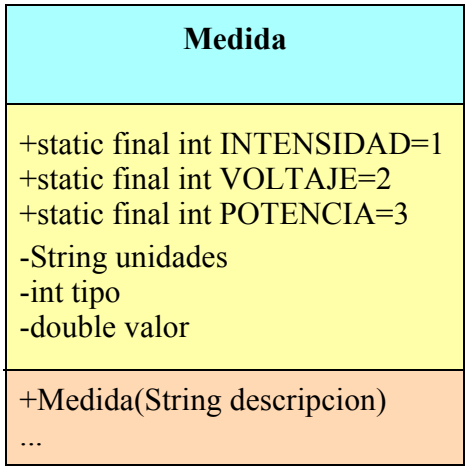
Examen de Programación
(Grados en Física y Matemáticas)

Septiembre 2015

Primera parte (1.25 puntos por cuestión, 50% nota del examen)

1) Se desea escribir el constructor de la clase Medida que representa una medida tomada en un circuito eléctrico y cuyo diagrama se muestra.



El constructor recibe como parámetro un texto que contiene tres elementos separados por uno o varios espacios en blanco. Son: el tipo de medida (INTENSIDAD, VOLTAJE o POTENCIA), el valor numérico de la medida (que es un número real) y las unidades (un texto). El tipo de medida se define con el texto "Intensidad", "Voltaje" o "Potencia", respectivamente para cada uno de los tres tipos. Ejemplos:

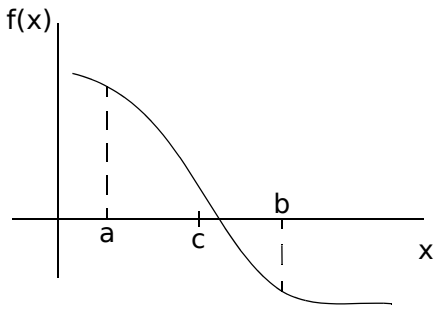
```
Intensidad 3.0 mA
Voltaje 4.5 V
Potencia 3.74 W
```

El constructor busca en el string el texto que describe el tipo de medida y pone en el atributo tipo el número correspondiente. Además, guarda el valor de la medida en el atributo valor y las unidades en el atributo unidades.

Para los tres ejemplos que se muestran, el resultado del constructor será, respectivamente:

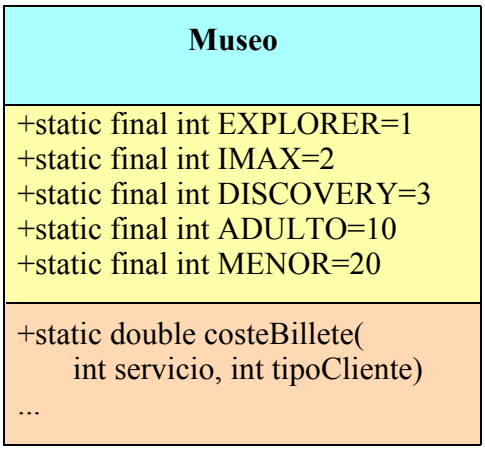
```
tipo=1  valor=3.0  unidades="mA"
tipo=2 valor=4.5  unidades="V"
tipo=3 valor=3.74  unidades="W"
```

2) Se desea encontrar una raíz de la funcion $f(x)$, es decir, un valor x que haga $f(x)=0$ o cercano a cero. La función $f(x)$ se halla ya escrita como un método cuyo parámetro es x y que retorna $f(x)$.



Escribir el pseudocódigo del algoritmo de la *bisección* para encontrar y retornar esta raíz. Este algoritmo parte de dos valores de x (a y b) que se pasan como parámetros, para los que el signo de $f(x)$ es diferente (como en la figura). El algoritmo comprueba el signo de $f(x)$ en el punto intermedio $c=(a+b)/2$. Si el signo en c es igual al signo en a (como en la figura), reemplaza a por c. En caso contrario reemplaza b por c. Este proceso se repite hasta que $f(c)$ sea cercano a cero más menos un margen *epsilon*$=10^{-8}$, o hasta que $b-a$ sea menor en valor absoluto a una *tolerancia*$=10^{-5}$. En ese momento, el punto intermedio c es la raíz buscada.

3) Se desea escribir el método costeBillete para la clase Museo cuyo diagrama se muestra en la figura. El método calcula y retorna el coste, en libras, de la entrada a determinados servicios del Museo de Ciencias de Londres. Los servicios disponibles son EXPLORER, IMAX o DISCOVERY). El coste depende del servicio y del tipo de cliente (ADULTO o MENOR). El cálculo se hace según los valores de esta tabla, que vienen en libras:



Tipo Cliente	EXPLORER	IMAX	DISCOVERY
ADULTO	25.00	11.00	6.00
MENOR	22.00	10.00	5.00

Se valorará la eficiencia de la implementación.

4) En un computador con sistema operativo Linux se dispone de un directorio llamado luminiscencia que contiene datos de unos experimentos de medida de luminiscencia. Este directorio está situado en el directorio del usuario y contiene 2 directorios llamados presion_ambiente y alta_presion. Dentro de cada uno de estos directorios hay dos directorios llamados fluorescencia y fosforescencia. A su vez, dentro de estos últimos hay ficheros de texto (acabados en .txt), gráficas (acabadas en .png) y ficheros de hoja de cálculo (acabados en .xlsx). No hay nada más.

Hacer un dibujo con un esquema de la estructura de directorios de partida.

Se desea escribir un *script* que separe los diferentes tipos de ficheros y los filtre, haciendo lo siguiente (usar las órdenes que sean necesarias en cada paso):

- moverse al directorio del usuario para comenzar a operar desde allí
- crear en el directorio del usuario dos nuevos directorios llamados textos y gráficas
- copiar en textos todos los ficheros acabados en .txt que estén en las carpetas llamadas fluorescencia o fosforescencia
- mover a gráficas todos los ficheros acabados en .png que estén en las carpetas llamadas fluorescencia o fosforescencia
- borrar del directorio fluorescencia que está por debajo de alta_presion todos los ficheros que comiencen por tmp
- borrar los dos directorios llamados fosforescencia con todos sus contenidos

Hacer también un dibujo con un esquema de la estructura de directorios resultante al final.

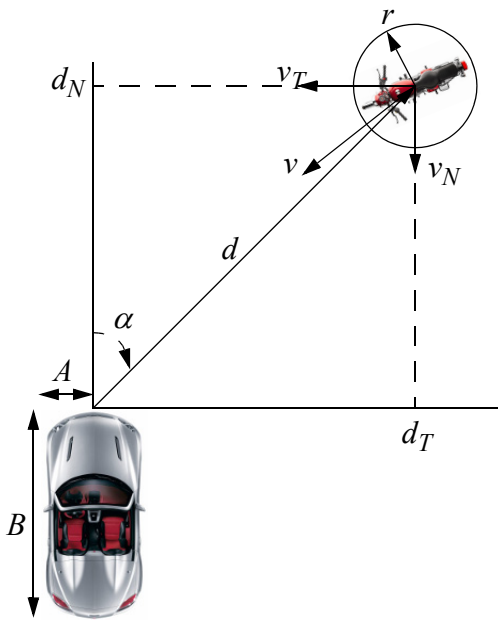
Examen de Programación
(Grados en Física y Matemáticas)

Septiembre 2015

Segunda parte (5 puntos, 50% nota del examen)

El pocos años está previsto que muchos automóviles tengan capacidades de conducción automática. Se desea implementar parte del software de un sistema para evitar colisiones. El automóvil dispone de un radar que ve los obstáculos que tiene delante y mediante un software de análisis de datos determina la velocidad y posición de cada obstáculo, así como su radio.

Obstáculo
-int id -double vT -double vN -double d -double alfa -double r
+Obstáculo(int id, double r) +set(double vT, double vN, double d, double alfa) +double tAlcance(double vC) +double tRebase(double vC, double B) +double margenAlcance(double vC) +double margenRebase(double vC, double B) +int id() +double r()



La clase Obstáculo está ya implementada y dispone de los siguientes atributos:

- id: entero que identifica al obstáculo
- vT: velocidad tangencial (v_T), en m/s
- vN: velocidad normal (v_N), en m/s
- d: distancia al obstáculo en m
- alfa: ángulo al obstáculo (α), en grados
- r: radio del obstáculo en m

Los métodos de esta clase son:

- *Constructor*: Recibe como parámetros los valores iniciales de los atributos id y r
- set(): Recibe como parámetros los valores actualizados de los atributos vT, vN, d y alfa

- tAlcance(): Calcula el tiempo hasta alcanzar el obstáculo en la dirección del coche, dada la velocidad del coche vC (v_C) en m/s

$$t_{alc} = \frac{d_N - r}{v_C - v_N}$$

- tRebase(): Calcula el tiempo hasta rebasar el obstáculo en la dirección del coche, dada la velocidad del coche vC (v_C) en m/s y la longitud del coche B en m

$$t_{reb} = \frac{d_N + r + B}{v_C - v_N}$$

- margenAlcance(): Calcula el margen de distancia tangencial entre el coche y el obstáculo cuando transcurra el tiempo de alcance

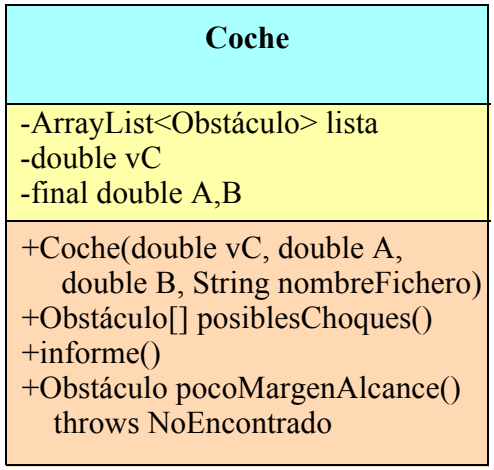
$$margen_{alc} = d_T + v_T t_{alc}$$

- margenRebase(): Calcula el margen de distancia tangencial entre el coche y el obstáculo cuando transcurra el tiempo de rebase

$$margen_{reb} = d_T + v_T t_{reb}$$

- id(): Observador del atributo id.
- r(): Observador del atributo r.

Lo que se pide es implementar en Java la clase Coche pensada para simular y probar el funcionamiento del sistema de detección de colisiones. Su diagrama de clases se muestra en la figura. La clase dispone de los siguientes atributos:



- lista: una lista de obstáculos
- vC: velocidad del coche (v_C) en m/s
- A, B: dimensiones del coche (semianchura y longitud) en m

Los métodos de la clase hacen lo siguiente:

- *constructor*: recibe como parámetros la velocidad del coche vC en m/s, las dimensiones A y B del coche en m y el nombre de un fichero de texto del que se leen datos para rellenar la lista de obstáculos. El fichero tiene una línea de encabezamiento que se ignorará, y un obstáculo por línea con todos sus atributos en el formato del ejemplo. Para cada línea se creará un objeto de la clase Obstáculo, se dará valor a sus atributos con el método set() y se añadirá el objeto a la lista. Ejemplo:

```
id      vT      vN      d       alfa    r
13456    4.56  -25.56   123.2     1.2   2.5
13457   -2.30  -10.4    220.0    30.3   4.5
13458   -5.43   12.56    83.4   -23.4   4.7
```

- posiblesChoques(): Retorna un array conteniendo todos los obstáculos para los que se detecta un posible choque. Para este método se usará el siguiente pseudocódigo:

```
método posiblesChoques() retorna Obstáculo[]
    // lista para meter los obstáculos que pueden chocar
    ArrayList<Obstáculo> chocan = nueva lista vacía
    // Recorrido para obtener los obstáculos que pueden chocar
    para cada obs de lista hacer
        booleano choca=false
        //solo puede haber choque si los tiempos de alcance están entre 0 y 30s
        si tiempo de alcance de obs entre [0,30] y
            tiempo de rebase de obs entre [0,30]
        entonces
            //El choque se produce si se da alguna de estas tres circunstancias:
            si valor absoluto del margen de alcance de obs <= r+A entonces
                choca=true
            si no, si valor absoluto del margen de rebase de obs <= r+A entonces
                choca=true
            si no, si (margen de alcance de obs) *(margen de rebase de obs) <0
                // el margen de alcance y el de rebase tienen distinto signo
                choca=true
            fin si
        fin si
        // si choca lo metemos en la lista
        si choca entonces
            añade obs a la lista chocan
        fin si
    fin para
    // Crear el array
    Obstáculo[] posibles= nuevo array de tamaño igual al de chocan
    // Metemos los datos de la lista chocan en el array
    para cada i desde 0 hasta tamaño de posibles-1 hacer
        posibles[i]=casilla i de chocan
    fin para
    retorna posibles
fin método
```

- informe(): Pone en pantalla un informe de todos los obstáculos. Para ello pone en pantalla una línea de encabezamiento y luego recorre la lista de obstáculos mostrando los datos de cada uno, uno por línea, con el formato del ejemplo:

```
id        tiempoAlcance    tiempoRebase   margenAlcance margenRebase
13456       8.26s            35.56s          3.2m           4.5m
13457      12.30s            21.42s        220.0m         224.5m
13458       5.43s             8.56s         83.4m          88.7m
```

  Se valorará poner los datos en columnas como aparecen en el ejemplo

- pocoMargenAlcance(): Busca en la lista el primer Obstáculo cuyo margen de alcance en valor absoluto es menor o igual que r+A y lo retorna. Si no lo encuentra lanza NoEncontrado.

La excepción NoEncontrado está ya creada en una clase aparte.

Finalmente, se pide hacer un programa principal de prueba de los métodos de la clase Coche, en una clase aparte, que haga lo siguiente:

a. Crea un objeto de la clase `Coche` con velocidad 18.5 m/s, dimensiones `A`=1.3, `B`=4.5 y nombre de fichero `"obstaculos.txt"`.

b. Muestra en pantalla el identificador del obstáculo obtenido con el método `pocoMargenAlcance()`. Si se lanza `NoEncontrado` se muestra un mensaje de error, se salta el paso c), pero se sigue por el paso d).

c. Muestra en pantalla los identificadores de todos los obstáculos obtenidos con `posiblesChoques()`.

d. Invoca el método `informe()` para obtener un informe de los obstáculos existentes.

Valoración:

- encabezamiento de la clase, atributos y constructor: 1 punto
- `posiblesChoques`: 1 punto
- `informe`: 1 punto
- `pocoMargenAlcance`: 1 punto
- programa principal: 1 punto