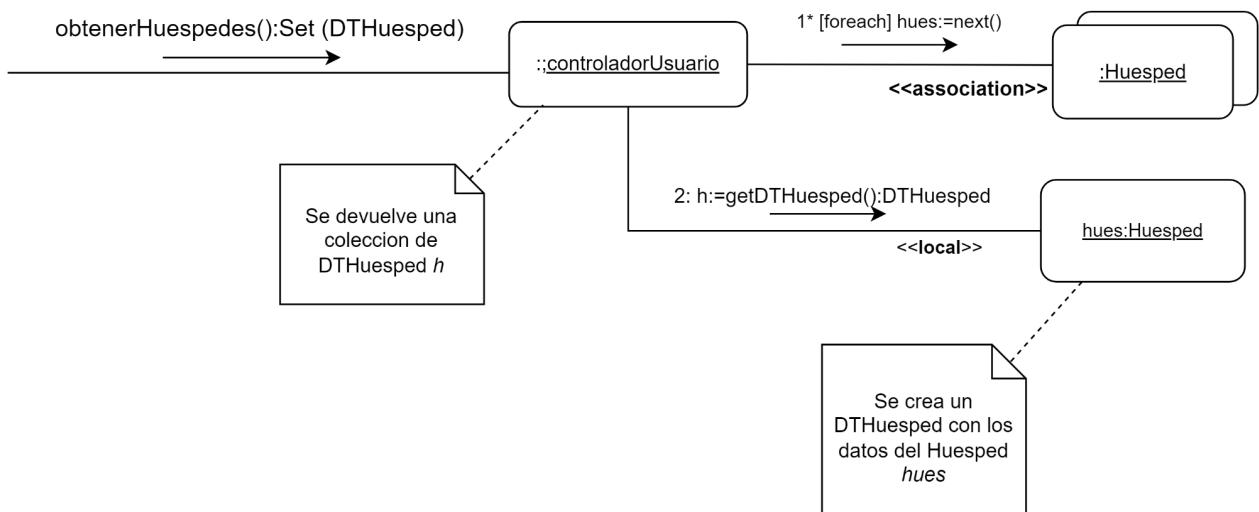
### Diagrama de comunicación de la operación obtenerHabitaciones(DTFecha, DTFecha):Set(DTHabitacion)



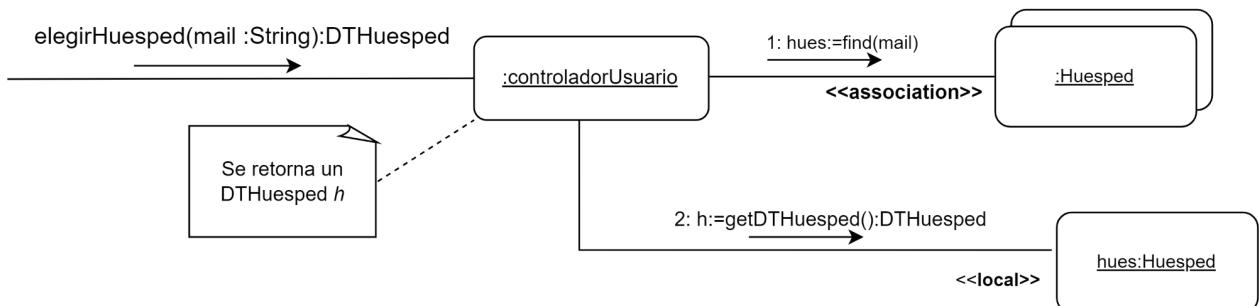
## Diagrama de comunicación de la operación elegirHabitacion(integer):DTHabitacion



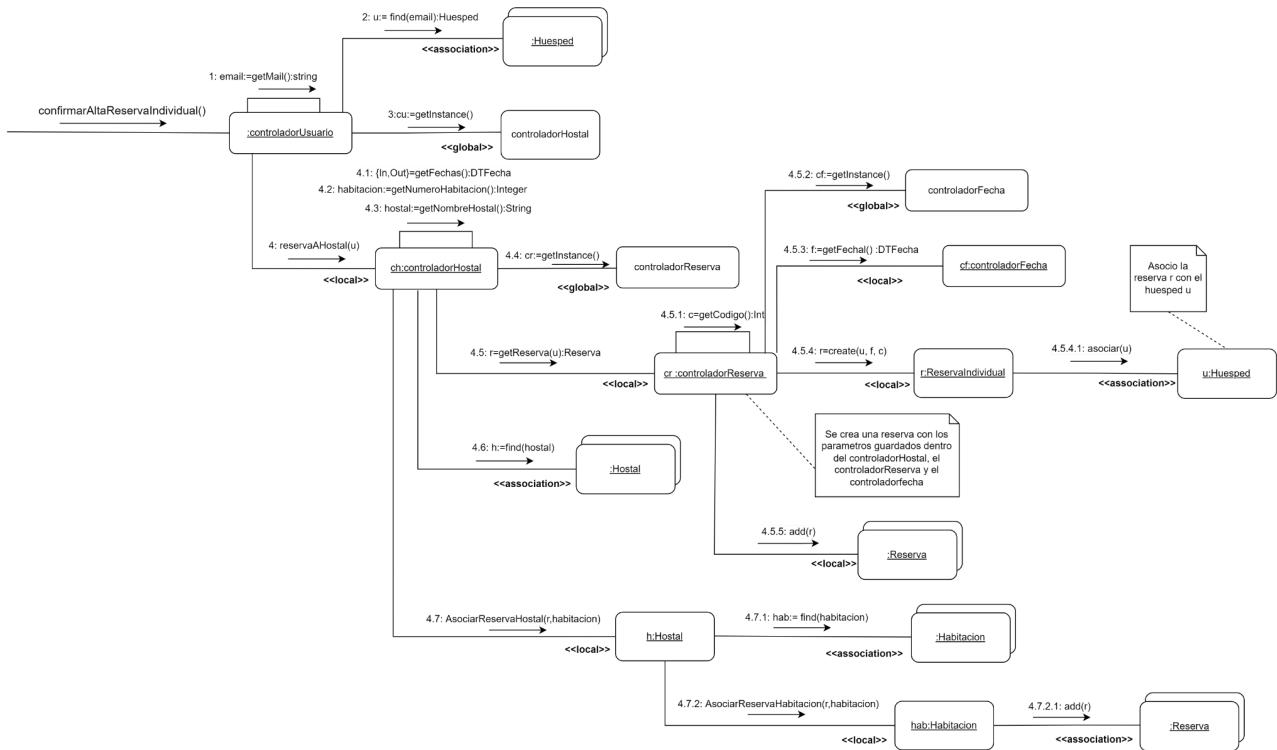
## Diagrama de comunicación de la operación obtenerHuespedes():Set (DTHuesped)



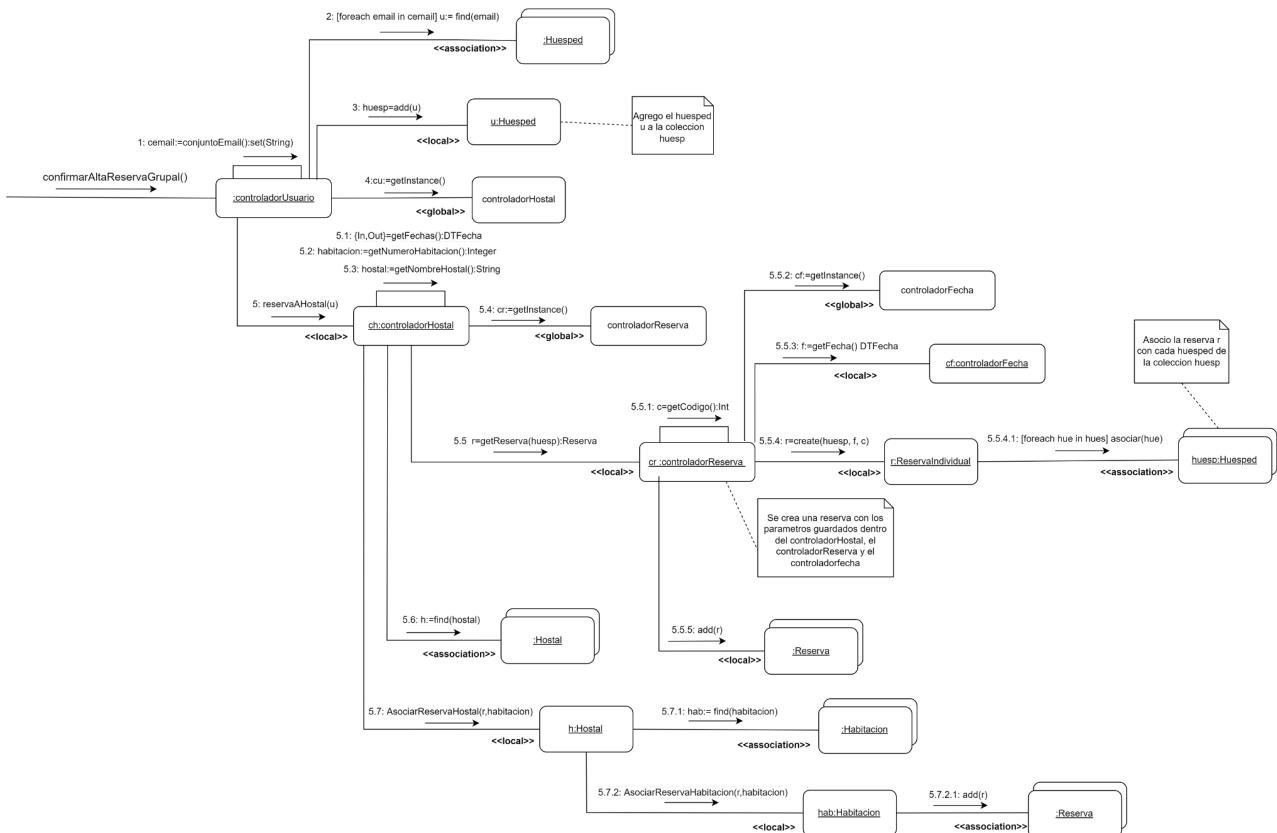
## Diagrama de comunicación de la operación elegirHuesped(String):DTHuesped



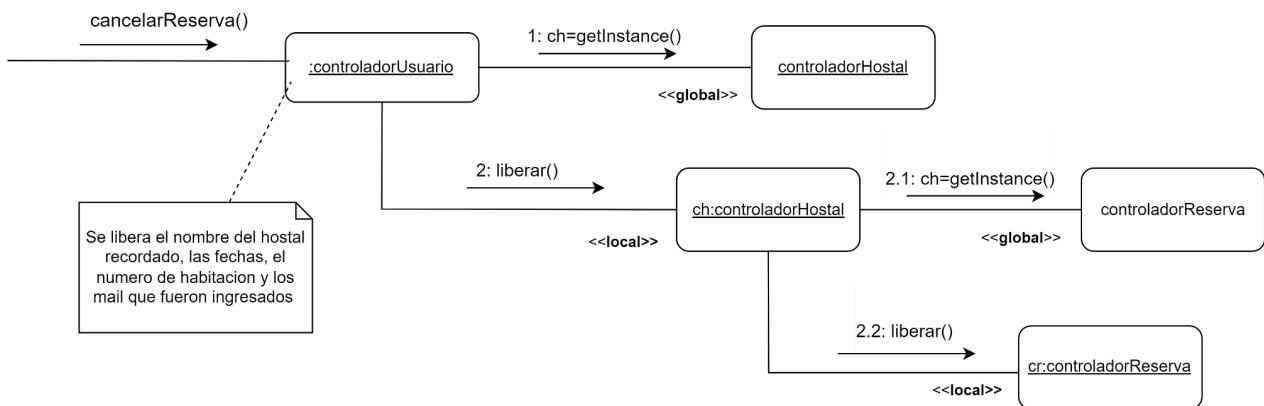
## Diagrama de comunicación de la operación confirmarAltaReservaIndividual()



## Diagrama de comunicación de la operación confirmarAltaReservaGrupal()

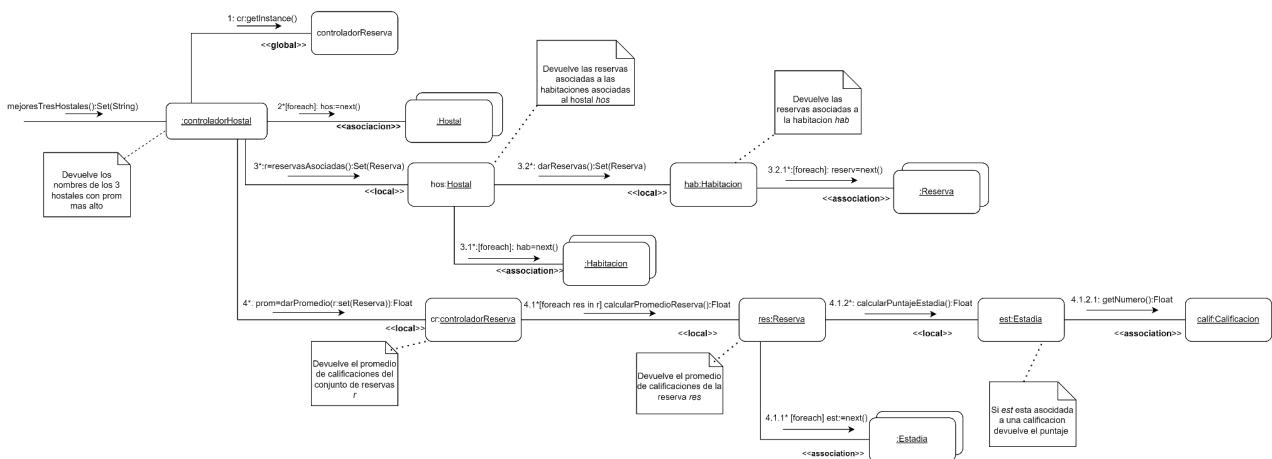


### Diagrama de comunicación de la operación cancelarReserva()

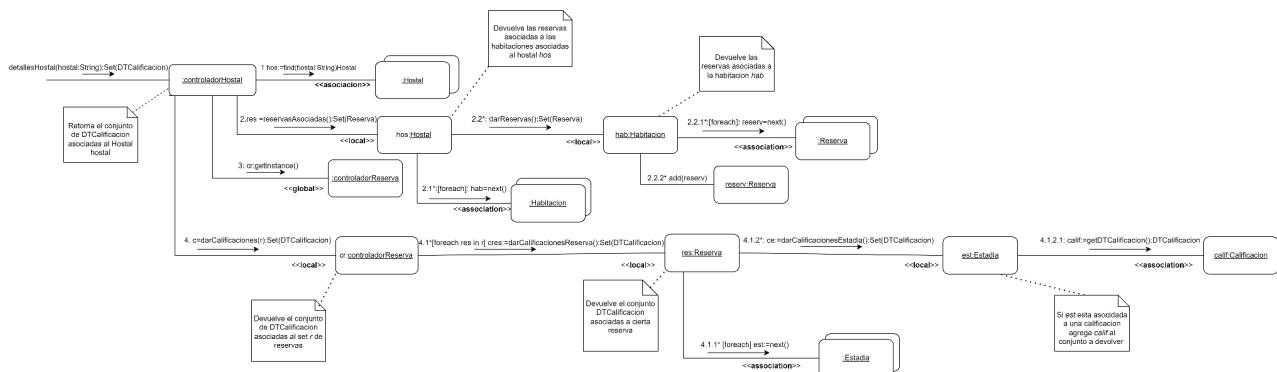


## 1.6 Consultar top 3 de hostales

## Diagrama de comunicación de la operación mejoresTresHostales():Set<String>

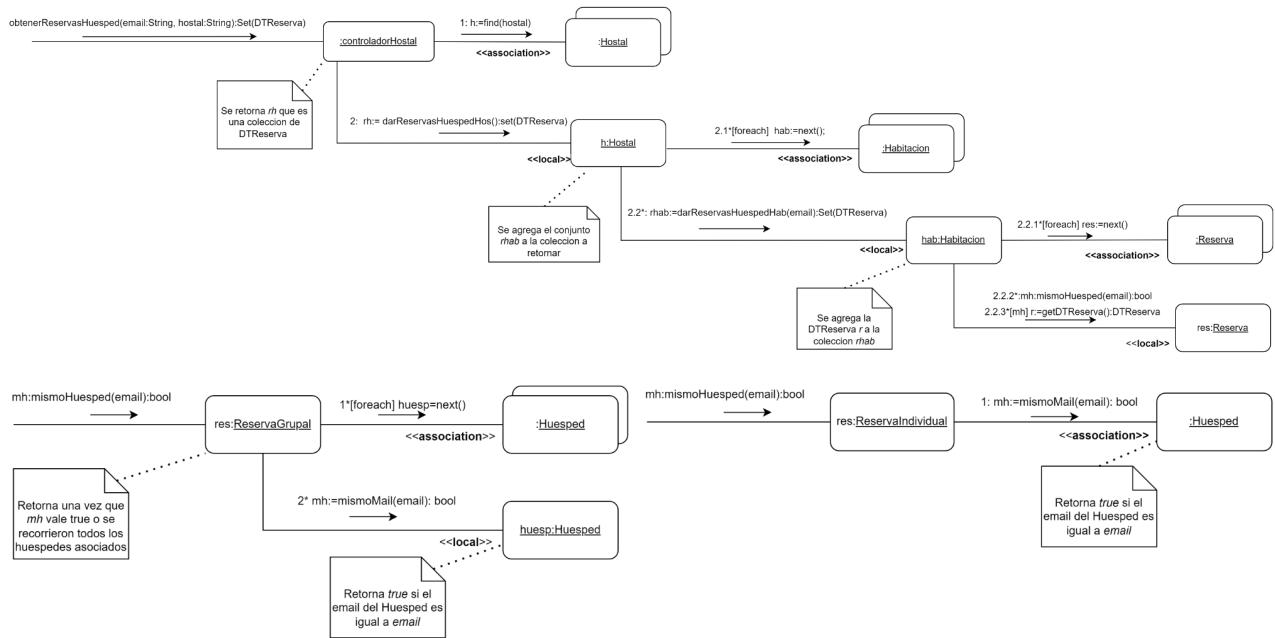


**Diagrama de comunicación de la operación** detallesHostal(String): Set(DTCalifcacion)

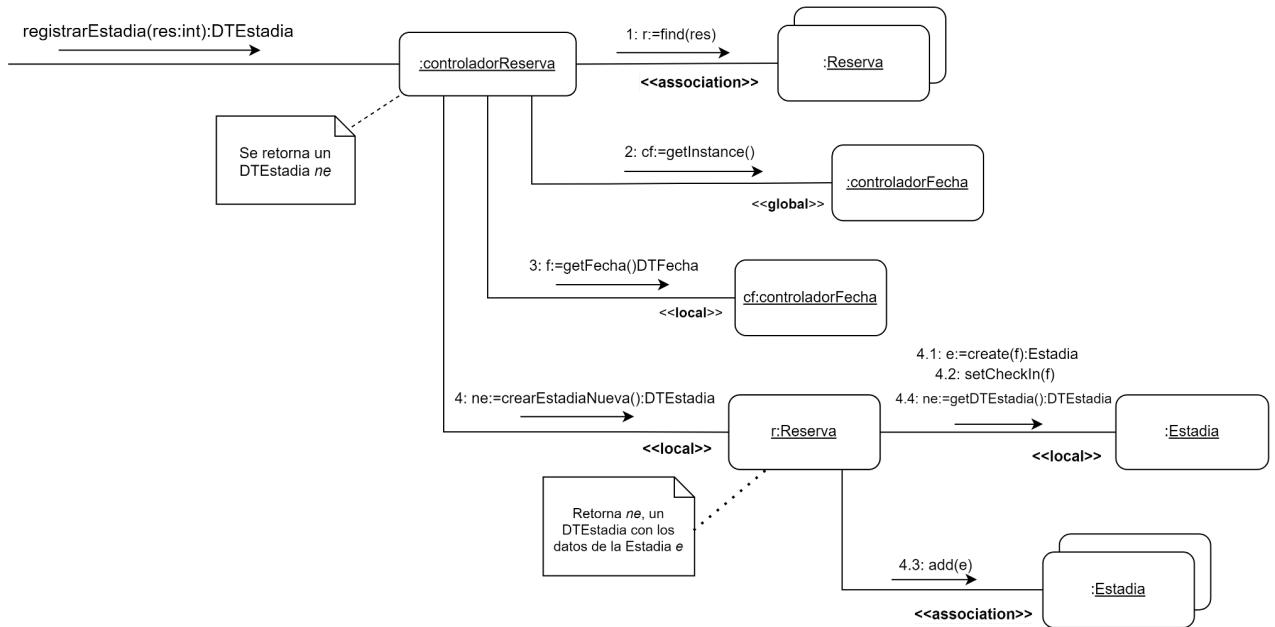


## 1.7 Registrar estadía

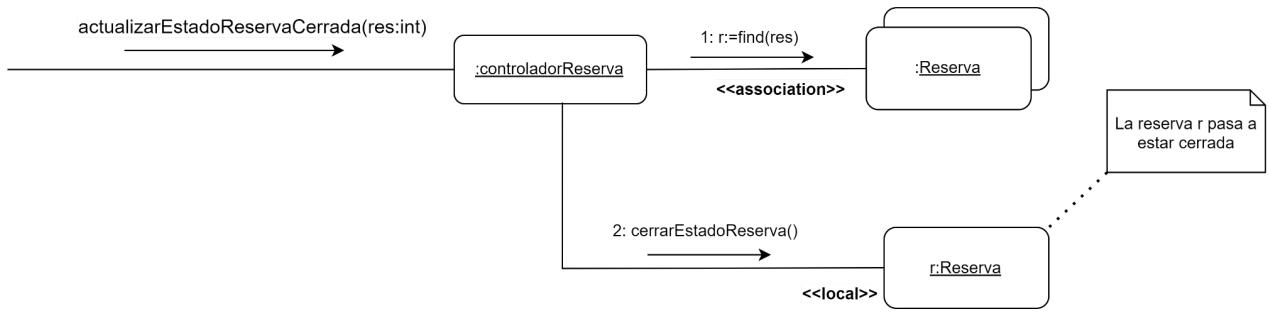
### **Diagrama de comunicación de la operación obtenerReservasHuesped(String, String):Set(DTReserva)**



## Diagrama de comunicación de la operación registrarEstadia(integer):DTEstadia

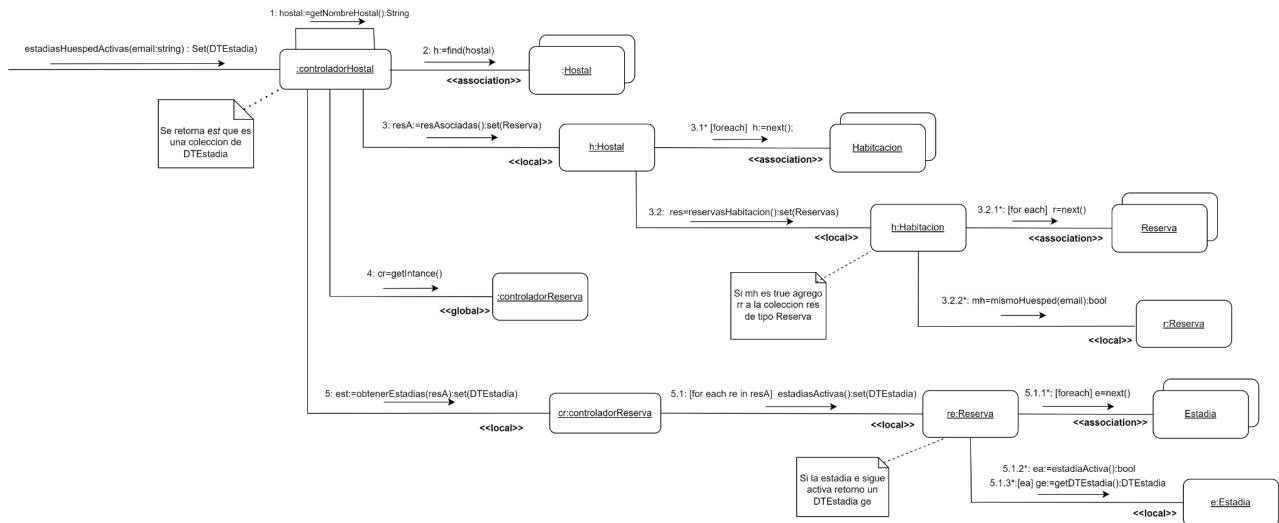


## Diagrama de comunicación de la operación actualizarEstadoReservaCerrada(integer)

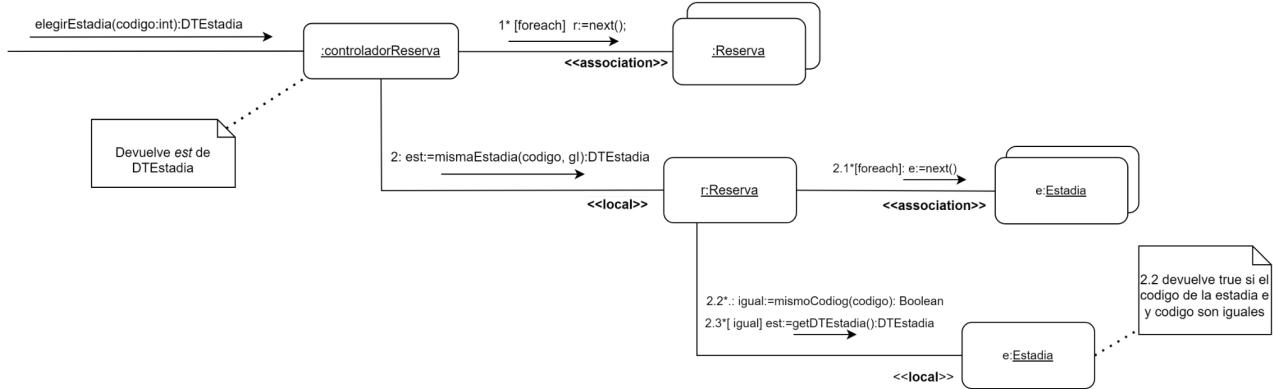


## 1.8 Finalizar Estadía

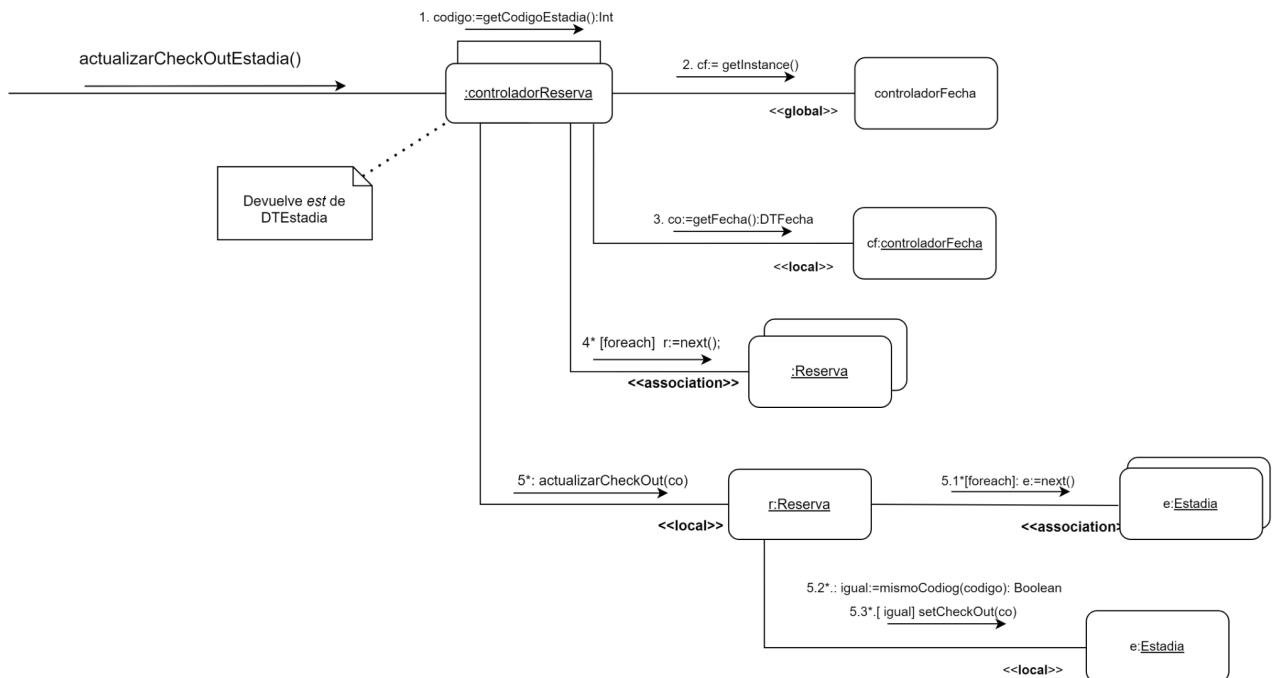
### Diagrama de comunicación de la operación estadiasHuespedActivas(String) : Set(DTEstadia)



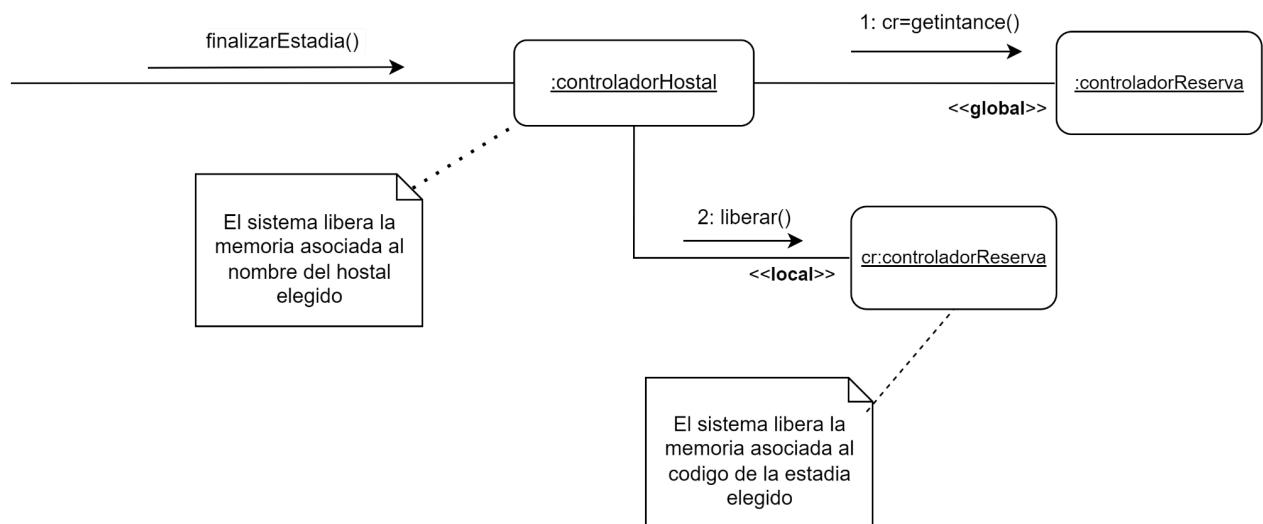
## Diagrama de comunicación de la operación elegirEstadia(integer):DTEstadia



## Diagrama de comunicación de la operación actualizarCheckOutEstadia(DTFecha)

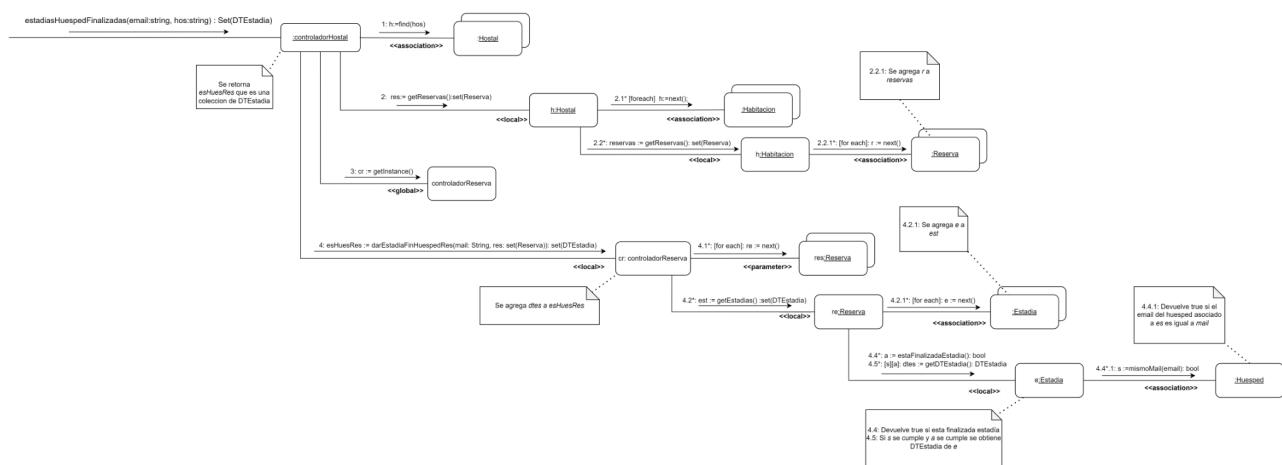


## Diagrama de comunicación de la operación finalizarEstadia()

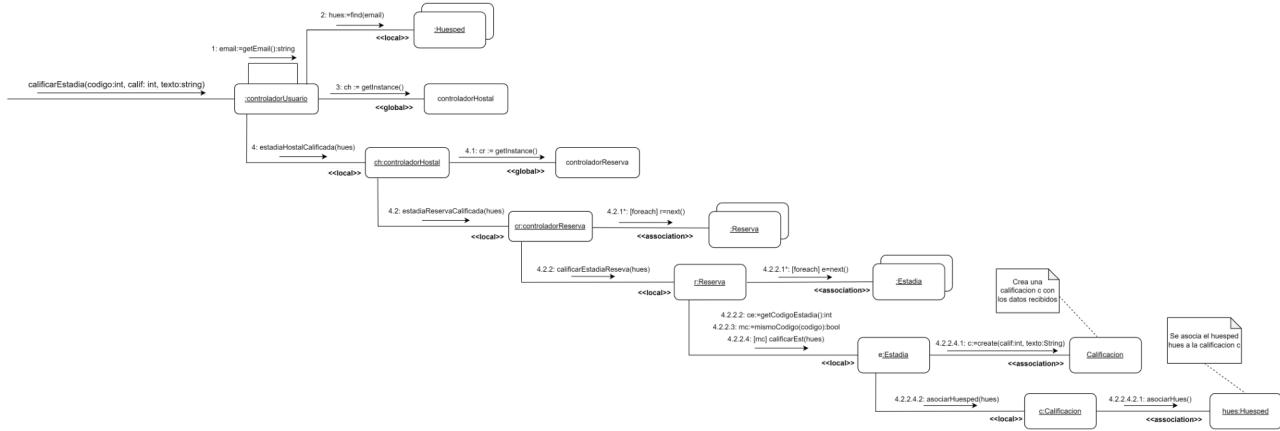


## 1.9 Calificar estadía

**Diagrama de comunicación de la operación** estadiasHuespedFinalizadas(String, String) : Set(DTEstadia)

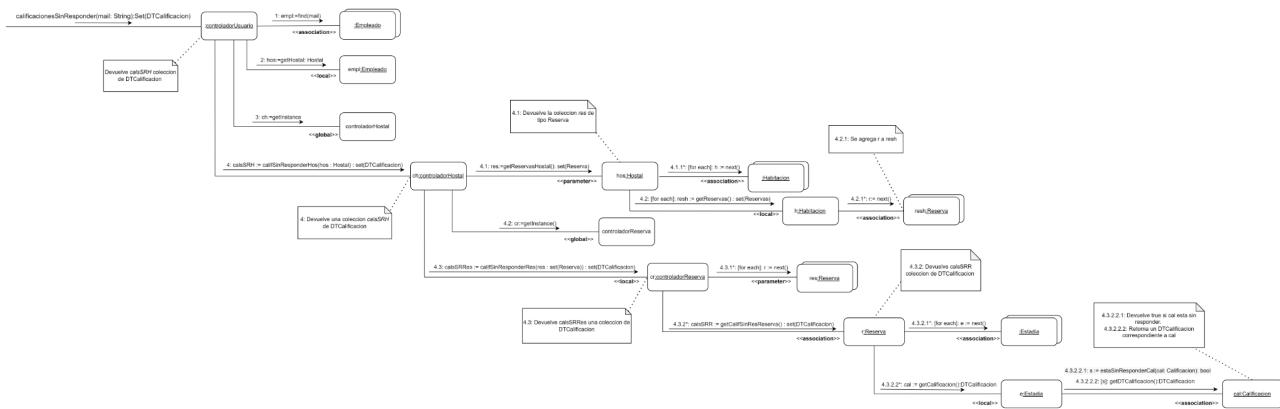


## Diagrama de comunicación de la operación calificarEstadia(integer, DTCalificacion, String)

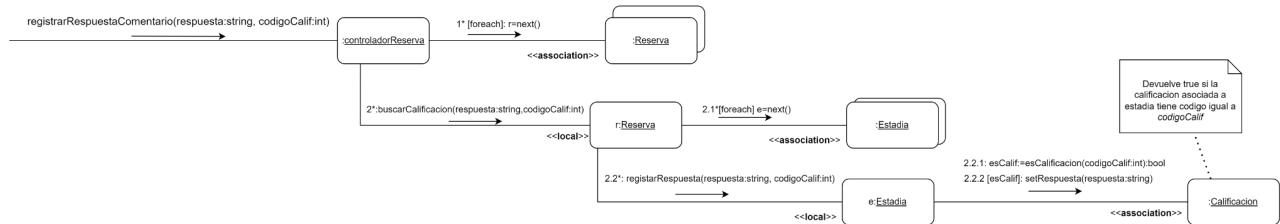


## 1.10 Comentar calificación

### Diagrama de comunicación de la operación calificacionesSinResponder(String): Set(DTCalificacion)

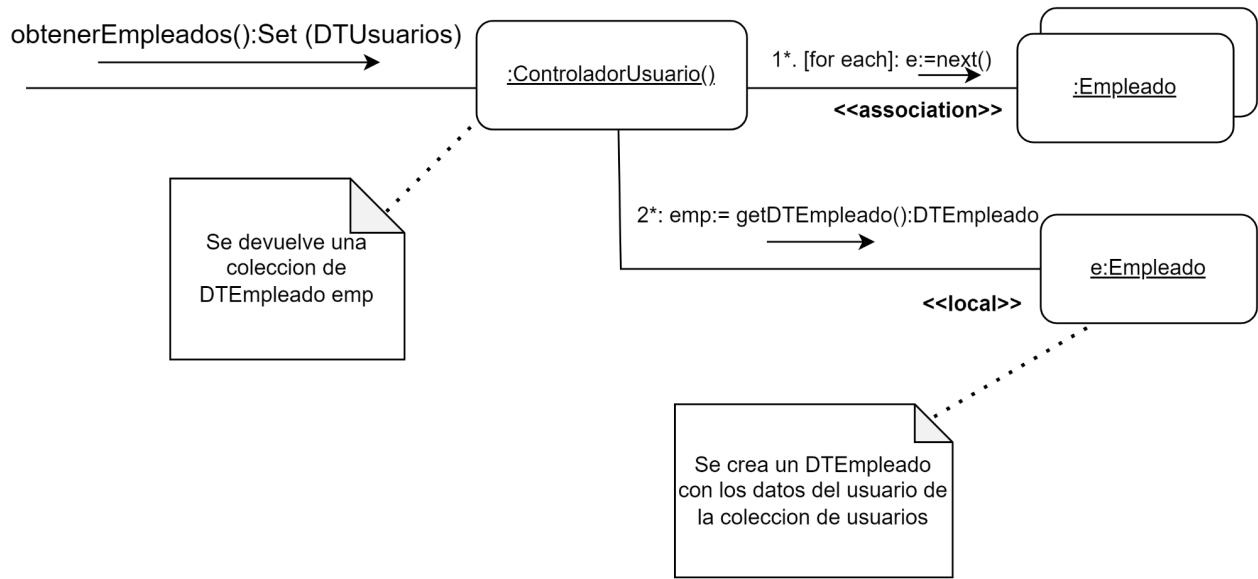


### Diagrama de comunicación de la operación registrarRespuestaComentario(String, integer)

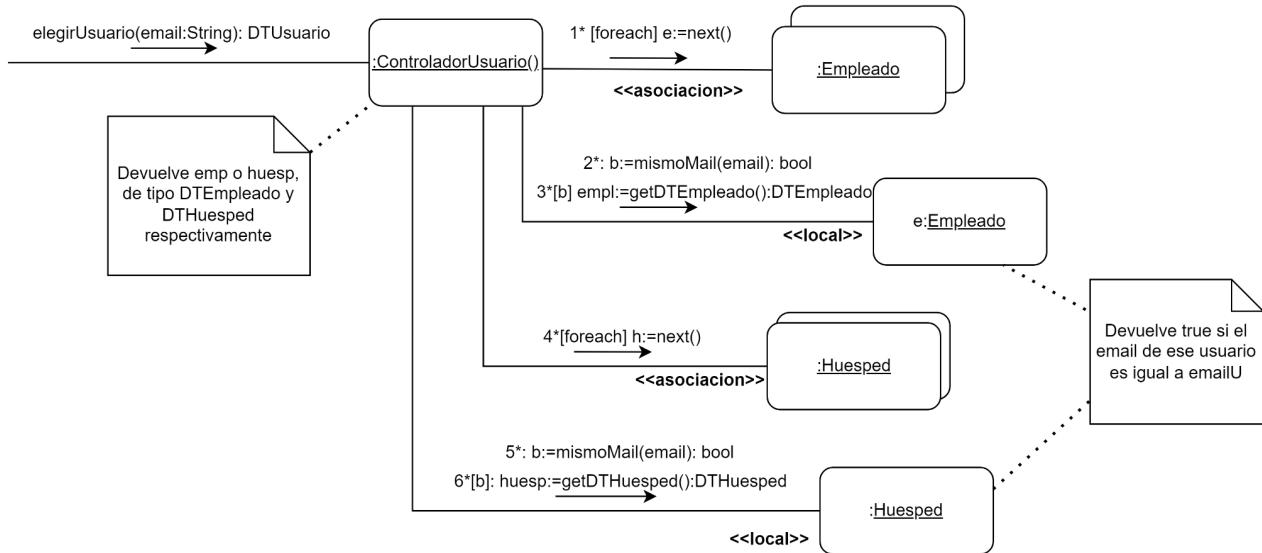


## 1.11 Consulta de Usuario

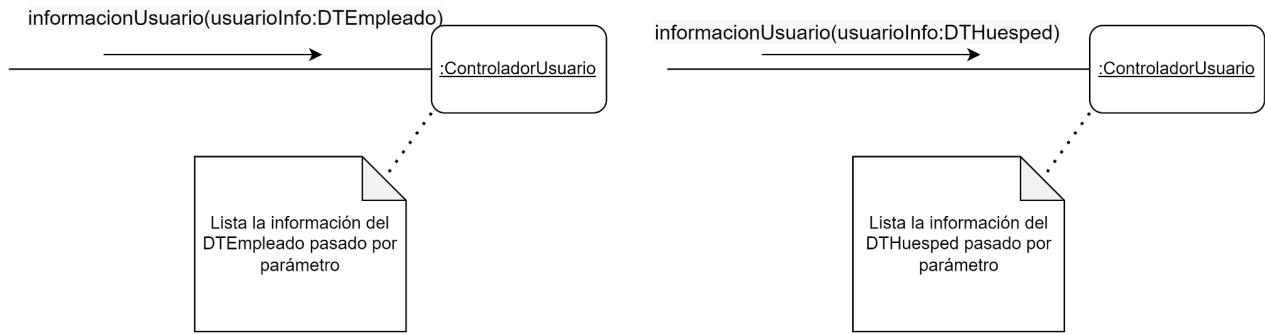
**Diagrama de comunicación de la operación obtenerEmpleados():Set(DTEmpelado)**



**Diagrama de comunicación de la operación elegirUsuario(String) : DTUsuario**

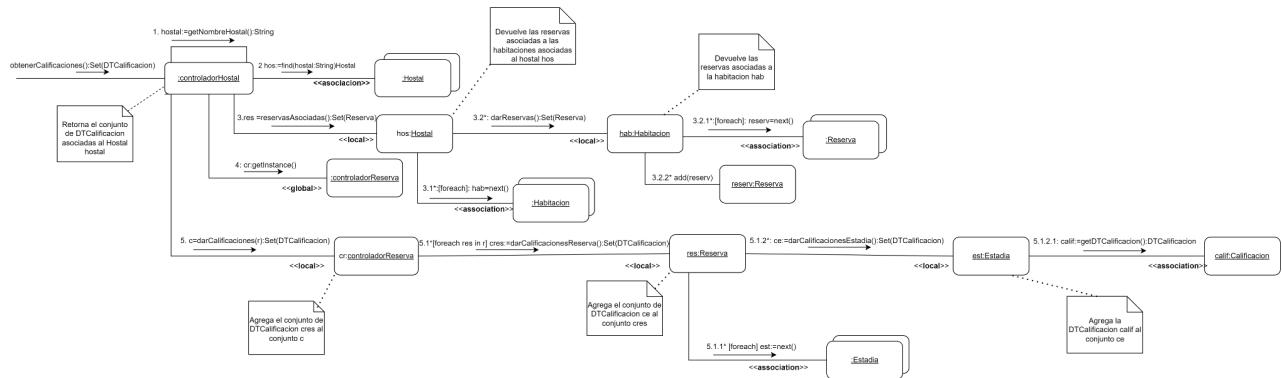


## Diagrama de comunicación de la operación informacionUsuario(DTUsuario)

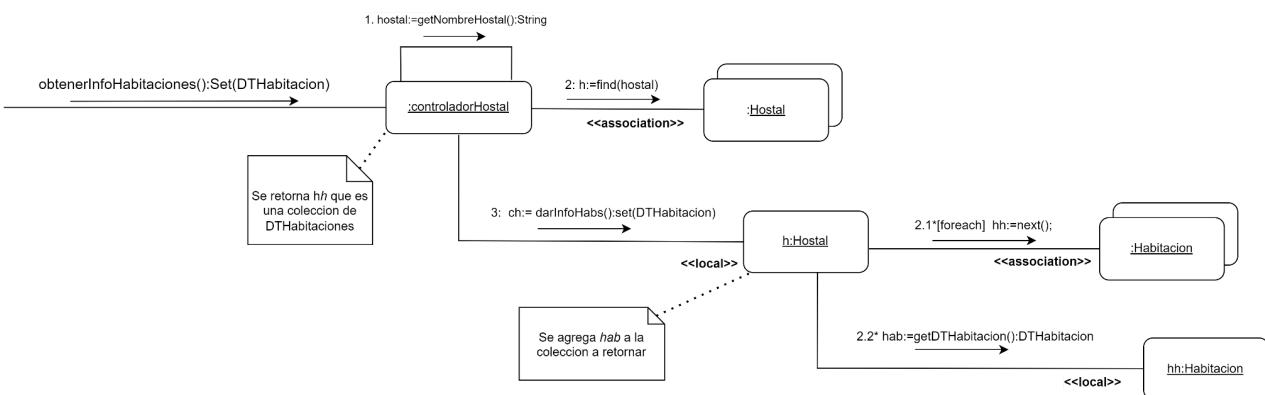


## 1.12 Consulta de Hostal

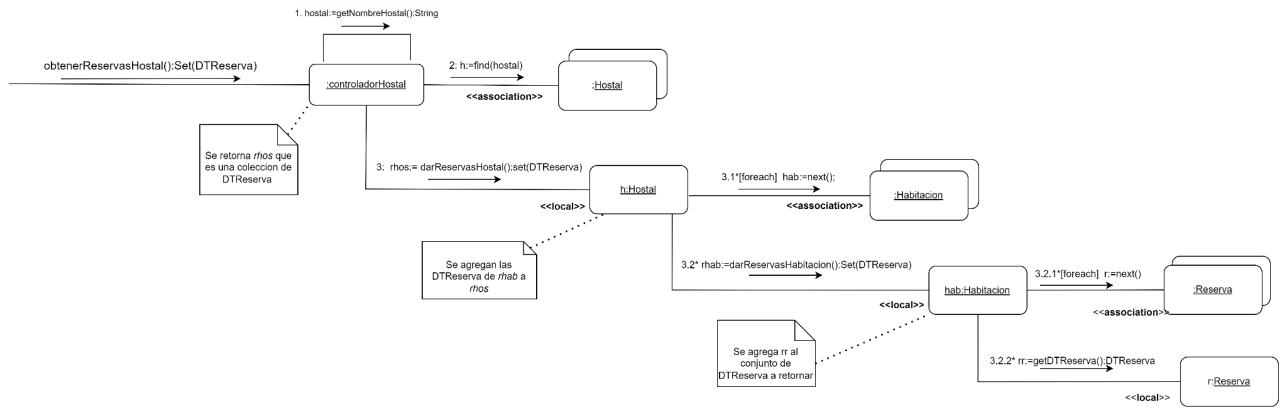
### Diagrama de comunicación de la operación obtenerCalificaciones():Set(DTCalificacion)



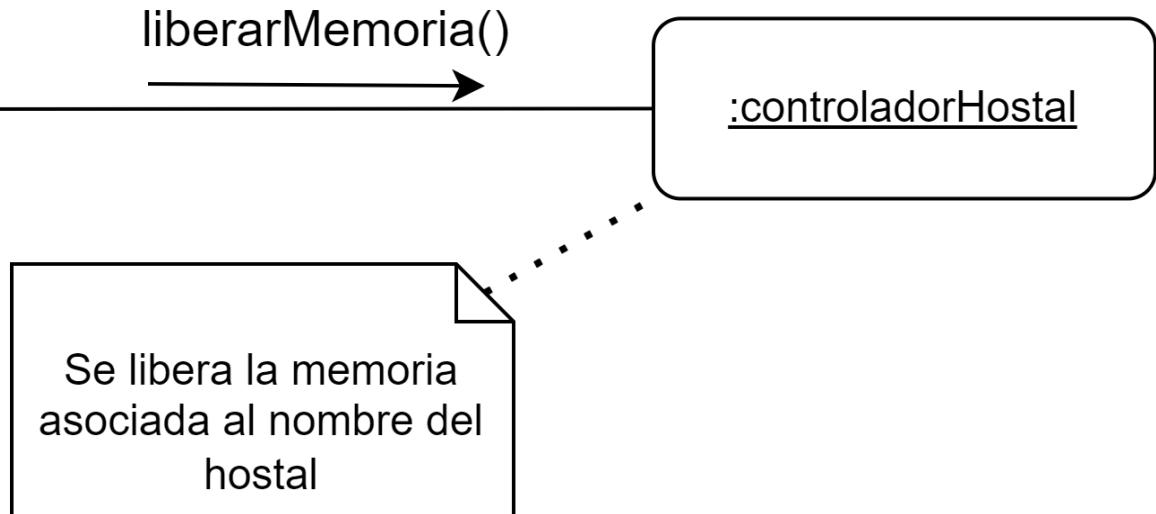
### Diagrama de comunicación de la operación obtenerInfoHabitaciones():Set(DTHabitacion)



## Diagrama de comunicación de la operación obtenerReservasHostal():Set(DTReserva)

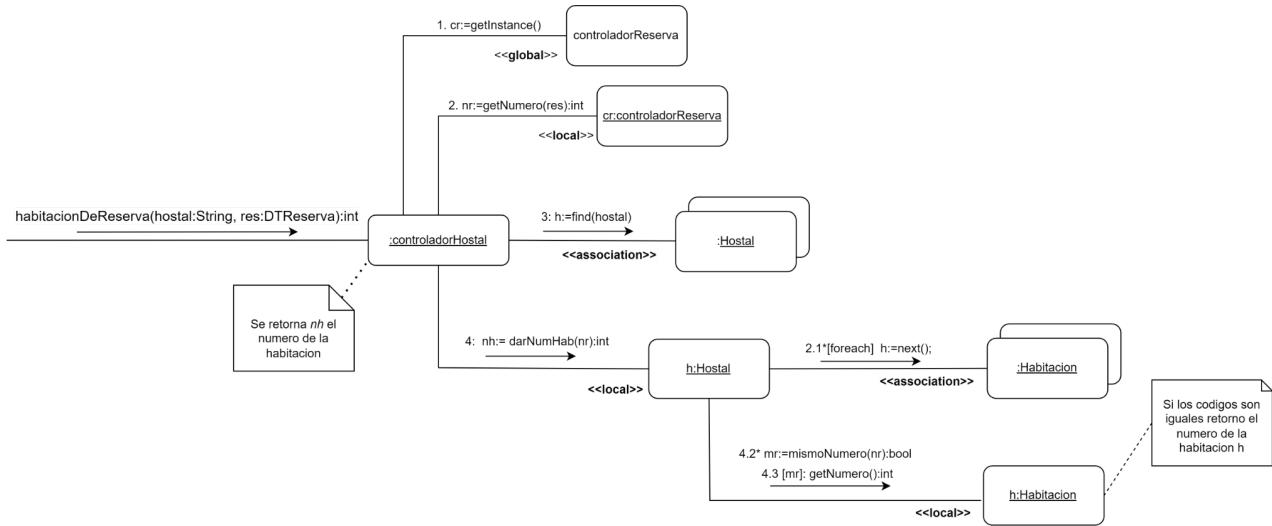


## Diagrama de comunicación de la operación liberarMemoria()

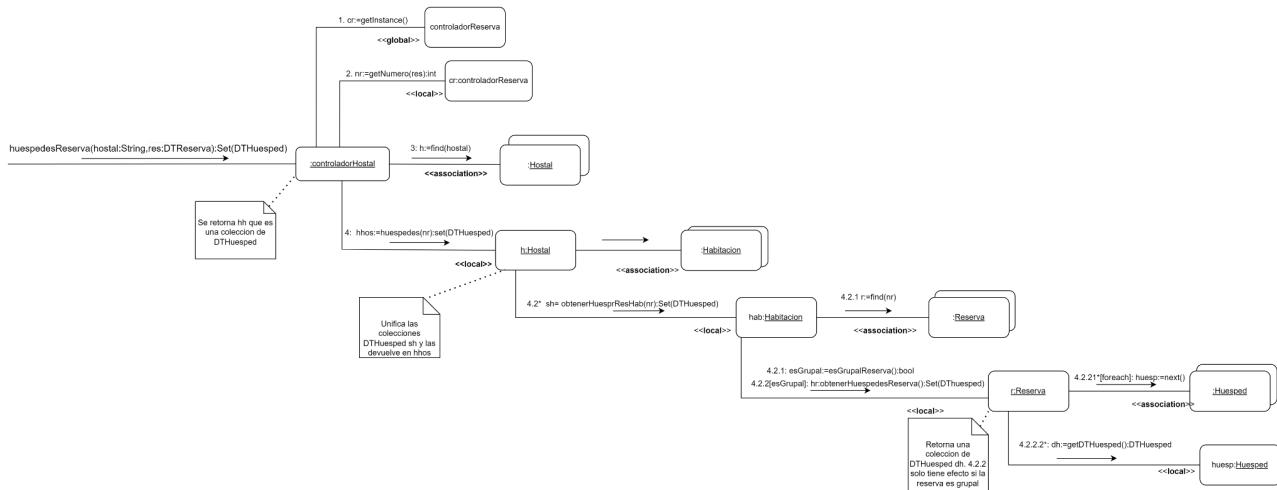


## 1.13 Consulta de Reserva

**Diagrama de comunicación de la operación habitacionDeReserva(String, DTReserva):integer**

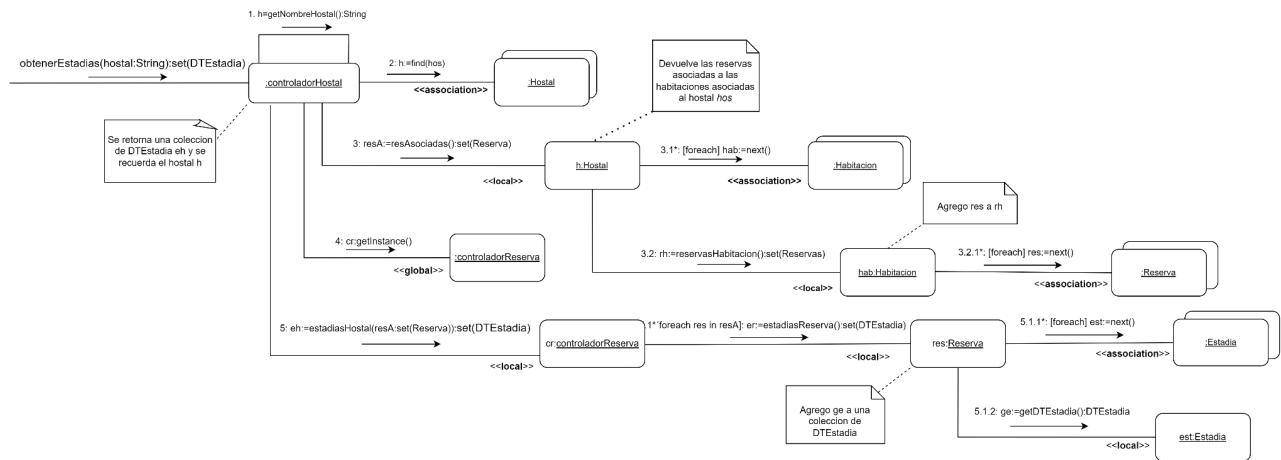


**Diagrama de comunicación de la operación huespedesReserva(String, DTReserva):Set(DTHuesped)**

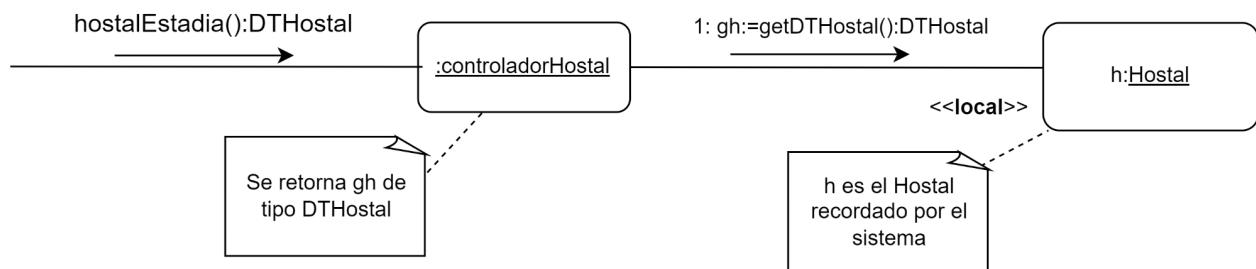


## 1.14 Consulta de Estadía

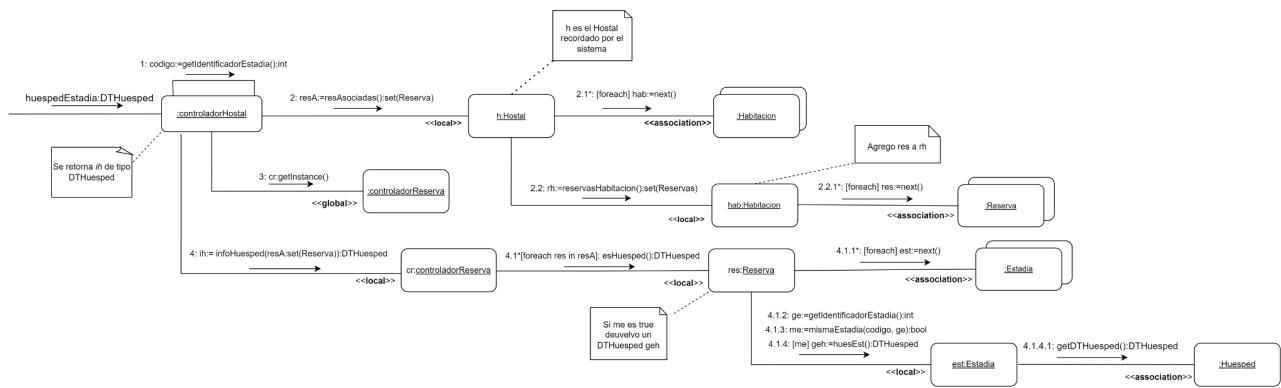
**Diagrama de comunicación de la operación obtenerEstadias(String) : set(DTEstadio)**



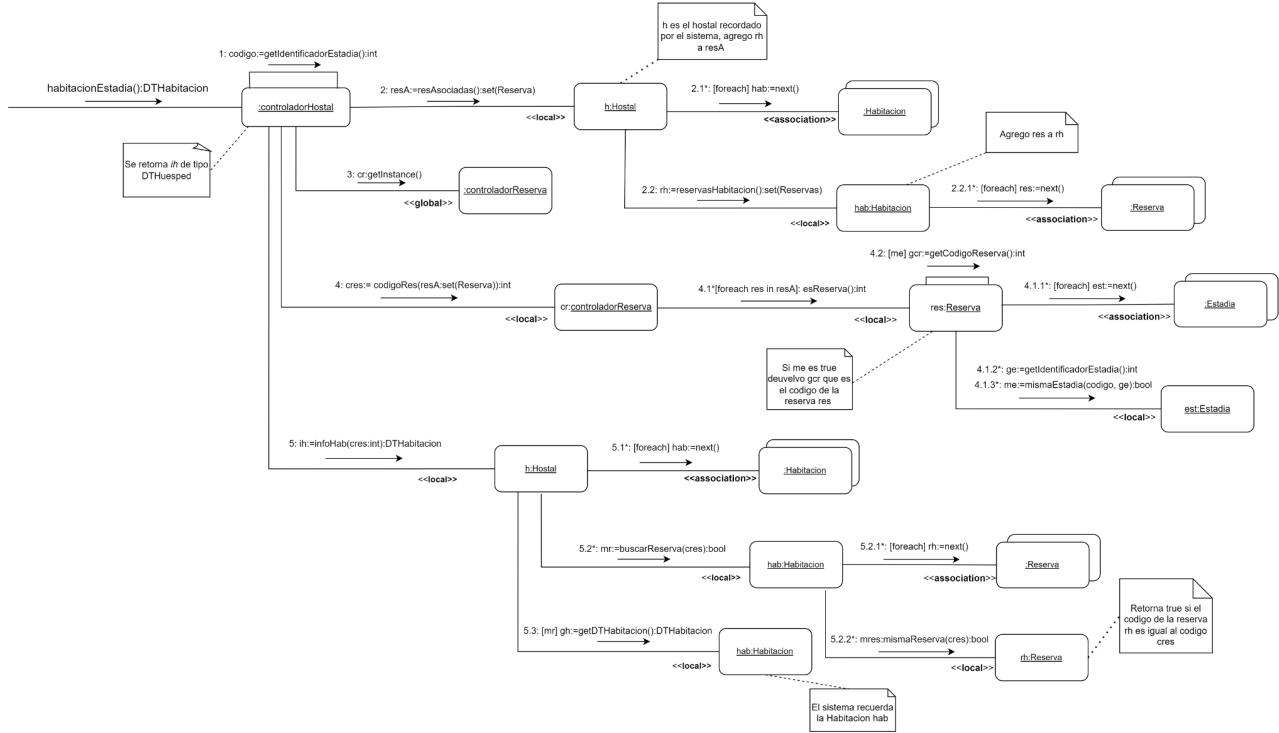
## Diagrama de comunicación de la operación hostalEstadia():DTHostal



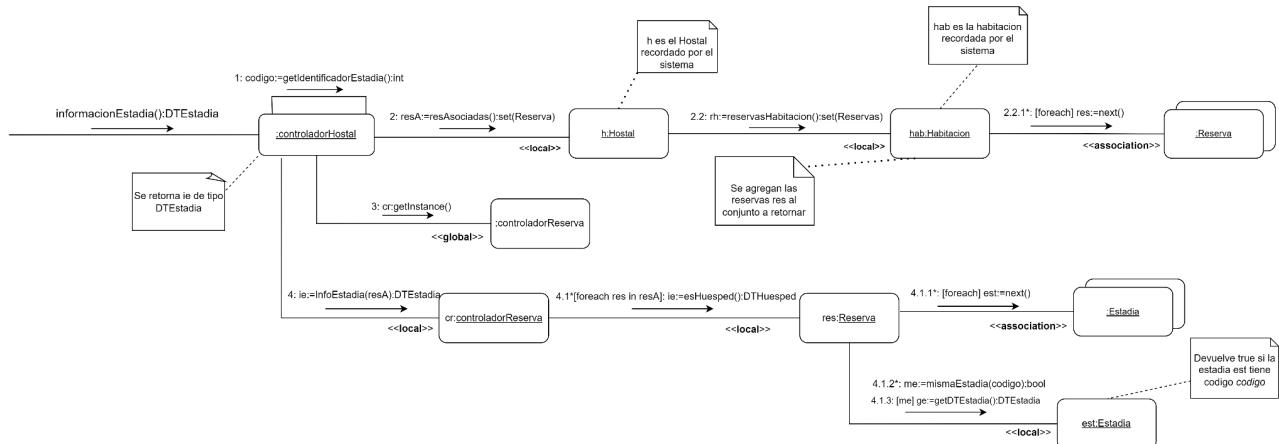
## Diagrama de comunicación de la operación huespedEstadia:DTHuesped



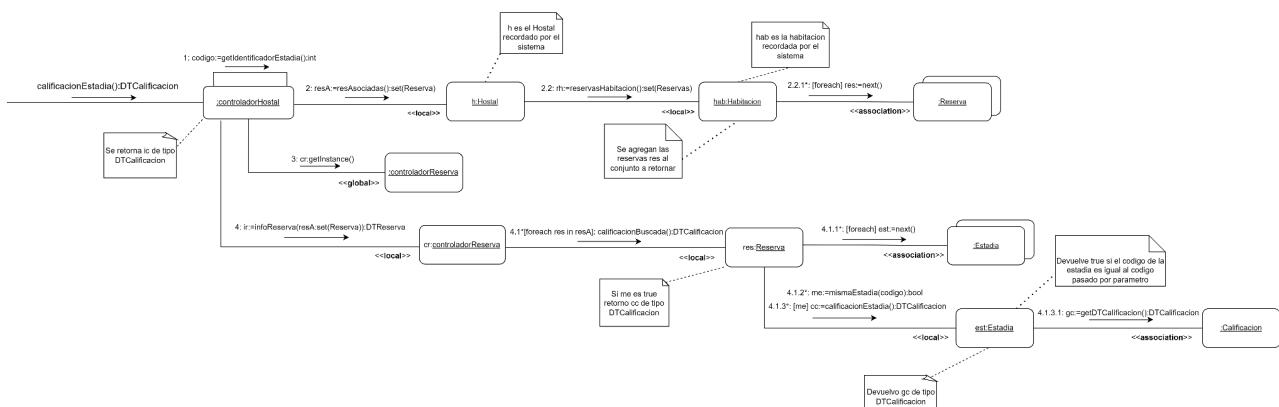
## Diagrama de comunicación de la operación habitacionEstadia():DTHabitacion



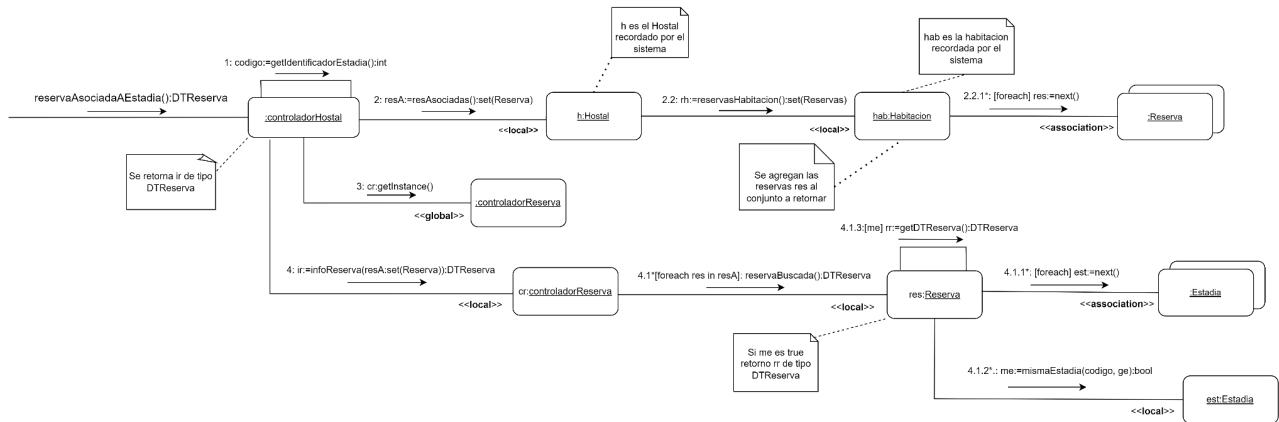
## Diagrama de comunicación de la operación informacionEstadia():DTEstadia



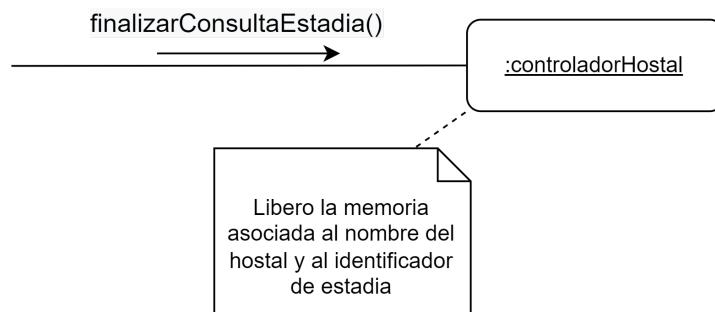
## Diagrama de comunicación de la operación calificacionEstadia():DTCalificacion



## Diagrama de comunicación de la operación reservaAsociadaAEstadia():DTReserva



## Diagrama de comunicación de la operación finalizarConsultaEstadia()



## 1.15 Baja de reserva

Diagrama de comunicación de la operación bajaReserva(integer)

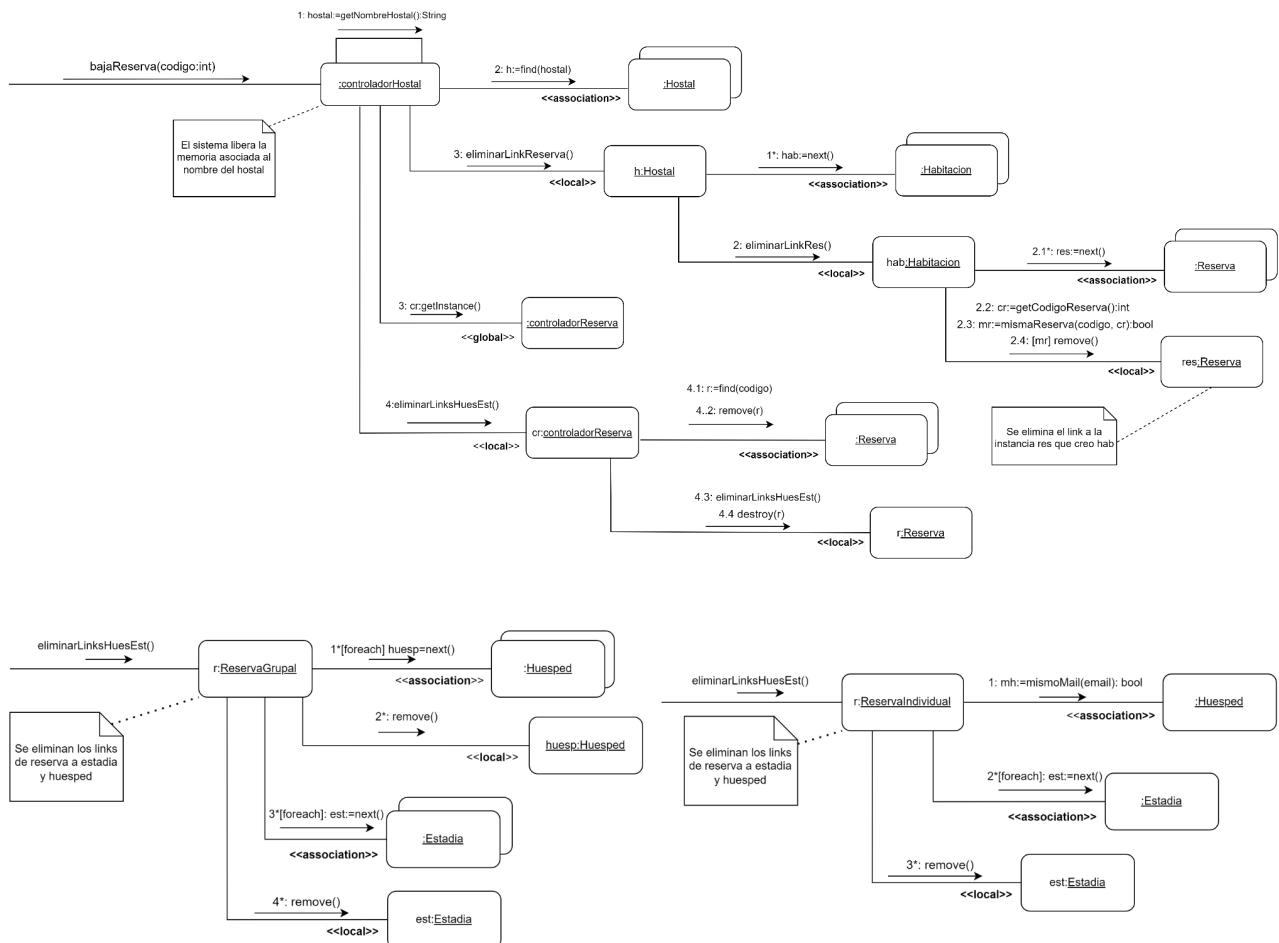


Diagrama de comunicación de la operación cancelarBajaReserva(integer)

cancelarBajaReserva(codigo:int)

:controladorHostal

El sistema libera la memoria asociada al nombre del hostal, los DTHostal y las DTReserva

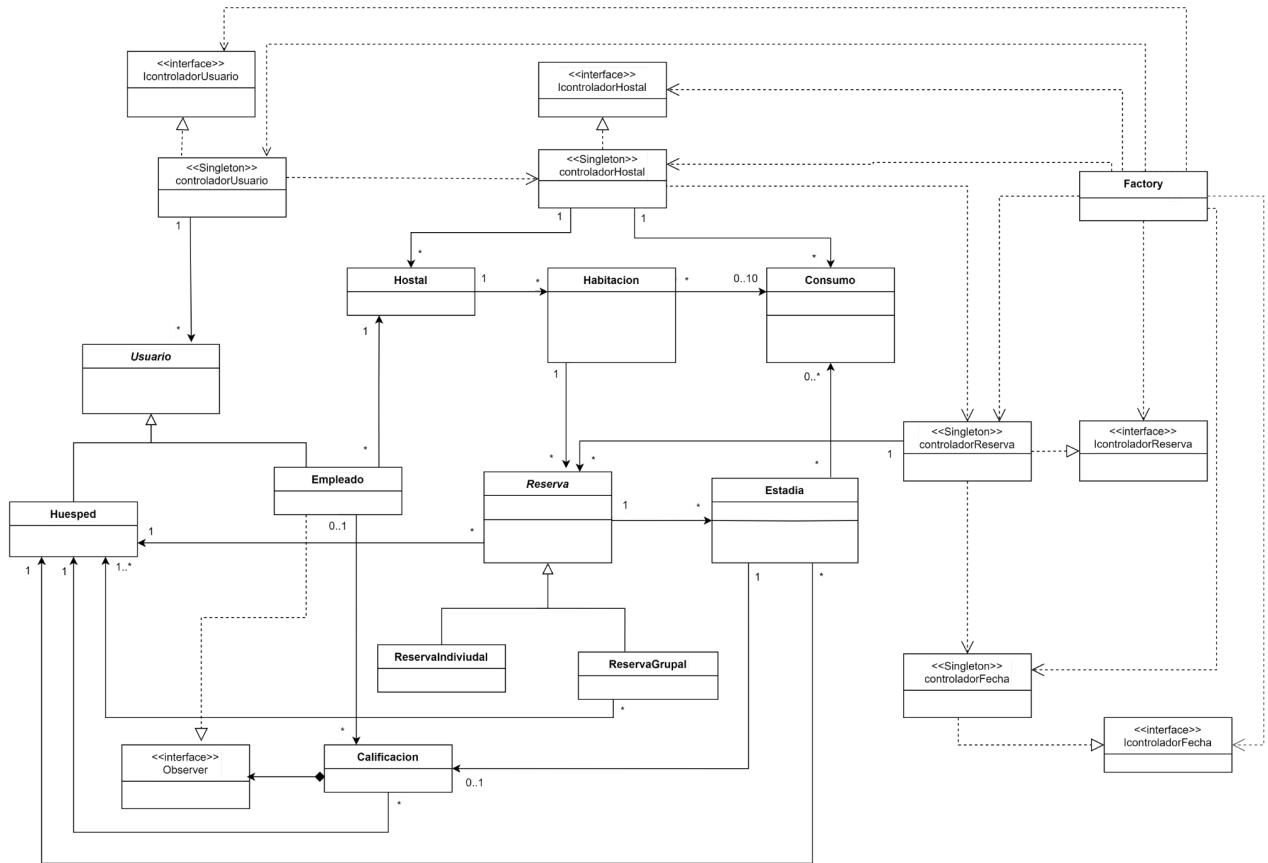
## **2 Criterios Generales**

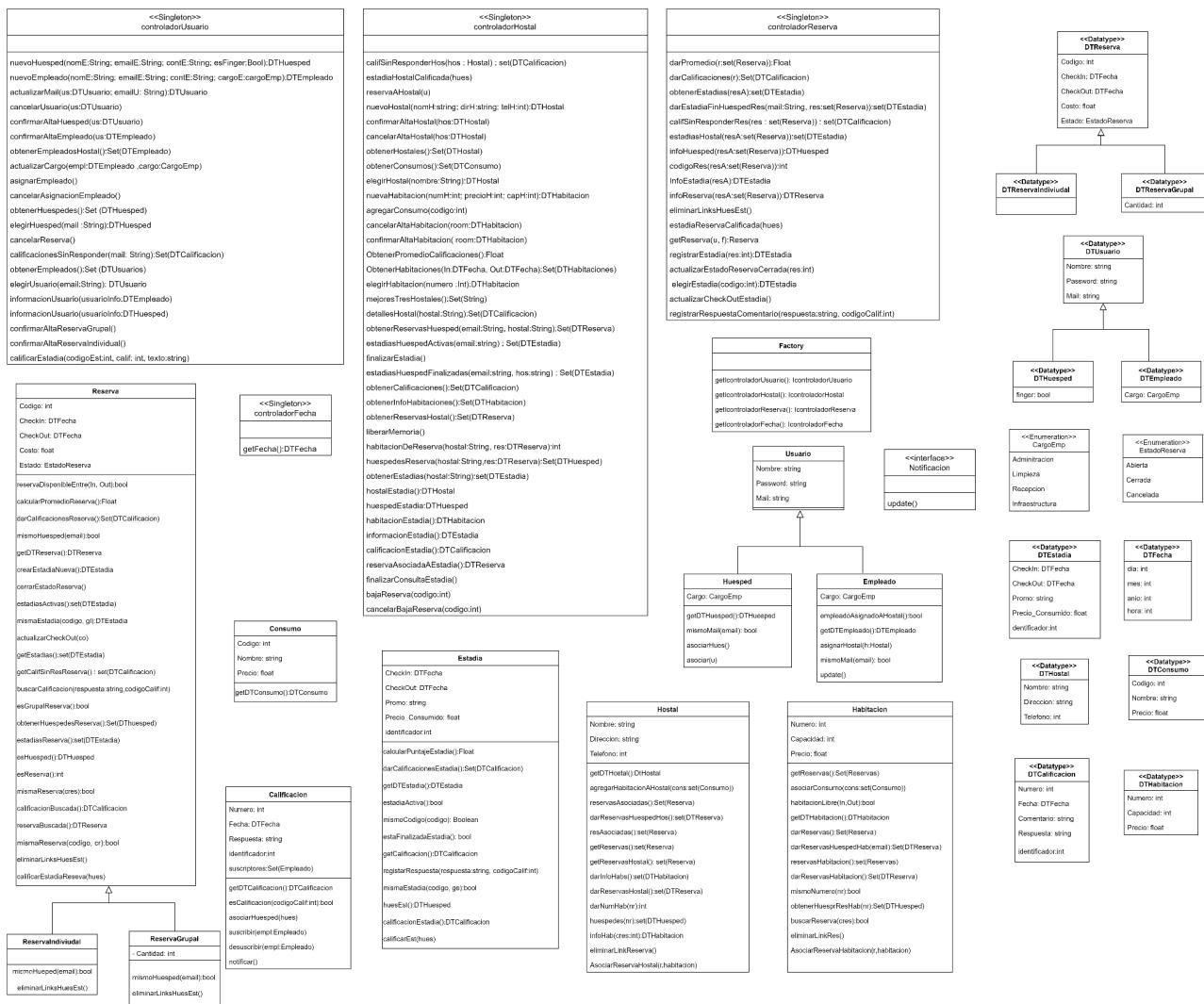
La elección de los controladores es detallada en el ejercicio 5. En esta sección solo se adelantara que como se puede ver en los diagramas de comunicación optamos por cuatro controladores, controladorUsuario, controladorReserva, controladorHostal y controladorFecha.

La responsabilidad de devolver el datatype de un objeto se la dimos a cada clase de dicho objeto, por ejemplo, dado una instancia de empleado esta puede devolver un DTEmpleado con los mismos atributos que dicha instancia.

Por último, se trató de asignar la responsabilidad de destruir una instancia al que tiene la responsabilidad de crearla. Por ejemplo, controladorReserva crea instancias de Reserva y además tiene la responsabilidad de destruir dichas reservas.

# Ejercicio 3





## Ejercicio 4

*Aplicar patrones de diseño para resolver los requerimientos solicitados en la etapa actual.  
Incluir el diseño propuesto en el diagrama de clases realizado en la parte anterior y justificar brevemente por qué los patrones aplicados resuelven el requerimiento.*

Para solucionar el nuevo requerimiento de las notificaciones, se utilizó el patrón *observer*. Este patrón define una relación one-to-many de dependencia entre objetos, de tal forma que cuando uno cambia su estado todos sus dependientes son notificados y actualizados automáticamente. Viéndolo de esta forma, resulta bastante claro que este patrón resuelve el requerimiento, ya que cada vez que una calificación es agregada se debe notificar a todos los empleados que deseen recibir dichas notificaciones. A priori, teniendo solo una clase concreta no es de gran utilidad utilizar este patrón. Sin embargo, igualmente optamos por esta opción, ya que, si eventualmente se agregan requerimientos en donde más clases quieren recibir las notificaciones, el patrón permite solucionarlo de forma sencilla (simplemente agregando nuevas clases concretas).

Para el requerimiento de la fecha se utilizó el patrón *singleton*. Este patrón asegura que una clase tendrá solo una instancia, proporcionando acceso global a esta instancia. Sabemos que la fecha del sistema es única, por lo que solo vamos a querer una instancia de dicha Fecha, por lo que el patrón singleton resulta ideal para solucionar este requerimiento.

# Ejercicio 5

*Explicar qué criterios GRASP y patrones de diseño se utilizaron. Para cada patrón, indicar las clases participantes y sus roles.*

Al momento de asignar las responsabilidades de manejar las operaciones del sistema, se utilizó el criterio *Controller*. Dicho criterio ayuda a asignar la responsabilidad de manejar una operación del sistema. Se utilizó una mezcla entre un controlador por caso de uso y un solo controlador *Façade*, agrupando los casos de uso por funcionalidades. Las operaciones relacionadas con los usuarios las maneja el controladorUsuario, las relacionadas a los hostales el controladorHostal y por último las relacionadas a las reservas el controladorReserva. También se utilizó un controladorFecha, el cual su única función es devolver la fecha actual del sistema, la cual puede ser modificada con el nuevo caso de uso. Este último controlador pudo haber sido unificado con el de controladorReserva, ya que solo depende de este. Sin embargo, optamos por no hacerlo, con el objetivo de mantener el alto acoplamiento dentro de controladorReserva. Todos los controladores son *singleton*, ya que solo se necesita una instancia de estas y queremos acceso global a estos.

Para crear nuevas instancias se tuvo en cuenta el criterio *Creator*, el cual nos dice cual es la clase correspondiente para crear una instancia de otra clase. Por ejemplo, para crear una nueva instancia de Hostal Alta de Hostal, se utilizó el controladorHostal, dado que es el que lleva registro de las instancias de hostales creados.

El criterio *Expert* se utilizó reiteradas veces, el cual dice de asignar una responsabilidad al experto en información, es decir la clase que tiene o conoce la información necesaria para cumplir con la responsabilidad. Por ejemplo, para obtener puntaje de calificación, es necesario pedirlo a la clase calificación, ya que es dicha clase la que tiene la información correspondiente.

También se intentaron de seguir al máximo los criterios de bajo acoplamiento (disminuir la cantidad de objetos con la que interactúa un objeto), alta cohesión (evitar que un mismo objeto haga demasiado y muy distinto trabajo) y no hables con extraños (evitar que un objeto gane temporalmente visibilidad sobre un objeto con el que no está conectado directamente).

Luego, respecto a los patrones de diseño utilizados, se utilizaron los mencionados anteriormente. Las clases participantes en observer son:

- Calificación: cumple el rol del *subject* notificando al *observer* cada vez que se agrega una nueva calificación.
- Notificación: es la interfaz *observer*, la cual se encarga de notificar a los *ConcreteObserver* cada vez que *subject* se lo pida.
- Empleado: es el *ConcreteObserver*, es decir, los que deben ser notificados a través del notificacion que se agregó una nueva calificación.

Además, como se mencionó anteriormente también se utilizó el criterio *Singleton*, no solo para solucionar el nuevo requerimiento de la fecha sino también para el controladorUsuario, controladorHostal y controladorReserva, con el objetivo de que tengan una sola instancia con acceso global.

Por último, se utilizó el patrón *factory*, con el objetivo de no exponer la capa de creación lógica al cliente, comunicándose con el nuevo objeto a través de una interfaz. En nuestro caso en particular, Factory tomó el rol de la *fábrica*, IcontroladorUsuario, IcontroladorReserva, IcontroladorHostal e IcontroladorFecha tomaron los roles de *Interfaz Proveedor* mientras que cada respectivo controlador tomó el rol de *Proveedor concreto*.