RELACIÓN DE EJERCICIOS DE DART (IV)
EJERCICIOS CON CLASES

Para la resolución de estos ejercicios, seguiremos la documentación oficial y también repasar los conceptos de Orientación a Objetos, y particularizarlos para Dart, se recomienda este artículo

1- Dado el siguiente programa Dart:

```
void main() {
  final persona = {
    'nombre': 'Andrea',
    'edad': 36,
    'altura': 1.84
  };
  print("Mi nombre es ${persona['nombre']}. Tengo ${person['edad']} años,
tengo ${persona['altura']} metros de altura.");
}
```

Refactoriza el código para crear una clase `Persona` que contenga como atributos `nombre`, `edad`, `altura`. Declara el constructor para generar instancias, un método `personaDescripcion()` para que muestre una instancia justo como el print de ese código. Reescribe el método `toString()`.

Una vez hecho esto, crea dos instancias de `Persona` y utilizalas para llamar al método `personaDescripcion()` y `toString()`.

2- Dada la siguiente clase:

```
class Restaurant {
  const Restaurant({
    required this.name,
    required this.cuisine,
    required this.ratings,
  });
  final String name;
  final String cuisine;
  final List<double> ratings;
}
```

Añade nuevas funcionalidades para que se pueda:

- obtener el número de ratings
- añadir otro rating
- añadir otros ratings (más de uno)
- calcular el rating medio

3- Vamos a crear una clase BanderaFranjas con los siguientes atributos:

- horizontal/vertical según sea la orientación de las franjas
- la lista de franjas de colores que la define
- nombre de país u organización (podría ser nulo)

Vamos a crear para ella:

- un método constructor
- un método que muestre la bandera por pantalla con el nombre de los colores en franjas
- un método que compare dos banderas y diga si son idénticas. Investiga el uso de **mapEquals()** ¿qué necesitamos instalar?
- un método que compare dos banderas y diga si tiene las mismas franjas en diferente orientación
- un método que invierta el orden de los colores
- un método que invierta la orientación de las franjas

4- Define una clase Punto. Un punto tiene tres coordenadas espaciales (x, y, z) que lo definen. Además debe tener:

- el método constructor
- un método que permita la traslación en el valor de x (horizontal)
- otro que permita la traslación en el valor de y (vertical)
- otro que permita la traslación en el valor de z (cota)
- un método toString que lo muestre de la siguiente forma: (x, y, z)

5- Define una clase Vector utilizando para ello la clase Punto:

- un vector queda definido por dos puntos: inicio y fin
- debe poder trasladarse en vertical, horizontal y en la cota (trasladando cada uno de sus puntos)
- debe poder calcularse el producto de un escalar por un vector
- debe poderse sumar con otro vector
- debe poderse calcular el producto vectorial (solo si tienes formación matemática)

6- Repitamos el ejercicio anterior pero partiendo de una perspectiva distinta: se ha introducido en Gemini el siguiente prompt:

“definir clase vector en el plano dart con dos puntos origen y destino constructor y métodos para producto escalar, suma de vectores, módulo”

y se ha obtenido el siguiente código, que si bien es correcto, no casa con lo que se pedía en el enunciado. Refactorízalo para que haga uso de la clase Punto ya definida:

```
class Vector {

 double x1, y1, z1; // Coordenadas del origen
 double x2, y2, z2; // Coordenadas del destino
```

```
 // Constructor
 Vector(this.x1, this.y1, this.z1, this.x2, this.y2, this.z2);

 // Método para calcular el producto escalar
 double productoEscalar(Vector otroVector) {

   return (x2 - x1) * (otroVector.x2 - otroVector.x1) +
          (y2 - y1) * (otroVector.y2 - otroVector.y1) +
          (z2 - z1) * (otroVector.z2 - otroVector.z1);
  }

 // Método para multiplicar un vector por un escalar
 Vector multiplicarPorEscalar(double escalar) {
    return Vector(x1, y1, z1, x1 + (x2 - x1) * escalar,
                  y1 + (y2 - y1) * escalar, z1 + (z2 - z1) * escalar);
    }

 // Método para sumar dos vectores
 Vector suma(Vector otroVector) {
    return Vector(x1, y1, z1, x2 + (otroVector.x2 - otroVector.x1), y2
            + (otroVector.y2 - otroVector.y1), z2 + (otroVector.z2 -
            otroVector.z1));
   }

 // Método para calcular el módulo del vector
double modulo() {
   return ((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1) + (z2 - z1) *
(z2 - z1));
 }
}
```

7- Dado el siguiente programa que define las clases `Figura`, `Círculos` y `Cuadrado`:

```
import 'dart:math';
abstract class Figura {
  double get area;
}

class Cuadrado extends Figura {
  Cuadrado(this.lado);
  final double lado;

  @override
  double get area => lado * lado;
}

class Circulo extends Figura {
  Circulo(this.radio);
  final double radio;
```

```
  @override
  double get area => pi * radio * radio;
}
```

Añade una nueva clase hija llamada `Triangulo` que tenga como atributos `lado1` , `lado2` y `lado3`.

Añade un nuevo método abstracto get `perimetro` a la clase `Figura` e impleméntala en todas las subclases.

Notas

- el perímetro de un cuadrado es igual a `4 * lado`
- el perímetro de un círculo es igual a `2 * pi * radio`
- el perímetro de un triángulo es igual a `lado1 + lad2 + lado3`

Después, añade un nuevo método `printValores()` a la clase `Figura`. Cuando sea llamado, deberá imprimir tanto `area` como `perimetro`.

Finalmente, crea una lista de Figuras que contenga tanto `Cuadrados` como `Circulos` y `Triangulos` y llame a `printValues()` con cada uno de sus componentes.

8- Define una clase CuentaBancaria con los siguientes atributos:

- número de cuenta
- nombre del titular
- saldo (inicial, 0 euros)
- número de operaciones (inicial, 0)

Define también las siguientes operaciones:

- constructor
- toString
- depositar dinero
- extraer dinero
- transferir dinero

9- En el ejercicio anterior, no estamos teniendo en cuenta si está permitido que una cuenta tenga el saldo negativo (cuenta de crédito) o no (cuenta de débito). En realidad, todo es prácticamente igual, salvo que en la de crédito, hay un límite máximo para el saldo negativo, y en la de débito, no se puede extraer dinero si el saldo no es suficiente.

Reestructura el ejercicio y define la clase CuentaBancaria como abstracta e instancia dos clases hijas CuentaDebito y CuentaCredito que tengan todo esto en cuenta en su operatoria.

Incluye un main que defina un objeto de cada tipo y pruebe que se ejecuta correctamente.

10- Volviendo al supuesto anterior, ambas cuentas son muy parecidas, habiendo entre ambas una única diferencia. Intenta factorizar el código para que no sea necesaria realizar la jerarquía de clases. Discute los pros y contras de que el código quede así (claridad, concisión, mantenibilidad, etc…)

11- Define la clase fracción en Dart con los atributos numerador y denominador. Define también los siguientes métodos:

- constructor
- sumar
- restar
- multiplicar
- dividir
- simplificar
- máximo común divisor
- toString()

12- Revisita la clase Fracción y crea una clase derivada FraccionComparable que incluya los métodos booleanos que calculen:

- esMayorQue
- esMenosQue
- esIgualA
- esMayorOIgualQue
- esMenorOIgualQue
- esDistintoQue

13- Crea un mixin en Dart llamado Comparable<T> . A partir de un tipo T, deberá crear una interfaz que incluya los métodos anteriores. ¿Podría usarse en el caso anterior?

14- Incluye testing a todos los métodos de los ejercicios 4 y 5.

15- Incluye testing al ejercicio 9.

16- Haz un proyecto para aplicar testing a la clase fracción.