Guía 3: Agregando una nueva funcionalidad con TDD

Introducción

El cliente de nuestro proyecto nos pidió que agreguemos una nueva funcionalidad a la aplicación. Ahora, las **reservas** de un restaurante van a ser más que tan solo un horario. El cliente quiere que tengan más atributos asociados.

En esta etapa, no vamos a relacionar un restaurante con una reserva, solo vamos a encargarnos de modelarlas y de definir sus funcionalidades principales.

Requerimientos de la aplicación

1) Modelar el objeto Reserva

Este objeto debe tener como atributos:

- **Horario:** objeto de tipo `Date` que va a representa la fecha y la hora de la reserva.
- **Cantidad de personas:** un número entero.
- **Precio por persona:** un número entero.
- **Código de descuento**: un string.

2) Desarrollar la funcionalidad que calcule el precio base de una reserva

El precio base de una reserva es igual a la cantidad de personas por el precio por persona.

3) Desarrollar la funcionalidad que calcule el precio total de una reserva

La reserva debe ser capaz de responder el precio final:

precio final = **precio base** + **adicionales** - **descuentos**

Los adicionales y los descuentos son los siguientes:

Descuentos:

1. **Descuento por grupos grandes:** si la cantidad de personas de la reserva está entre 4 y 6 personas, se agrega un **5%** de descuento. Para grupos entre 6 y 8 personas un **10%** de descuento y para grupos mayores a 8 personas un **15%** de descuento.
2. **Descuento por código:** algunas reservas pueden tener un código de descuento asociado. Si no tienen ninguno, no se les otorga ningún descuento. Los códigos son:
    - **DES15:** obtiene un 15% de descuento.
    - **DES200:** obtiene $200 de descuento.
    - **DES1:** obtiene de descuento el valor equivalente al precio de una persona.

Adicionales:

1. **Adicional por horario:** las franjas horarias de 13 a 14 y de 20 a 21 horas son muy concurridas. Se agrega un adicional del **5%** si la reserva fue hecha para un horario dentro de esos rangos.
2. **Adicional por fin de semana:** si la reserva fue realizada para alguno de los días del fin de semana (viernes, sábado o domingo) se le agrega un adicional del **10%**.

Paso 1 - Red: Convertí los requerimientos en pruebas unitaria

Lee nuevamente con atención los **requerimientos de la aplicación.** Una vez que los tengas claros, escribí los tests que validen su funcionamiento.

Tendrás que validar:

- Que un restaurante calcule correctamente su **precio base**.
- Que un restaurante calcule correctamente su **precio final**, contemplando bien los descuentos y los adicionales.

Te damos algunos ejemplos de restaurantes con su respectivo precio base y precio final para que te ahorres los cálculos. Igualmente, podés crear los objetos que quieras para realizar las pruebas.

```
var reserva1 = new Reserva (new Date(2018, 7, 24, 11, 00), 8, 350, "DES1")

var reserva2 = new Reserva (new Date(2018, 7, 27, 14, 100), 2, 150, "DES200")
```

Reserva 1:

- Precio base: 2800
- Precio final: 2310

Reserva 2:

- Precio base: 300
- Precio final: 100

Prestá atención a la creación del objeto `Date`. Revisá primero, que representan los parámetros que recibe. Por ejemplo, para representar los meses, se utilizan valores del 0 al 11, dónde 0 es enero y 11 es diciembre.Te dejamos documentación para que leas sobre este objeto.

Una vez que termines de crear todos los tests, ejecutalos abriendo el archivo `test.html` y fijate que todos fallen. En el siguiente paso, vas a escribir el código necesario para que comiencen a funcionar.

Paso 2 - Green: Escribí el código mínimo e indispensable para que las pruebas pasen

En este paso vas a comenzar a programar para que las pruebas empiezen a funcionar. Recordá que no tenes que focalizarte en crear buen código, tenés desarrollar la menor cantidad de código para que las pruebas pasen. Tendrás que:

- Crear el archivo `reserva.js` y linkearlo al HTML. Recordá que tiene que estar antes del archivo `tests.js`.
- Crear el **constructor** del objeto reserva con los parámetros que corresponden: horario, cantidad de personas, precio por persona y código de descuento.
- Crear la función que devuelva el **precio base** de la reserva.
- Crear la función que calcule el **precio final** de la reserva. Esta función seguramente te quede muy larga, ya que tiene que calcular todos los **descuentos y adicionales**. No te preocupes por eso, en el siguiente paso vas a encargarte de mejorar tu código.

A medida que vayas desarrollando las funcionalidades, podes ejecutar el archivo `test.html` para probar que los test comiencen a dar el resultado esperado.

Paso 3 - Refactor: Mejorá las funcionalidades de la reserva

En este paso vas a refactorizar y mejorar el código que escribiste en el paso anterior.

Te focalizarás en la función que calcula el **precio final** de la reserva. Seguramente esa función sea larga y se encargue de hacer muchas cosas. Tiene que calcular el precio base, calcular los descuentos, calcular los adicionales y sumar y restar los valores obtenidos para obtener el precio final. No está bueno que eso pase. Cómo ya sabemos, una buena práctica de programación es que las responsabilidades estén divididas y las funciones modularizadas, cada una encargándose de lo que le corresponde.

Para refactorizar el código tendrás que **modularizar la función que calcula el precio final**, dividiendo en otras funciones el cálculo de cada adicional y cada descuento. Tenés que lograr que esa función solo se encargue de realizar el cálculo: precio base - descuentos + adicionales.

A medida que vayas haciendo modificaciones en tu código, corré los tests para verificar que todo siga funcionando.