Examen de Programación I
(Ingeniería Informática)

Febrero 2008

Primera parte (4 puntos, 40% nota del examen)

1) Escribir un diseño iterativo para la siguiente especificación:

```
método buscaPalabra
    (caracter[1..N] texto, caracter[1..M] palabra
     retorna booleano
  {Pre: N>=M, M>=1}
       busca si la palabra está o no en el texto
  {Post: retorna true si texto contiene a palabra,
          es decir si existe un entero k tal que
                ∀(j ∈ [1...M]) , texto[k + j – 1] = palabra[j]
   retorna false en caso contrario }
fmétodo
```

Utilizar para este diseño el siguiente invariante:

1<=i<=N+1, los caracteres de texto anteriores a i no contienen la palabra
1<=j<=M+1, los caracteres de palabra anteriores a j coinciden con los caracteres de texto anteriores a i
encontrado es un booleano que indica si los M caracteres de texto anteriores a i coinciden con palabra

La condición de permanencia en el bucle es

i<N+1 y no encontrado

En el cuerpo del bucle se debe comparar texto[i] con palabra[j]. Si son iguales, avanzar j y si se ha llegado al final de la palabra hacer encontrado=true. Si no son iguales, hacer j=1. Además, en todos los casos, avanzar la i.

2) Se dispone de la siguiente clase:

```
clase  Persona
  atributos
     Texto nombre;
     carácter genero; // 'H': hombre, 'M': mujer
     entero edad;
     real altura; // metros
     real peso;   //Kg
  fatributos

  método altura() retorna real
    {Pre:}
        retorna la altura en m
    {Post: valor retornado=altura}
  fmétodo

  método peso() retorna real
    {Pre:}
        retorna el peso en Kg
    {Post: valor retornado=peso}
  fmétodo
```

```
método cumplePerfil (carácter gen, carácter grupoEdad) retorna booleano
  {Pre: gen es 'H' o 'M', grupoEdad es  'N', 'J', 'A', o 'M'}
       Comprueba si la persona cumple con el perfil de género y grupo de edad
  {Post: retorna true si y sólo si gen==genero y además se cumple:
      si grupoEdad=='N',  edad<16,
      si grupoEdad=='J', 16<=edad<26,
      si grupoEdad=='A', 26<=edad<66,
      si grupoEdad=='M',  edad>=66}
  fmétodo
fclase
```

Escribir en pseudocódigo un diseño para el método `cumplePerfil`, utilizando instrucciones alternativas. *Nota*: las letras 'N', 'J', 'A', o 'M' del grupo de edad corresponden respectivamente a los grupos niño, joven, adulto, o mayor.

3) Escribir una especificación y un diseño para un método externo a la clase `Persona` de la cuestión 2. Al método se le pasa una secuencia de objetos de esa clase, y debe devolver la primera persona de la secuencia que cumpla la propiedad de que su índice de masa corporal (IMC) sea menor que `imcMax`. Suponer que la secuencia sigue la interfaz vista en clase. Utilizar el esquema de búsqueda en secuencia visto en clase. La cabecera del método será:

```
método buscaIMCMenorQue
    (Secuencia<Persona> sec, real imcMax) retorna Persona
```

El índice de masa corporal se calcula como

$$IMC = \frac{peso}{altura^2}$$

4) Se desea diseñar un método que permita obtener a partir de una secuencia original de números reales otra nueva secuencia de números reales que contiene aquellos valores de la original cuya diferencia relativa con la media, en valor absoluto, no sea superior al 20%. Es decir, tratamos de descartar en la nueva secuencia los valores demasiado alejados de la media.

Para conseguir este objetivo, se opta por hacer un diseño recursivo de un método que trabaja con el elemento actual de la secuencia. El método tiene la siguiente cabecera:

```
método obtenerBuenos
    (Secuencia<real> original, Secuencia<real> nueva, double media)
```

El caso directo corresponde a la situación en que la secuencia original está finalizada (fin de secuencia). En este caso no hay que hacer nada.

En el caso recursivo, llamaremos *actual* al elemento actual de la secuencia original. Si $|(actual - media)/(media)| < 0.2$ se añade *actual* a la secuencia nueva y se actualiza la media como $media = media \cdot 0.9 + actual \cdot 0.1$. Además, se avanza la secuencia original y luego se llama recursivamente al mismo método con las mismas secuencias original y nueva, y con la media actualizada.

Examen de Programación I
(Ingeniería Informática)

Febrero 2008

Segunda parte (6 puntos, 60% nota del examen)

Se desea escribir una clase Java que ayude en los cálculos necesarios de las trayectorias de un robot móvil que se mueve en un plano X-Y. La clase se llama `Trayectoria` y tiene dos tablas que contienen respectivamente las coordenadas X e Y de los puntos de la trayectoria que debe seguir el robot. El punto i-ésimo tiene su coordenada X en `x[i]` y su coordenada Y en `y[i]`. Además, la clase guarda el punto actual, al que se debe dirigir el robot inmediatamente (coordenadas `actualX` y `actualY`), y el índice del punto objetivo de la trayectoria, que es el punto de la trayectoria al que debe dirigirse el robot a medio plazo. El punto actual no tiene por qué ser un punto de la trayectoria; habitualmente será un punto intermedio entre dos puntos de la trayectoria. Los atributos y cabeceras de métodos se indican a continuación:

```
/**
 * Clase que contiene una trayectoria en el plano X-Y
 */
public class Trayectoria
{
    // puntos de la trayectoria
    private double x[]; // milímetros
    private double y[]; // milímetros

    // punto a donde nos queremos dirigir ahora mismo
    // (punto actual)
    private double actualX; // milímetros
    private double actualY; // milímetros

    // indice del punto de la trayectoria al que queremos llegar
    // (punto objetivo)
    private int objetivo;

    /**
     * Constructor de la trayectoria
     */
    public Trayectoria(double x[], double y[],
                       double actualX, double actualY) {...}

    /**
     * Número de puntos
     */
    public int numPuntos() {...}

    /**
     *  Coordenada x del punto
     *  a donde nos queremos dirigir ahora mismo
     */
    public double coordX() {...}

    /**
     *  Coordenada y del punto
     *  a donde nos queremos dirigir ahora mismo
     */
    public double coordY() {...}

    /**
     * Comprueba si hay puntos seguidos repetidos
     * Se consideran dos puntos iguales si están
     * a menos de 0.1 milímetros de distancia
     */
    public boolean hayRepetidosSeguidos() {...}
```

```
    /**
     * Distancia de (px,py) al punto i-ésimo de la trayectoria
     */
    public double distancia(double px, double py, int i) {...}

    /**
     * Calcula el nuevo punto objetivo, como el indice
     * del punto de la trayectoria mas cercano
     * a (px,py). Además, lo retorna
     */
    public int masCercano(double px, double py)  {...}

    /**
     * Si el punto actual está cerca del objetivo, y éste
     * no está al final de la trayectoria, incrementar el índice
     * del punto objetivo
     */
    public void actualizaObjetivo() {...}

    /**
     * Calcula el siguiente punto al que hay que dirigirse
     * durante el proximo intervalo tiempo
     * para seguir la trayectoria hacia el punto objetivo
     * a la velocidad indicada en mm/s
     */
    public void avanza(double velocidad, double tiempo) {...}
}
```

Se pide implementar en Java los siguientes métodos:

1) `hayRepetidosSeguidos` (1.5 puntos): Retorna `true` si dos puntos consecutivos de la trayectoria están repetidos, y `false` en caso contrario. Se considera que están repetidos si la distancia entre ellos es menor que 0.1 mm. Usar el esquema de recorrido (parcial) de una tabla.

2) `distancia` (0.5 puntos): Retorna la distancia entre el punto (px,py) y el punto i-ésimo de la trayectoria. La distancia entre dos puntos (x1,y1) y (x2,y2) se calcula como:

$$dist = \sqrt{(x2-x1)^2 + (y2-y1)^2}$$

3) `masCercano` (2 puntos): Calcula el índice del punto de la trayectoria más cercano a (px,py). Después de calcularlo, lo guarda en el atributo objetivo, y lo retorna.

4) `avanza` (2 puntos): Calcula el siguiente punto actual al que hay que dirigirse durante el proximo intervalo de tiempo para seguir la trayectoria hacia el punto objetivo a la velocidad indicada, en mm/s. Para ello, si la distancia entre el punto actual y el punto objetivo es menor que velocidad*tiempo, hace el punto actual igual al punto objetivo. Si no, hace los siguientes cálculos:
   - calcula las coordenadas x e y del vector que va del punto actual al punto objetivo (coordenadas del punto objetivo menos coordenadas del punto actual)
   - calcula el módulo de ese vector (raíz cuadrada de los cuadrados de las coordenadas)
   - divide cada coordenada del vector por el módulo, para obtener un vector unitario
   - multiplica cada coordenada del vector por velocidad*tiempo
   - añade el vector al punto actual (añadiéndole las respectivas coordenadas x e y)