Examen de Programación I
(Ingeniería Informática)

Febrero 2009

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

1) Escribir un diseño iterativo para el siguiente método al que se le pasa como parámetro una tabla que contiene las medidas obtenidas por el altímetro de un planeador, en metros, tomadas a cada segundo desde el inicio del vuelo. El método retorna el número total de segundos en que el planeador ha estado subiendo

```
método tiempoSubiendo(real[1..N] altura) retorna entero
  {Pre: N>1}
      Calcula el tiempo que un planeador ha estado subiendo
  {Post: valor retornado = cantidad de casillas de altura que cumplen
          altura[i]<altura[i+1] con i en el intervalo desde 1 hasta N-1}
fmétodo
```

Indicar para este diseño lo siguiente:

- el invariante
- las variables a crear y sus inicializaciones
- la condición de permanencia en el bucle
- el cuerpo del bucle
- la función de cota, razonando sobre si el bucle termina
- el estado a la terminación, con las instrucciones a realizar después del bucle

Poner además el pseudocódigo completo

2) Escribir la especificación y diseño de un método que determina si una subida de precio es baja, media o alta, en función de la tabla que se muestra a la derecha. El incremento se calcula de esta manera:

$$\frac{(precioNuevo - precioAntiguo)}{precioAntiguo} \cdot 100$$

incremento	caso	valor retornado
del 0% al 3%	bajo	B
>3% hasta 5%	medio	M
>5%	alto	A

El método recibe como parámetros (números reales) el precio antiguo y el nuevo y retorna el carácter que se indica en la tabla según el caso encontrado.

3) Utilizar el esquema de búsqueda en tablas visto en clase para escribir el diseño del siguiente método, que retorna el índice de la casilla de la tabla a que tiene el primer número que es divisible de manera exacta entre d:

```
método buscaDivisible (entero[1..N] a, entero d) retorna entero
```

4) Indicar razonadamente si el diseño que se propone para este método cumple la especificación indicada

```
método estanOrdenados(real a, real b, real c) retorna booleano
   {Pre:}
      si (a<b) & (b<c) entonces
         retorna verdad
      fsi
      si (b>=c) & (a>=b) entonces
         retorna verdad
      fsi
   {Post: valor retornado = (a<b y b<c) o (a>=b y b>=c)}
fmétodo
```

5) Escribir una especificación y un diseño recursivo para un método que muestra en pantalla las letras de una secuencia de caracteres, en orden inverso, una letra por línea. El caso directo es aquel en que hemos llegado al final de la secuencia; en este caso no se hace nada. El caso recursivo ocurre en la condición contraria (si no hemos llegado aún al final de la secuencia) y lo que hay que hacer en él es:

- anotar en una variable local el carácter actual de la secuencia
- avanzar la secuencia
- llamar recursivamente al mismo método
- mostrar en pantalla el carácter que fue anotado en la variable local

Suponer que la secuencia cumple la interfaz de las secuencias vista en clase, y que antes de la primera llamada al método ha sido reiniciada. La cabecera del método será:

```
método muestraAlReves (Secuencia<caracter> texto)
```

Examen de Programación I
(Ingeniería Informática)

Febrero 2009

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

Se dispone de una clase llamada `Avistamiento` que permite guardar los datos de un avistamiento de una especie de aves, anotando el nombre de la especie, el número de ejemplares, y el mes en que se realizó el avistamiento. La clase contiene los siguientes métodos:

```
public class Avistamiento {

    /** Constructor que pone los datos del avistamiento a
     * los valores indicados
     */
    public Avistamiento(String especie, int numEjemplares, int mes) {...}

    /** Escribe en una ventana los datos del avistamiento
     */
    public void muestra() {...}

    /** Retorna la especie
     */
    public String especie() {...}

    /** Retorna el número de Ejemplares
     */
    public int numEjemplares() {...}

    /** Retorna el mes en que se realizó el avistamiento
     */
    public int mes() {}

}
```

Asimismo se dispone de una clase `SecuenciaAvistamientos` que contiene una secuencia de objetos de la clase `Avistamiento` y que responde a la interfaz de la secuencia vista en clase:

```
/**
 * Clase que representa una secuencia de avistamientos
 */
public class SecuenciaAvistamientos {

    /** Constructor que crea la secuencia vacía
     */
    public SecuenciaAvistamientos() {...}

    /** Reinicia
     */
    public void reinicia() {...}

    /** Obtiene el avistamiento actual
     */
    public Avistamiento actual() {...}


    /** Avanza al siguiente avistamiento
     */
    public void avanza() {...}

    /** fin de secuencia
     */
    public boolean fds() {...}
```

```
    /** Inserta un avistamiento nuevo al final de la secuencia
     */
    public void escribeAlFinal(Avistamiento a) {...}
}
```

Lo que se pide es escribir una clase llamada `Parque` que contiene como atributo un objeto de la clase `SecuenciaAvistamientos` con la secuencia de avistamientos de aves en un parque natural, y que tiene también operaciones para trabajar con esta secuencia. La secuencia no tiene porqué estar ordenada. La clase contiene el siguiente atributo y las siguientes cabeceras de métodos:

```
public class Parque
{
    // la secuencia de avistamientos
    private SecuenciaAvistamientos sec;

    /** Constructor que hace la secuencia vacía
     */
    public Parque(){...}

    /**
     * Inserta un avistamiento si es correcto. Retorna true si el
     * avistamiento se ha insertado y false si no se ha insertado
     * porque es incorrecto. El avistamiento es correcto si mes está
     * entre 1 y 12, y el número de ejemplares es >0
     */
    public boolean inserta(Avistamiento av) {...}

    /** Retorna un array que contiene todos aquellos avistamientos de la
     * secuencia cuyo número de ejemplares (obtenido con numEjemplares())
     * es mayor que el parámetro numero
     */
    public Avistamiento[] masNumerosos(int numero) {...}

    /**
     * Retorna la suma total total de los números de ejemplares de la especie
     * indicada vistos entre todos los avistamientos comprendidos
     * entre los meses mesInic y mesFin, ambos inclusive
     */
    public int vistosEnPeriodo(String especie, int mesInic, int mesfin)
    {...}

    /**
     * Retorna un booleano que indica si la especie indicada ha sido vista,
     * o no, en el mes indicado alguna vez
     */
    public boolean vistoEnMes(String especie, int mes) {...}

}
```

Se pide implementar en Java los métodos de la clase `Parque`, según lo descrito en sus comentarios de documentación, y teniendo en cuenta lo siguiente:

1) Constructor: (0.2 puntos)

2) `inserta`: (0.8 puntos)

3) `masNumerosos`: requiere un doble recorrido de la secuencia; uno para saber cuántos elementos tendrá el array que se retorna, y otro para rellenarlo (1.5 puntos)

4) `vistosEnPeriodo`: usar el esquema de recorrido en secuencias (1.5 puntos)

5) `vistoEnMes`: usar el esquema de búsqueda en secuencias (1 punto)