EXAMEN TAP

curso 2017/18 convocatoria ordinaria (25/1/18)
FCT/ZTF – UPV/EHU

Los tres ejercicios se resolverán dentro de un mismo proyecto, cada uno en un paquete denominado: edu.upv.fct.examenTap1718.<su DNI>.ejN, donde N será el número del ejercicio.

1- ESCRITURA DE UNA CLASE (valoración 1/3)

Este ejercicio supone aplicar conocimientos específicos, tomar algunas decisiones y respetar unas cuantas "obligaciones" y "recomendaciones" que ha de conocer. Se valorará puntualmente y por separado cada uno de estos aspectos. **Hacer que la clase sea "robusta" (gestione bien posibles errores aritmeticos, infinitos, NaNs, cuestiones de precisión, etc.) es algo que puede llevar horas. No se trata de eso, basta con plantear bien toda la estructura de la clase y el código genérico de cada método conforme a las operaciones aritméticas mostradas.**

Defina la clase "PoligonoRegular" de modo **que tenga** :

- **un constructor:**

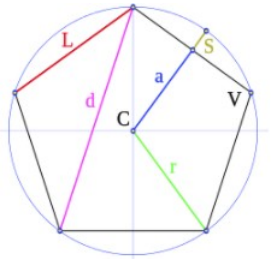
  PoligonoRegular(int numeroDeLados, double logitudDelLado)

- **y cuatro generadores de instancias concretas**:

  newTriangulo(double logitudDelLado) newCuadrado(double logitudDelLado)
  newPentagono(double logitudDelLado) newHexagono(double logitudDelLado)

Escriba getters y/o setters teniendo en cuenta que los polígonos regulares deben ser no alterables (que sean inmutables).
Escriba **métodos** para obtener ...

(1) el radio (r) = $L/(2*\sin(\pi/N))$ (2) la apotema (a) = $L/(2*\tan(\pi/N))$ (3) el área (A) = $(P * a)/2$
(4) el perímetro (P) = $N * L$ (5) la ságita (S) = $r-a$



donde N es el número de lados y L es la longitud del lado

Reescriba los métodos de Object que considere oportunos. El "`toString()`" es obligatorio.

Implemente los interfaces que considere adecuados a la clase. En particular implemente el interfaz **Comparable** tomando el área como criterio de relación de orden entre dos polígonos regulares (es mayor el que mayor área tiene).

2- ESCRITURA DE ALGORITMOS (valoración 1/3)

La **factorización de enteros** o **factorización de primos** consiste en descomponer un número natural en sus divisores primos, que cuando son multiplicados dan el número original.

1800	2
900	2
450	2
225	3
75	3
25	5
5	5
1	

La imagen de la izquierda muestra la factorización del número 1800, de donde se deduce que $1800 = 2^3 * 3^2 * 5^2$. Nótese que para obtener la descomposición se debe partir con el 2 a la derecha y dividir el número de la izquierda; si resulta no ser divisible, se incrementa el valor de la derecha y se reintenta. De esta manera, los factores que se obtienen serán siempre primos (no es preciso comprobarlo: en el ejemplo anterior, viendo que 25 no es divisible por 3, se comprueba si es divisible por 4, y obviamente no lo será porque ya se habrá probado con el 2 y no lo era, de modo que se pasa a comprobar si es divisible por 5.)

Escriba el cuerpo de las dos funciones siguientes:

- **public static int[] getFactors(int n);**
  Recibe un número entero positivo y devuelve un array con todos sus factores primos, ordenados de menor a mayor y con el número de apariciones correspondiente. Si el valor de entrada fuera 1800, debería devolver: [2, 2, 2, 3, 3, 5, 5]
- **public static SortedMap<Integer,Integer> getFactorsMap(int n);**
  Recibe un número entero positivo y devuelve un mapa ordenado cuyas claves son los factores primos y los valores el número de repeticiones. Si el valor de entrada fuera 1800, debería devolver: {2-->3, 3-->2, 5-->2}

Inclúyalas en una clase **Factorization** y añada un "main" de prueba que resuelva el siguiente problema:

> Encontrar el número más grande menor o igual a 100_000 que tenga la mayor cantidad de factores primos diferentes.

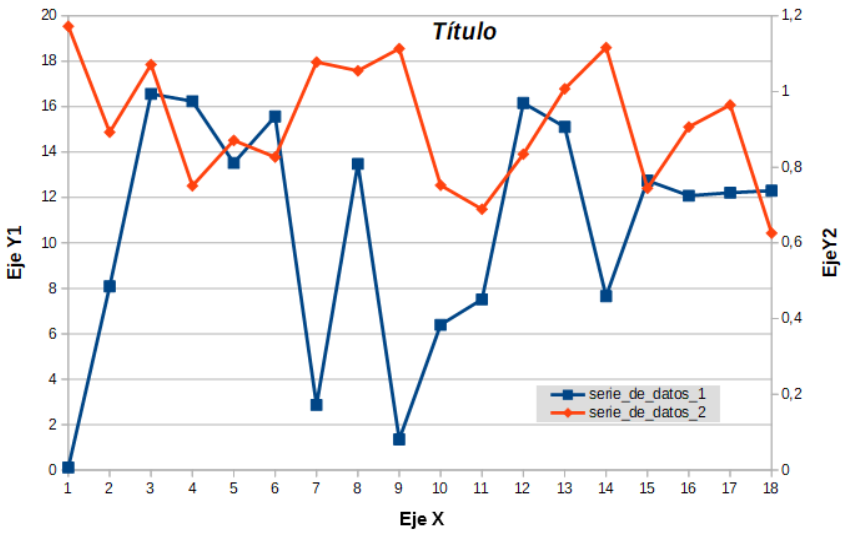
Por ejemplo, para el límite de 10 dicho número es el 10: 2 factores (2^1*5^1) para 100 es el 90: 3 factores ($2^1*3^2*5^1$), para 1000 es 990: 4 factores ($2^1*3^2*5^1*11^1$), o para 10_000 es el 9870: 5 factores ($2^1*3^1*5^1*7^1*47^1$)...

3- PLANTEMIENTO DE UNA SOLUCIÓN ESTRUCTURADA (valoración 1/3)

Escriba toda la estructura que considere necesaria (clases/interfaces/enumeraciones/, campos, métodos, etc. excepto el cuerpo de los métodos (o sí, pero no se exige)

para disponer de la posibilidad de generar gráficas como las de la figura (series de datos representadas con lineas y marcadores). No se trata de hacer nada gráfico, sino de disponer de la información necesaria para que otra clase se pueda encargar de la representación en un panel ("renderizar") que podrá incluirse en un GUI.

Por tanto se trata de plantear **public class Grafica {}** con todos los campos y métodos que se consideren necesarios, así como otras clases/interfaces/... que resulten a su vez necesarias para dar soporte a ello.



Piense por ejemplo que:

- una **gráfica** tiene un título
- una **gráfica** tiene varias **series** de datos (ilimitradas)
- una **gráfica** tiene dos o tres **ejes** (X, Y y puede que un Y secundario)
- una **gráfica** tiene una leyenda con información de cada serie de datos
- ...
- un **eje** tiene un máximo y un mínimo
- un **eje** tiene un título
- ...
- una **serie** de datos lleva asociado un tipo de **linea** y unas **marcas** geométricas en sus valores
- una **serie** de datos está ligada bien al **eje** Y o bien al **eje** Y secundario.
- cada **linea** y cada **marca** asociados a una **serie** pueden tener estilo propio (tipo/grosor/color/...) (tipo/tamaño/color/...).
- ... (de este modo puede establecer la posibilidad de disponer de gráficas más o menos elaboradas en función de cuantas características añada... piense en todo momento que de lo que se trata es de que otra clase pueda acceder a la información de un modo razonable para hacer el "renderizado")

(nota: para conseguir compilar sin errores, los métodos con valor de retorno necesitarán un "return", que puede escribirse con una constante. p.ej.: return null, o return 0.0
nota2: valore hasta qué punto desarrolla la solución para demostrar que domina el problema, pero tenga en cuenta que dispone de un tiempo limitado.)