Examen de Introducción al Software
(Ingeniería Informática)

Febrero 2017

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

1) Escribir un método Java al que se le pasa como parámetro un array de Strings, y que calcula y retorna el número de elementos del array que comienzan por el texto "grado", "máster" o "doctorado". Se recuerda que la clase String dispone del siguiente método que permite saber si el string actual comienza por el texto str:

```
boolean startsWith(String str)
```

2) Escribir un método Java que aplicando el algoritmo iterativo de Ramanujan calcule y retorne una aproximación del número π.

$$\frac{1}{\pi} \cong \frac{2\sqrt{2}}{9801} \sum_{k=0}^{n} \frac{(4k)!(1103 + 26390k)}{(k!)^4 396^{4k}}$$

Para hacer este cálculo de forma eficiente, crear una variable para el factorial de (4*k*), otra para el factorial de *k* y otra para la potencia 396^{4k}. Tener en cuenta que tras calcular el valor actual del sumatorio para *k*, para obtener los próximos valores de estas variables bastará multiplicar la variable con el factorial de (4*k*) por (4*k*+1)*(4*k*+2)*(4*k*+3)*(4*k*+4); para el factorial de *k* basta multiplicar la variable por (*k*+1); y para la potencia multiplicar la variable por 396*396*396*396.

Este algoritmo es extremadamente eficiente; solo con *n*=1 se obtiene la precisión de un número *double* y si usásemos números con mayor precisión que el *double*, con cada nueva iteración se obtendrían otros 8 decimales más. Existe una evolución más moderna pero más compleja, el algoritmo de Chudnosvsky, que obtiene 16 decimales con cada iteración.

3) Escribir un método Java que reciba como parámetros los siguientes valores reales relativos a la altura *h* de una columna de líquido que se descarga a través de un capilar de radio *R* situado en su parte inferior. El método recibe como parámetros:
   - *R*: radio del capilar, en m
   - h_0: altura inicial en m
   - *t*: tiempo transcurrido en s
   - ρ: densidad del líquido en Kg/m^3
   - η: viscosidad, en Kg/(ms)
   - *L*: longitud del capilar, en m
   - *S*: sección de la columna, en m^2

   El método calcula y retorna la altura, *h*, en m obtenida mediante las siguientes expresiones, siendo *g* la gravedad:

$$\lambda = \frac{\pi}{8} \cdot \frac{R^4(\rho g)}{\eta L S} \qquad h = h_0 \cdot e^{-\lambda t}$$

4) Escribir el *pseudocódigo* de un método recursivo llamado `darVuelta` que permite dar la vuelta a las palabras de un texto que se pasa como parámetro. Las palabras se separan con un solo espacio en blanco y no hay espacios al principio ni al final. El método retorna un string con las palabras del string original pero puestas al revés. Por ejemplo, si el texto original es `"andrés come arroz"` el resultado será `"arroz come andrés"`.

   Para este método se cuenta con las siguientes operaciones aplicables a un texto: la operación `buscaEspacio` que busca el primer espacio en el texto y retorna su posición, o `-1` si no hay ninguno (similarmente al `indexOf(" ")` de Java). Y también la operación `substring(posIni,posFin)` que obtiene un fragmento del texto situado entre las posiciones `posIni` (incluido) y `posFin` (excluido), al estilo Java.

   Para hacer el método tener en cuenta que el caso directo se da cuando el texto original no tiene ningún espacio y en ese caso hay que retornar el texto original. En el caso recursivo hay que retornar `darVuelta`(texto original sin su primera palabra) + un espacio en blanco + la primera palabra.

5) Contestar *razonadamente* a las siguientes preguntas. Utilizar un *máximo* de *5 líneas* para cada respuesta:

   a. En una hoja de cálculo tenemos esta fórmula escrita en la celda `E1`: `=$B2*C1`. ¿Qué fórmula aparecerá en la celda `F2` si copiamos allí esta fórmula? ¿Y si la fórmula original fuese `=B2*$C$1`?

   b. Disponemos de una hoja de cálculo con valores entre las filas 3 a 53 en las columnas `B`, `C` y `D`. Indicar los pasos que darías para que en la columna `E` aparezca la suma de las casillas no vacías de la fila correspondiente.

   c. Se parte de una hoja de cálculo con las columnas `A`, `B`, `C` y `D` con datos ya metidos. Describe brevemente los pasos a realizar para añadir una columna nueva que contenga el valor máximo de las casillas no vacías de la fila.

   d. Explica con brevedad cómo se establece una relación entre dos tablas de una base de datos relacional.

   e. En una base de datos relacional un informe se puede basar en una tabla o en una consulta. Explica las principales diferencias entre hacerlo con una o con otra.

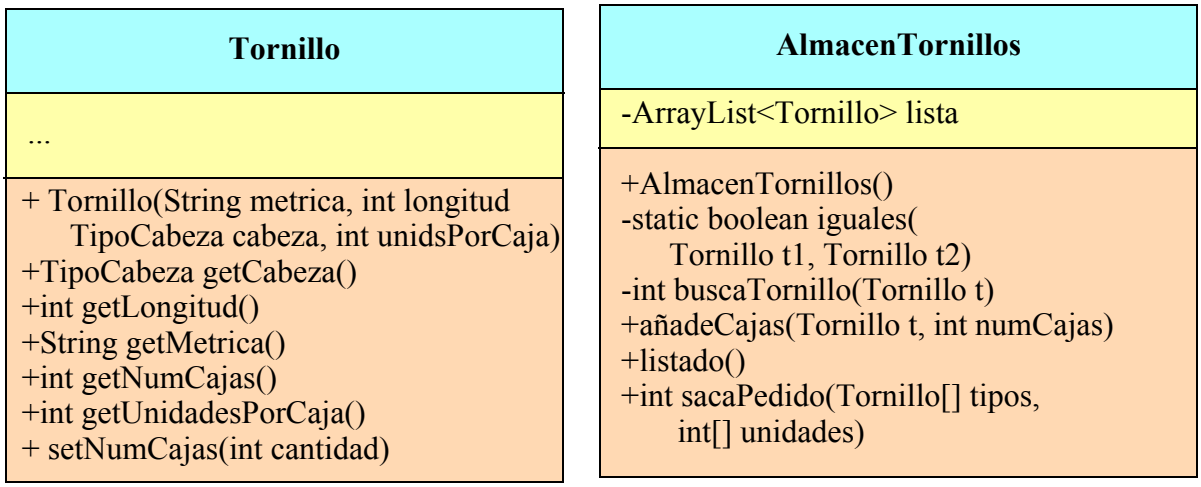
   Nota: en esta cuestión, las respuestas correctas suman 0.2 puntos, las incompletas o las que pasen de 5 líneas ni suman ni restan y las erróneas restan 0.1 puntos. Se valora la *precisión* de la respuesta.

Examen de Introducción al Software
(Ingeniería Informática)

Febrero 2017

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

Se desea realizar una parte del software perteneciente a un almacén de tornillería. Para ello se dispone de la clase Tornillo ya realizada, cuyos objetos almacenan datos relativos al tornillo como su métrica, longitud, número de unidades por caja y cajas disponibles. Dispone de métodos para construir el tornillo, observadores de los atributos y un método para cambiar el número de cajas disponibles. Se desea crear la clase AlmacenTornillos. Los diagramas de estas clases se muestran aquí:



Los métodos de la clase Tornillo hacen lo siguiente:

- *Constructor*: Construye el tornillo a partir de la métrica, longitud, tipo de cabeza y unidades por caja, que se pasan como parámetros. Los valores permitidos para la métrica son: M1, M2, M3, M4, M5, M6, M7, M8, M10, M12, M14, M16, M18, M20, M22, M24, M27, M30
- getCabeza(): Retorna el tipo de cabeza (ALLEN, HEXAGONAL o AVELLANADA).
- getLongitud(): Retorna la longitud en mm, sin contar la cabeza.
- getMetrica(): Retorna la métrica del tornillo. Si vale ERROR es porque ha habido un error al crear el tornillo.
- getNumCajas(): Retorna las cajas disponibles para este tornillo.
- getUnidadesPorCaja(): Retorna las unidades por caja.
- setNumCajas(): Modifica las cajas disponibles poniéndolas al valor indicado por el parámetro cantidad.

El tipo de cabeza se expresa como un dato de la clase enumerada TipoCabeza que está definida aparte de esta forma:

```
public enum TipoCabeza {ALLEN, HEXAGONAL, AVELLANADA}
```

La clase AlmacenTornillos dispondrá de un atributo que es un ArrayList de objetos de la clase Tornillo y que contiene la lista de los tipos de tornillos del almacén. Cada tipo de tornillo aparece

en esta lista una sola vez. Sus métodos deberán hacer lo siguiente:

- *Constructor*: Crea la lista vacía.
- iguales(): Retorna un booleano que indica si los tornillos t1 y t2 son del mismo tipo o no. Son del mismo tipo cuando coinciden la métrica, longitud y cabeza.
- buscaTornillo(): Busca un tipo de tornillo en la lista y retorna el índice de la casilla donde está, o -1 si no se encuentra. La búsqueda es solo usando la métrica, longitud y cabeza.
- añadeCajas(): Añade al almacén cajas de tornillos de un tipo. Si ya hay tornillos del mismo tipo añade numCajas a las que ya había. En otro caso establece las cajas del tornillo t iguales a numCajas y añade ese tornillo a la lista de tornillos.
- listado(): Hace un listado en pantalla de todos los tipos de tornillos del almacén. Se muestra primero una cabecera explicativa. Luego un tipo de tornillo por línea mostrando su métrica, longitud, tipo de cabeza, unidades por caja y unidades totales.
- sacaPedido(): Retira del almacén un pedido compuesto por diferentes tipos de tornillos, indicados en el array tipos. De cada tipo de tornillo se retirará el número de unidades indicado en la casilla correlativa (con igual índice) del array unidades. Si los arrays tipos y unidades no tienen el mismo tamaño, se retorna -1. Después de esta comprobación pero antes de realizar ninguna otra operación, se comprobará si el almacén contiene suficiente número de unidades de todos los tornillos del pedido. Si no hay suficiente de alguno de ellos, se retornará 0, sin hacer ninguna modificación en el almacén. En caso de que se pueda completar el pedido, se actualizará el número de cajas disponibles en el almacén para cada tipo de tornillo, restando el número de cajas sacadas de cada uno y el método retornará el número total de cajas sacadas del almacén (si un tipo de tornillo se queda con cero cajas seguirá figurando en la lista). Para este método se seguirá el siguiente pseudocódigo:

```
método sacaPedido(Tornillo[] tipos, entero[] unidades) retorna entero
   // Comprobamos el tamaño de los arrays
   si longitud de tipos != longitud de unidades entonces
      retorna -1
   fin si

   // Comprobamos la disponibilidad
   booleano disponible=true
   // recorremos todos los tornillos del pedido pero si encontramos uno que no
   // existe en el almacén o que no tiene suficientes tornillos disponibles
   // abandonamos el bucle
   entero i=0
   mientras (i<tipos.length && disponible)
      entero indc=buscaTornillo(tipos[i])
      si indc=-1 ó (número de cajas de la casilla indc de lista) *
          (unidadesPorCaja de la casilla indc de lista) < unidades[i]
      entonces
         disponible=false
      si no
         i++
      fin si
   fin mientras

   // Sacar las cajas en su caso
   si disponible entonces
      // hay disponibilidad y sacamos las cajas del almacén
      // recorremos todos los tornillos del pedido
```

```
        entero totalCajas=0 // contador de las cajas sacadas
        para j desde 0 hasta longitud de tipos-1
            entero indice=buscaTornillo(tipos[j])
            Tornillo tEnLista=casilla indice de lista
            entero porCaja=unidades por caja de tEnLista
            entero cajasDisponibles=número de cajas de tEnLista
            entero cajasSacadas=redondeo por arriba(unidades[j])/porCaja)
            establecer número de cajas de tEnLista a cajasDisponibles-cajasSacadas
            totalCajas=totalCajas + cajasSacadas
        fin para
        retorna totalCajas
    si no
        // no hay disponibilidad
        retorna 0;
    fin si
fin método
```

Escribir además en una clase aparte un programa de prueba con un main que haga:

- Crea el almacén vacío
- Añade unos cuantos tipos de tornillos (al menos tres)
- Hace un listado
- Saca un pedido para el que no hay suficiente stock, mostrando el valor retornado
- Saca un pedido para el que sí hay suficiente stock, mostrando el valor retornado
- Hace un listado

Valoración (sobre 10):

1) Encabezamiento de la clase, atributo y constructor: 0.5 puntos
2) iguales: 0.5 puntos
3) buscaTornillo y listado: 1.5 puntos cada uno
4) añadeCajas, sacaPedido: 2 puntos cada uno
5) main: 2 puntos