Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Junio 2004

Cuestiones (5 cuestiones, 5 puntos en total)

1) Crear una clase que sirva para almacenar una lista de nombres (Strings) en un array. Tendrá un único atributo privado: el array de nombres. Tendrá también un constructor público al que se le pasa como parámetro el número máximo de nombres. El constructor crea el array con un tamaño igual a ese número máximo, y pone cada uno de los nombres igual a un `String` vacío.

2) Añadir a la clase del problema anterior un par de métodos públicos. El primero debe retornar el nombre almacenado en la casilla `i`, siendo `i` un entero que se le pasa como parámetro al método. El segundo método tiene como parámetros un entero (llamado `i`) y un `String`, y debe poner el nombre número `i` al valor indicado en el segundo parámetro.

3) El siguiente algoritmo permite multiplicar dos matrices cuadradas de tamaño *n* por *n*. Indicar razonadamente su tiempo de ejecución según la notación *O*(*n*):

```
multiplica a[n][n] por b[n][n] y guarda resultado en c[n][n]:
   lazo desde i=0 hasta n-1
      lazo desde j=0 hasta n-1
          c[i][j]=0
          lazo desde k=0 hasta n-1
              c[i][j]=c[i][j]+a[i][k]*b[k][j]
          fin de lazo
       fin de lazo
   fin de lazo
fin de algoritmo
```

4) Escribir un método al que se le pasa como parámetro un `String`, lo convierte a número real con el método `parseDouble`() de la clase `Double`, y retorna ese número real. Si durante la conversión se lanzase la excepción `NumberFormatException`, se debe devolver, en cambio, el número real más negativo.

5) Indicar razonadamente cuánto ocupan en memoria las siguientes variables. Asimismo, indicar para las cinco primeras el rango de valores que admiten.

```
int i;
byte b;
char c;
boolean cond;
double d;
float[] f=new float[1000];
String pepe="Pepe sale de casa";
double[][] m=new double [3][4];
```

Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Junio 2004

Problema (5 puntos)

Se dispone de una clase ya realizada que permite representar en una pantalla gráfica un mapa topográfico. La pantalla dispone de un lápiz óptico con el que se puede señalar un punto, y existe un método para leer ese punto (`posLapiz()`). La clase obedece a la siguiente especificación:

```
public class Pantalla {
    /**
     * Constructor que requiere el nombre del mapa
     */
    public Pantalla(String nombreMapa) {...}

    /**
     * Cambia el zoom con el que se ve el mapa en la pantalla
     */
    public void Zoom(double nuevoZoom) {...}

    /**
     * Reposiciona la pantalla con el nuevo punto como centro
     */
    public void centra (Utm nuevoCentro) {...}

    /**
     * Retorna la posición actual del lápiz óptico en coordenadas Utm
     * Lanza NoActivo si el lápiz no está activo
     */
    public Utm posLapiz() throws NoActivo {...}

    /**
     * Retorna la altitud de un punto del mapa, en metros
     */
    public double altitud(Utm punto) {...}

}
```

Existen dos excepciones escritas en clases aparte de la forma:

```
public class NoCabe extends Exception {}

public class NoActivo extends Exception {}
```

Los puntos del mapa representado en la clase `Pantalla` se especifican en coordenadas UTM, para lo que se utiliza la clase siguiente. Observar que sus atributos son públicos, ya que es una clase muy simple:

```
public class Utm
{
    public double x, y;  // Coordenadas Utm

    /**
     * Constructor al que se le pasan las dos coordenadas
     */
    public Utm(double x, double y) {...}

    /**
     * Retorna la distancia en metros entre el puntos actual y
     * el indicado por "otro"
     */
    public double distancia (Utm otro) {...}

 }
```

Lo que se pide es crear a partir de estas clases una nueva clase denominada `Recorrido` para almacenar una lista de puntos del plano junto a sus altitudes (obtenidas con el método `altitud`()). Estos puntos representan un recorrido que se define sobre el mapa y que se obtiene leyendo sucesivas posiciones del lápiz óptico. La clase permite calcular la distancia total y el desnivel de subida del recorrido almacenado. Su especificación deberá responder a:

```
public class Recorrido
{
    /**
     * Constructor al que se le pasa el maximo numero de puntos
     * y un objeto de la clase Pantalla
     */
    public Recorrido(int maxPuntos, Pantalla p) {...}

    /**
     * Intenta anadir un nuevo punto; retorna true si lo consigue
     * y false si no. Lanza NoCabe si ya no caben más puntos
     */
    public boolean anadePunto() throws NoCabe {...}

    /**
     * Distancia total del recorrido, en metros
     */
    public double distanciaTotal() {...}

    /**
     * Desnivel de subida total del recorrido, en metros
     */
    public double subidaTotal() {...}
}
```

La clase deberá tener los siguientes atributos privados:

- `puntos`: array de objetos de la clase `Utm`, que servirá para almacenar las coordenadas UTM de los puntos del recorrido.
- `altura`: array de números reales para almacenar la altitud de los puntos en metros; guardaremos la altitud del punto número `i` del array `puntos` en la casilla `i` del array `altura`.
- `numPuntos`: número actual de puntos almacenados en el recorrido; inicialmente es cero.
- `maxPuntos`: número máximo de puntos que caben en el recorrido.
- `p`: objeto de la clase `Pantalla` que representa el mapa utilizado en el recorrido.

Por otro lado, hay que implementar cada uno de los métodos para que realicen lo siguiente:

- Constructor: Guarda los argumentos en los respectivos atributos. Pone `numPuntos` a cero. Crea los dos arrays con un tamaño igual `maxPuntos`.
- `anadePunto`(): Si el número actual de puntos almacenados es ya igual al máximo, lanza la excepción `NoCabe`. En caso contrario, lee las coordenadas del punto señalado por el lápiz óptico (con `posLapiz`()), guarda ese punto en la primera casilla libre del array `puntos` y su altitud en la casilla correspondiente del array `altura`. Además, incrementa el atributo `numPuntos` en una unidad y, si todo ha ido bien, retorna `true`. Si en cambio se lanza `NoActivo`, retorna `false`.
- `distanciaTotal`(): Suma las distancias entre cada punto del recorrido y el siguiente y retorna el resultado de esa suma. La clase `Utm` tiene una función para calcular la distancia entre dos puntos.

- `subidaTotal`(): Retorna el desnivel de subida total, obtenido al aplicar el siguiente algoritmo, en el que sólo se tienen en cuenta los tramos de subida:

```
sube=0
anterior=altura del primer punto
lazo para cada punto i desde el segundo al último:
   si altura del punto i > anterior entonces
       sube=sube+(altura del punto i)-anterior
   fin de si
   anterior=altura del punto i
fin del lazo
retorna sube
```