

Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Junio 2005

Cuestiones (5 cuestiones, 5 puntos en total)

- 1) Crear una clase para guardar unos datos de las medidas de la intensidad de la radiación medida en un recorrido. Cada dato de intensidad asociado con una posición geográfica se guarda en un objeto del tipo `DatoRadiacion` que está definido como:

```
public class DatoRadiacion
{
    public double latitud, longitud; //grados
    public double intensidadRadiacion; //becquerels
}
```

La nueva clase debe tener un atributo que sea un array de objetos de la clase `DatoRadiacion`, y un entero que indique cuántos datos hemos almacenado.

La clase debe tener un constructor al que se le pasa el tamaño máximo del array. Debe crear el array y poner el número de datos a cero.

- 2) Añadir a la clase anterior un método que calcule la intensidad de radiación media para todos los datos almacenados. Si el número de datos es nulo, debe lanzar la excepción `NoHayDatos`, declarada en una clase aparte de la forma siguiente:

```
public class NoHayDatos extends Exception {}
```

- 3) Indicar usando la notación $O(n)$ el tiempo de ejecución del algoritmo `Busca_En_Lista`, suponiendo que el número de datos en la lista es n :

```
Algoritmo Busca_En_Lista (String nombre, Lista de nombres lista)
retorna los datos personales de una persona
comienzo
    si nombre > nombre del elemento (n/2) de lista entonces:
        lazo para i desde 1 hasta n/2
            si nombre = nombre del elemento i de la lista entonces:
                retorna elemento i de la lista
            fin de si
        fin de lazo
        retorna no encontrado
    si no:
        lazo para i desde n/2+1 hasta n
            si nombre = nombre del elemento i de la lista entonces
                retorna elemento i de la lista
            fin de si
        fin de lazo
        retorna no encontrado
    fin de si
fin Busca_En_Lista
```

- 4) Se dispone de la siguiente clase que almacena los nombres de un equipo de fútbol de 16 jugadores:

```
public class Equipo
{
    public String[] nombreJugador=new String[16];
}
```

Añadirle un método que lea los nombres de los 16 jugadores de un fichero de texto cuyo nombre se pasa como parámetro. El nombre de cada jugador esta en una línea. Tratar por separado las excepciones de "fichero no encontrado" e `IOException`.

- 5) Indicar qué órdenes utilizarías en una *shell* de un sistema operativo Unix para copiar todos los ficheros acabados en ".java" y todos los acabados en ".class" desde el directorio `/usr/pepe/programas` al directorio `copia_seguridad` que está colocado en tu directorio inicial de usuario.

Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Junio 2005

Problema (5 puntos)

Se desea hacer un programa para mover un telescopio a la vez que una cubierta giratoria que lo protege. El telescopio se controla mediante la clase Telescopio ya implementada:

```
public class Telescopio
{
    /**
     * Inicia el movimiento del telescopio hacia las coordenadas
     * indicadas en grados
     */
    public void mueve(double acimut, double elevacion) {...}

    /**
     * Retorna el acimut actual del telescopio
     */
    public double acimutActual() {...}

    /**
     * Retorna la elevacion actual del telescopio
     */
    public double elevacionActual() {...}
}
```

La cubierta dispone de un sensor de lluvia representado por la clase SensorLluvia, con un método para saber si llueve o no:

```
public class SensorLluvia
{
    /**
     * Indica si llueve o no
     */
    public boolean llueve() {...}
}
```

La cubierta dispone de una ventana que debe abrirse para poder usar el telescopio, y cerrarse si llueve. Además puede girar un ángulo comprendido entre un tope inferior y un tope superior. Está representada por la clase Cubierta ya realizada:

```
public class Cubierta
{
    /**
     * Constructor al que se le pasa el sensor de lluvia
     */
    public Cubierta (SensorLluvia sensor) {...}

    /**
     * Retorna el sensor de lluvia asociado a esta cubierta
     */
    public SensorLluvia sensor() {...}

    /**
     * Retorna el estado de los topes: SUPERIOR, INFERIOR,
     * o NINGUNO, en caso de que no se haya alcanzado ninguno
     */
    public Tope estadoTopes() {...}

    /**

```

```

    * Hace que el motor gire segun la consigna indicada:
    * POSITIVO, NEGATIVO, o PARADO
    * Si la ventana esta abierta y llueve, lanza ErrorLlueve
    */
    public void actuaMotor(Consigna con) throws ErrorLlueve {...}

    /**
    * Abre (con abierta=true) o cierra (con abierta=false)
    * la ventana de la cubierta. Si se pide abrirla y llueve,
    * lanza ErrorLlueve
    */
    public void actuaVentana(boolean abierta) throws ErrorLlueve {...}

    /**
    * Retorna la posición de la cubierta, en grados
    */
    public double angulo() {...}

    /**
    * Indica si la cubierta esta abierta (true) o cerrada (false)
    */
    public boolean ventanaAbierta() {...}
}

```

En esta clase hemos usado un par de tipos enumerados y una excepción, definidos en clases aparte:

```

public enum Tope
{
    SUPERIOR, INFERIOR, NINGUNO
}

public enum Consigna
{
    POSITIVO, NEGATIVO, PARADO
}

public class ErrorLlueve extends Exception {}

```

Lo que se pide es implementar la clase Observatorio, que integra el telescopio y la cubierta, y que debe tener la siguiente interfaz:

```

public class Observatorio
{
    public Observatorio(Telescopio tele, Cubierta cub) {...}
    /**
    public void mueve (double acimut, double elevacion) {...}

    public void cierra() {...}
}

```

La clase Observatorio debe contener los siguientes elementos (se valorarán en función de su dificultad):

- Atributos: Tres atributos privados que son referencias a objetos de las clases Telescopio, SensorLluvia, y Cubierta.
- Constructor: Copia los parámetros en los atributos respectivos, y copia en el atributo de la clase SensorLluvia la referencia al sensor de lluvia obtenida del objeto cub.
- Método mueve(): Este método hace lo siguiente:
 - Mueve el telescopio a la posición indicada por los parámetros.
 - Si la diferencia entre el acimut y la posición de la cubierta es mayor de 10 grados en valor absoluto, mueve la cubierta (con el método mueveCubierta() descrito abajo) a un valor igual al acimut.

- Si la cubierta está cerrada y no llueve, se abre.
- Si se lanza la excepción `ErrorLlueve`, cerrar la ventana y dejar el motor de la cubierta parado. Observar que al hacer estas acciones puede volverse a lanzar la excepción. En ese caso, tratar la excepción pero no hacer nada.
- Método `cierra()`: Si la ventana estaba abierta, la cierra. Además, pone el telescopio en posición de reposo (acimut 180.0° , elevación 45.0°) y la cubierta también ($\text{ángulo}=180.0^\circ$). Para mover la cubierta se usa el método `mueveCubierta()` descrito abajo. Si se lanza `ErrorLlueve`, dejar el motor de la cubierta parado y observar lo mismo que en tratamiento de esta excepción en el método anterior.
- Método privado `mueveCubierta()`: Se le pasa como parámetro el ángulo deseado para la cubierta. Hace lo siguiente:
 - Si el ángulo actual de la cubierta es mayor que el ángulo deseado, se hace girar el motor de la cubierta en sentido negativo, y se anota que el tope de recorrido a comprobar es el inferior. En caso contrario, se hace girar el motor de la cubierta en sentido positivo, y se anota que el tope de recorrido a comprobar es el superior.
 - Se entra en un lazo que se ejecuta mientras la diferencia entre el ángulo actual de la cubierta y el ángulo deseado sea en valor absoluto mayor que 1.0° . Si durante la ejecución de lazo observamos que llegamos al tope a comprobar, nos saldremos del lazo.
 - Al finalizar el lazo, paramos el motor de la cubierta.
 - El método no trata excepciones.