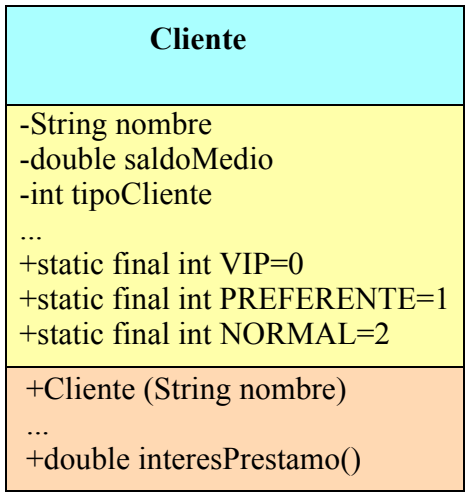
Examen de Programación
(Grados en Física y Matemáticas)

Junio 2013

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

1) Se dispone de la clase `Cliente` que guarda los datos de un cliente de un banco y obedece al diagrama de clases que se muestra.



Se desea escribir el método `interesPrestamo()` que calcule y retorne el interés, en %, que se cobrará al cliente cuando pida un préstamo. Para el cálculo se partirá de la base indicada en la tabla, según el atributo `tipoCliente`:

`tipoCliente`	base del interés
VIP	4%
PREFERENTE	6%
NORMAL	7%

La base se multiplicará por un factor de escala que será de 0.8 si el `saldoMedio` es superior a 100000 euros, de 0.9 si está entre 100000 y 20000, y de 1.0 en otros casos.

2) Escribir el código Java de un método recursivo que permite reemplazar un texto `viejo` por otro `nuevo` en un texto más grande llamado `completo`. El método retornará el texto obtenido. Su cabecera será:

```
public static String reemplaza(String completo, String viejo, String nuevo)
```

Llamaremos `n` a la longitud de `viejo`.

El caso directo se da cuando el texto `completo` tiene una longitud menor que `n`. En este caso se retorna el texto `completo`.

En el caso recursivo:

- Si los primeros `n` caracteres de `completo` son iguales a `viejo` se retorna el texto `nuevo` concatenado al valor retornado por `reemplaza` usando como primer parámetro los caracteres de `completo` excepto los `n` primeros, y con los mismos parámetros `viejo` y `nuevo`.
- Si no son iguales, se retorna el primer carácter de `completo` concatenado al valor retornado por `reemplaza` usando como primer parámetro el texto `completo` excepto su primer carácter, y con los mismos parámetros `viejo` y `nuevo`.

3) Escribir el pseudocódigo de un método iterativo que calcula y retorna el desarrollo en serie de la función:

$$f(x) = (1+x)\cdot e^{x} = \sum_{i=0}^{n} \frac{i+1}{i!} x^{i}$$

El método recibe como parámetros los valores x y n. Para hacer más eficiente el cálculo, observar que los valores del factorial $i!$ y de la potencia x^i se pueden obtener en la iteración número i para la iteración siguiente multiplicando el factorial anterior por i+1 y la potencia anterior por x.

4) Escribir un método que lee de un fichero de texto cuyo nombre se le pasa como parámetro todos los números reales que contenga y calcula y retorna su suma. El fichero solo contiene números reales separados por espacios, tabuladores o saltos de línea. La cabecera del método debe ser:

```
public static double sumaCoeficientes(String nombreFichero)
```

5) En un computador con sistema operativo Linux se dispone de un directorio llamado `proyectos` que está dentro del directorio del usuario. A su vez, dentro de `proyectos` hay un subdirectorio, llamado `p1`.

Se desea escribir un *script* que se ejecute desde el directorio del usuario y que haga lo siguiente:

- crear en el directorio `proyectos` un nuevo directorio llamado `definitivos` y otro llamado `borradores`
- crear dentro de `definitivos` un subdirectorio llamado `graficas` y otro llamado `src`
- copiar en `src` todos los ficheros acabados en `.java` que estén en `p1`
- mover a `graficas` todos los ficheros acabados en `.graph` que estén en `p1`
- borrar de `p1` todos los ficheros que acaben en `.class` y los que acaben en `.ctxt`
- mover todos los ficheros que queden en `p1` a `borradores`
- borrar el directorio `p1`

Examen de Programación
(Grados en Física y Matemáticas)

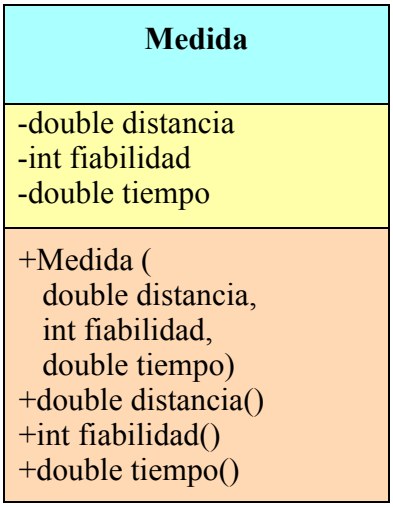
Junio 2013

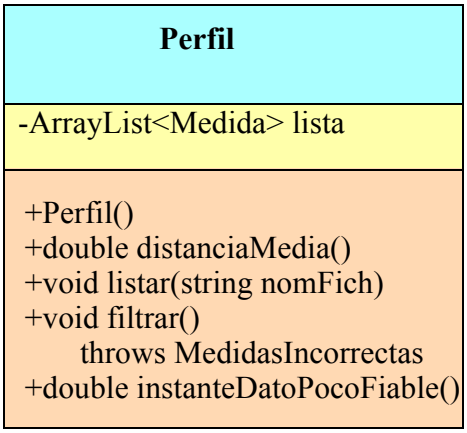
Segunda parte (5 puntos, 50% nota del examen)

Se desea escribir parte del software de análisis de los resultados de la medida de un perfil de distancias obtenido con un sensor láser. Cada medida se guardará en un objeto de la clase `Medida` cuyo diagrama se muestra abajo y que ya está implementada. Sus atributos son:

- `distancia`: es la distancia en metros desde el sensor a la pieza que se está midiendo.
- `fiabilidad`: un número de 1 a 5 que nos da el sensor láser e indica la fiabilidad de la medida, siendo el 1 el grado menos fiable y el 5 el más fiable.
- `tiempo`: el tiempo en segundos desde el inicio del experimento.

Como puede observarse, se dispone de un constructor al que se le pasan los datos de la medida, y de métodos observadores, uno para cada atributo:





Lo que se pide es implementar en Java la clase `Perfil` cuyo diagrama de clases se muestra arriba. La clase dispone de un atributo llamado `lista` en el que se guarda una lista de medidas obtenidas en instantes consecutivos a lo largo del tiempo. El primer elemento de la lista es el más antiguo, y el último el más nuevo. Los métodos de la clase hacen lo siguiente:

- *constructor*: es el método que realiza las medidas y las guarda en la `lista`. Este método se supone ya realizado, y por tanto no se pide.
- `distanciaMedia`(): calcula la distancia media del perfil, que se obtiene con la expresión

$$(t_{n-1} - t_0) \sum_{i=0}^{n-2} \frac{d_{i+1} + d_i}{2} (t_{i+1} - t_i)$$

  donde n es el número de medidas de la lista, d_i es la distancia de la medida i, y t_i es el tiempo de la medida i.
- `listar`(): escribe en el fichero de texto cuyo nombre se pasa como parámetro una línea de título seguida de todas las medidas de la lista con el formato que se muestra en este ejemplo (respetar el número de decimales y la alineación):

```
tiempo     distancia   fiabilidad
0.01       0.125       4
0.02       0.126       4
0.04       0.128       5
...
```

- `filtrar`(): Elimina de la lista las medidas cuya distancia se aleja de la media en más de un 20% y las que tienen una fiabilidad menor que 3. Si hay más de 10 medidas en este caso se lanza `MedidasIncorrectas`. El método seguirá el siguiente pseudocódigo:

```
método filtrar() lanza MedidasIncorrectas
   real media=distanciaMedia()
   // hallar cuántas medidas están demasiado alejadas
   entero malas=0
   para i desde 0 hasta longitud de lista -1 hacer
      si (|distancia de la medida i - media|>0.2*media) o
        (fiabilidad de la medida i<3)
      entonces
         malas++
      fin si
   fin para
   si malas>10 entonces
      lanza MedidasIncorrectas
   fin si
   // crear una nueva lista para poner las medidas buenas
   ArrayList<Medida> buenas= nueva lista vacía
   // Pasar todas las medidas buenas a la nueva lista
   para i desde 0 hasta longitud de lista -1 hacer
      si no ((|distancia de la medida i - media|>0.2*media) o
        (fiabilidad de la medida i<3))
      entonces
         añade casilla i de lista a buenas
      fin si
   fin para
   // sustituir la lista vieja por la nueva
   lista=buenas
fin método
```

- `instanteDatoPocoFiable`(): retorna el primer instante con una medida de fiabilidad 1, o Double.NaN si no hay ninguno.

Finalmente, se pide hacer un programa principal en una clase aparte que haga lo siguiente:

a. Crear un objeto de la clase `Perfil`

b. Invocar el método `filtrar()`

c. Listar el perfil invocando `listar`() con en el fichero "`perfil.txt`"

d. Mostrar en pantalla el valor retornado por `distanciaMedia`()

e. Mostrar en pantalla el resultado de invocar `instanteDatoPocoFiable`()

En este programa, si se lanza `MedidasIncorrectas` se deben abandonar los pasos *c* y *d* y mostrar un mensaje de error. El paso *e* debe ejecutarse tanto si se lanza `MedidasIncorrectas` como si no.

Valoración: métodos y programa principal: 1 punto cada uno