Examen de Programación I
(Ingeniería Informática)

Febrero 2010

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

1) Determinar el pseudocódigo de las inicializaciones y del cuerpo del bucle en este diseño iterativo para que se cumpla la especificación indicada. La secuencia sigue la interfaz de la secuencia vista en clase.

```
método desviacionEstándar(Secuencia<real> sec, real media) retorna real
  {Pre: sec tiene al menos dos elementos,
        media= media aritmética de los valores de sec}
    var
      real suma:=...; entero num:=..; real x;
    fvar
    sec.reinicia();
    {Inv: suma es la suma de los errores cuadráticos de pi(sec),
          num es la longitud de pi(sec)}
    {Cota: longitud de pd(sec)}
    mientras no sec.fds() hacer
          x:=sec.actual()-media
          ...
    fmientras
    retorna raíz cuadrada(suma/(num-1))
  {Post: si llamamos x_i a los sucesivos valores de sec,  1<=i<=n
     valor retornado=
```

desviación estándar de los reales de sec= $\sqrt{\frac{\sum_{i=1}^{n}(x_i - media)^2}{n-1}}$ }

```
fmétodo
```

Nota: los errores cuadráticos corresponden a los términos del tipo $(x_i - media)^2$

2) Utilizar el esquema de búsqueda en secuencias visto en clase para escribir la especificación y el diseño del método cuya cabecera se indica abajo y que sirve para saber si hay en la secuencia `sec` algún número cuya diferencia con `valorMedio` en valor absoluto es superior al 5% de `valorMedio`.

```
método hayDesviacionesGrandes (Secuencia<real>  sec, real valorMedio) retorna booleano
```

3) Escribir la especificación y diseño de un método que retorna el nivel de riesgo de un préstamo solicitado por un cliente a un banco. El nivel de riesgo es función de los siguientes datos que se pasan como parámetros al método:
   - *sueldo*: entero con el sueldo mensual del cliente, en euros
   - *tieneHistorial*: booleano que indica si el cliente tiene o no historial de pagos
   - *moroso*: booleano que indica si el cliente se ha retrasado en el pasado en algún pago, o no; este dato sólo es significativo si el cliente tiene historial de pagos
   - *hipoteca*: entero que indica cuánto paga el cliente al mes de hipoteca, en euros

El nivel de riesgo es un carácter `'A'`, `' M'` o `'B'`, que indica respectivamente riesgo alto, medio o bajo, y se calcula del siguiente modo:

- *Riesgo alto*: el cliente tiene historial de pagos y es moroso, o tiene una hipoteca que es más del 30% del sueldo
- *Riesgo medio*: el cliente no tiene historial de pagos y tiene una hipoteca que es menor que el 20% del sueldo, o tiene historial de pagos y no es moroso y tiene una hipoteca menor que el 30% del sueldo
- *Riesgo bajo*: el resto de los casos.

4) Indicar razonadamente si el diseño que se propone para este método cumple la especificación indicada, y en caso de que no la cumpla qué habría que hacer para solucionarlo.

```
método elementoMenor(real[1..3] a) retorna real
   {Pre:}
      si (a[1]<a[2] && a[2] <a[3]) entonces
         retorna a[1]
      fsi
      si (a[2]<a[1] && a[1] <a[3]) entonces
         retorna a[2]
      fsi
      si (a[3]<a[2] && a[2] <a[1]) entonces
         retorna a[3]
      fsi
   {Post: valor retornado = min{a[1],a[2],a[3]}}
fmétodo
```

5) Escribir un diseño recursivo para el siguiente algoritmo que permite encontrar de manera aproximada una raíz (valor de x para el que la función vale cero) de una función $f(x)$ real monótona creciente (es decir, que según x aumenta la función aumenta).

El caso directo es aquel en el que $|f(a)|<1.0E\text{-}10$, en cuyo caso se considera que se ha encontrado la raíz aproximada. En el caso directo se retorna a. En el caso recursivo se retorna el valor obtenido llamando a la misma función para el intervalo $[a,c]$ si $f(c)>0$ o para el intervalo $[c,b]$ si $f(c)<=0$, siendo c el valor medio del intervalo $[a,b]$.

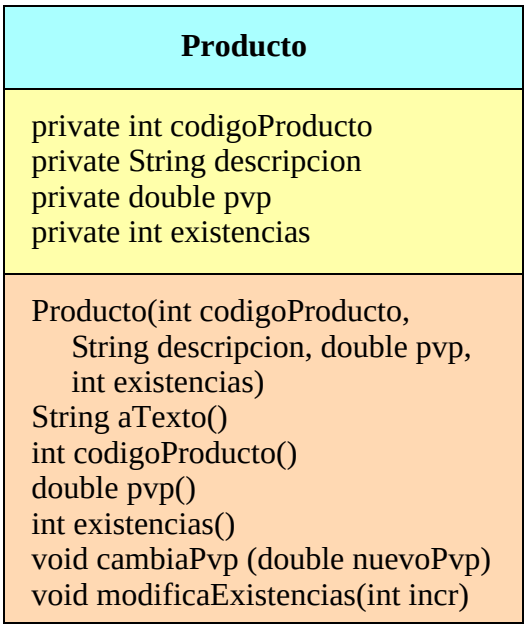
```
método raiz (real a, real b) retorna real
{Pre: f(a)<=0, f(b)>=0, a<b}
   calcula raiz
{Post: valor retornado=r, |f(r)|<1.0E-10}
```

Examen de Programación I
(Ingeniería Informática)

Febrero 2010

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

Se dispone de una clase ya realizada llamada `Producto` que permite almacenar los datos de un producto en un almacén. El diagrama de clases se muestra en la figura. El significado de los atributos es:



- `codigoProducto`: codigo numérico que identifica el producto
- `descripcion`: texto que describe el producto
- `pvp`: precio de venta al publico, en euros
- `existencias`: numero de unidades en el almacén

Y los métodos hacen lo siguiente:

- constructor: copia los parámetros en los atributos
- `aTexto`: retorna un texto con los datos del producto
- `codigoProducto`: retorna el código de producto
- `pvp`: retorna el `pvp`
- `existencias`: retorna las existencias
- `cambiaPvp`: cambia el `pvp` a la cantidad indicada
- `modificaExistencias`: modifica las existencias incrementándolas en la cantidad indicada, con su signo (si es negativa, decrementa las existencias)

Se pide implementar en Java la clase `Almacen` que sirve para guardar los datos de las existencias de un almacén. Para ello, el almacén tiene una lista variable de productos, almacenados en la parte inicial de un array de objetos de la clase `Producto` llamado `prod`. Los objetos se guardan en las primeras `num` casillas del array, siendo `num` un número entero.

La clase debe tener los datos `prod` y `num` descritos arriba como atributos (*nota*: no añadir más atributos). Además debe disponer de los siguientes métodos según lo descrito en los comentarios de documentación:

```
public class Almacen {
    /**
     * Constructor que crea el array con el tamaño máximo indicado
     * y deja el almacen vacío (con cero productos)
     */
    public Almacen(int max)    {...}
```

```
    /**
     * Inserta un nuevo producto en el almacen; retorna false si no cabe o si
     * ya existe un producto con el mismo código; en caso contrario inserta
     * el producto al final de la lista y retorna true;
     */
    public boolean insertaProducto(Producto p) {...}

    /**
     * Indica si hay algún producto con datos erróneos. Se considera error:
     * precio menor o igual a cero, o existencias negativas
     */
    public boolean hayErrores() {...}

    /**
     * Retorna el valor total de los productos almacenados:
     * suma de pvp*existencias para todos los productos
     */
    public double valorTotal() {...}

    /**
     * Incrementa el precio de todos los productos en el porcentaje indicado
     */
    public void incrementaPrecio(int porcentaje) {...}

}
```

Tener en cuenta los siguientes aspectos:

1) Constructor y atributos: (0.5 puntos)

2) `insertaProducto`: (1.5 puntos) usar el esquema de búsqueda en tablas para averiguar si ya existe un producto con el mismo código.

3) `hayErrores:` (1 punto) usar el esquema de búsqueda en tablas para averiguar si hay errores.

4) `valorTotal:` (1.5 puntos) usar el esquema de recorrido en tablas.

5) `incrementaPrecio`: (0.5 puntos) usar el esquema de recorrido en tablas.