**Patrón Builder**

Usamos este patrón para la creación de habitaciones, ya que las distintas habitaciones que puede tener el hotel no todas tienen las mismas comodidades, es decir algunas pueden contar con despertador, tv, internet o minibar y otras no o solo contar con algunas de estas opciones, por esta razón decidimos que el patrón “builder” nos resolvería el problema de tener un constructor extenso que muchas veces le pasaríamos valores en “false”. A su vez cuando una persona quiere buscar una habitación debe poner en los filtros si quiere o no estas comodidades, por eso también utilizamos la misma interfaz que usamos para crear habitaciones para crear una clase que contenga los filtros de la búsqueda agregando al “builder” de “filtro” las características especiales del mismo como son “setCheckIn” y “setCheckout”

**Patrón Singleton**

El patrón “Singleton” lo utilizamos para aquellas clases que no queremos que tengan más de una instancia ya que las usamos como “herramientas” dentro de nuestro sistema. Por ejemplo, tenemos una clase llamada “extras” que se encarga de almacenar los precios de los distintos extras que puede tener la habitación, esta clase también es usada para consultar los mismo a la hora de hacer el cálculo de cuánto cuesta una habitación, como contiene datos sensibles para nuestro sistema debemos controlar sus instancias.

**Patrón State**

Este patrón lo usamos para controlar el estado de la reserva, ya que dependiendo cuando esta fue realizada varía el precio final de la misma.

**Patrón Observer**

Este patrón lo usamos, ya que tenemos varias clases que desean ser notificados cuando cambia el estado de una reserva, entonces cuando una reserva sufre un cambio de estado, esta revisa su lista de “Observers” y utiliza el servicio de mensajería de nuestro sistema para enviarles la notificación del cambio variando el tipo de contacto que solicitaron.