Práctica Swift KeepCoding 2023

Profesor: Daniel Illescas Romero

Gestión de Reservas de Hotel Luchadores

Descripción: Desarrolla un sistema para gestionar reservas de hotel para Goku y sus amigos.

Requisitos:

1. Modela las estructuras: `Client` (cliente), `Reservation` (reserva) y `ReservationError` (errores de la reserva).
    1. `Client`: nombre, edad, altura (cm).
    2. `Reservation`: ID único, nombre del hotel, lista de clientes, duración (días), precio, opción de desayuno (true/false).
    3. `ReservationError.` Enumerado que implemente `Error` con tres errores (cases) posibles: se encontró reserva con el mismo ID, se encontró reserva para un cliente y no se encontró reserva.
2. Gestiona las reservas con la clase `HotelReservationManager`.
    1. Crea un listado para almacenar reservas.
    2. Crea método para añadir una reserva. Añade una reserva dado estos parámetros: lista de clientes, duración, opción de desayuno. Asigna un ID único (puedes usar un contador por ejemplo), calcula el precio y agrega el nombre del hotel.
        1. Verifica que la reserva sea única por ID y cliente antes de agregarla al listado. En caso de ser incorrecta, lanza o devuelve el error `ReservationError` correspondiente. Es decir, no puede haber otra reserva con el mismo identificador ni puede haber una reserva en la que coincida algún cliente con una reserva previa.
        2. El cálculo del precio es así: `número de clientes * precio base por cliente (20 euros quizá) * días en el hotel * 1,25 si toman desayuno o 1 si no toman desayuno.`
           `Ejemplo: 3 (clientes) * 20 euros (precio base) * 2 (días en hotel) * 1,25 (porque toman desayuno) = 3*20*2*1.25 = 150`
        3. Añade la reserva al listado de reservas.
        4. Devuelve la reserva.
    3. Crea un método para cancelar una reserva. Cancela la reserva dado su ID, lanzando un `ReservationError` si esta no existe. Para cancelar una reserva simplemente elimínala del listado de reservas dado su identificador.
    4. Crea un método (o propiedad de solo lectura) para obtener un listado de todas las reservas actuales.

Para probar tú mismo tu propio código crea las siguientes funciones y ejecútalas en el Playground:

1. `testAddReservation`: verifica errores al añadir reservas duplicadas (por ID o si otro cliente ya está en alguna otra reserva) y que nuevas reservas sean añadidas correctamente.
2. `testCancelReservation`: verifica que las reservas se cancelen correctamente (borrándose del listado) y que cancelar una reserva no existente resulte en un error.
3. `testReservationPrice`: asegura que el sistema calcula los precios de forma consistente. Por ejemplo: si hago dos reservas con los mismos parámetros excepto el nombre de los clientes, me deberían dar el mismo precio.

Consejo para los tests: usa la función `assert` para hacer comprobaciones de si algo es verdadero, ejemplo: `assert(hotelReservationManager.reservations.count == 1),` si el número de reservas es 1 entonces continúa, sino, el programa falla alertándote del error. Puedes usar además `assertionFailure("no debe ocurrir")` para lanzar un error cuando algo no deba ocurrir.