Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2004

Cuestiones (5 cuestiones, 5 puntos en total)

1) Crear una clase que permita almacenar datos de un animal doméstico. Deberá tener atributos privados para guardar el tipo de animal (entero), el nombre (texto), el número de vacunas aplicadas (entero) y un array para guardar hasta 10 vacunas (cada una representada por un texto). Asimismo escribir el constructor al que se le pasarán como parámetros el tipo de animal y su nombre, para copiarlos en los respectivos atributos; también debe poner el número de vacunas a cero.

2) Añadir a la clase anterior un método al que se le pase como parámetro el nombre de un fichero de texto. El método debe escribir en ese fichero los datos del animal, con un formato igual al del siguiente ejemplo, en el que había dos vacunas almacenadas (por cada vacuna almacenada se pondrá una línea similar a las dos últimas):

```
Tipo : 3
Nombre : Kira
Vacuna 1 : Rabia
Vacuna 2 : Triple
```

3) Indicar usando la notación $O(n)$ el tiempo de ejecución del siguiente algoritmo que sirve para buscar coincidencias de nombres y fechas en un array de fichas personales, donde cada persona tiene como atributos públicos su apellido, nombre y fecha de nacimiento:

```
busca_coincidencias (Persona p, Persona[] lista)
   apellido_igual=0
   nombre igual=0
   fecha_igual=0
   n=lista.length

   lazo para i desde 0 hasta n-1
       si p.apellido igual a lista[i].apellido entonces
          incrementa apellido_igual
   fin de lazo
   lazo para j desde 0 hasta n-1
       si p.nombre igual a lista[j].nombre entonces
          incrementa nombre_igual
       si p.fecha_nac igual a lista[j].fecha_nac entonces
          incrementa fecha_igual
   fin de lazo
   muestra apellido_igual, nombre_igual y fecha_igual
fin
```

4) Indica en **seis líneas** o menos cuáles son los objetivos y las ventajas de la programación orientada a objetos.

5) Modificar el siguiente método de la clase `Vector` para que lance la excepción `No_Iguales` si las longitudes de los vectores no son iguales.

```java
public class Vector {

   private double vec[];

   // Calcular el producto escalar
   public static double prodEscalar(Vector v1, Vector v2) {
      double producto=0.0;

      for (int i=0; i<v1.vec.length; i++) {
         producto=producto+v1.vec[i]*v2.vec[i];
      }
      return producto;
   }
}
```

La excepción está declarada en una clase aparte de la forma siguiente:

```java
  public class NoIguales extends Exception {}
```

Examen de Fundamentos de Computadores y Lenguajes

Examen Final. Septiembre 2004

Problema (5 puntos)

Se desea escribir un programa que gestiona el funcionamiento de un robot de venta de productos que están en una estantería organizada en varias filas y columnas.

Se dispone de dos clases ya realizadas que corresponden al robot y al teclado o panel de control a través del cual el usuario selecciona el producto que desea:

```
/** Representa un robot capaz de desplazarse delante de una
    estanteria y recoger un objeto del estante que tiene delante
*/
public class Robot {

  /** Mueve el robot al punto deseado. La coordenada Y es la altura
   */
  public void mueve(double coordX, double coordY)  throws NoAlcanzable
  {...}

  /** Indica si se ha alcanzado (o no) el punto destino
   */
  public boolean destinoAlcanzado() {...}

  /** Recoge un objeto de la estanteria
   */
  public void recogeObjeto() {...}

}


public class Teclado {

  /** Indica si se ha pulsado o no una tecla
   */
  public boolean hayTeclaPulsada() {...}

  /**Indica la fila de la tecla pulsada. Lanza NoPulsada
   * si no hay tecla pulsada
   */
  public int filaTeclaPulsada() throws NoPulsada {...}

  /**Indica la columna de la tecla pulsada. Lanza NoPulsada
   * si no hay tecla pulsada
   */
  public int columnaTeclaPulsada() throws NoPulsada {...}

  /** Estas operaciones encienden respectivamente la luz de error,
   *  preparado, en operacion, o averiado, apagando las otras tres
   */
  public void indicaError() {...}
  public void indicaPreparado() {...}
  public void indicaEnOperacion() {...}
  public void indicaAveriado() {...}
}
```

Las excepciones que se usan están declaradas en clases separadas, en la forma:

```
  public class NoAlcanzable extends Exception {}
  public class NoPulsada extends Exception {}
  public class Inexistente extends Exception {}
  public class NoHay extends Exception {}
```

La estantería tiene un número fijo de filas y columnas. Cada combinación de fila y columna representa un estante donde hay un cierto número de productos. La clase que representa la estantería tiene la siguiente interfaz:

```
public class Estanteria {

  /** Las operaciones de esta clase lanzan Inexistente si la fila indicada
   *  es mayor a maxFilas-1 o la columna indicada es mayor a maxColumnas-1
   */
  public static final int maxColumnas=30;
  public static final int maxFilas=10;

  /** Constructor al que se le pasan el robot y el teclado
   */
  public Estanteria(Robot rob, Teclado tec) {...}

  /** Retorna la posición central de una columna de la estanteria
   */
  public double posicion(int columna) throws Inexistente  {...}

  /** Retorna la altura de una fila de la estanteria
   */
  public double altura(int fila) throws Inexistente  {...}

  /** Retorna el numero de unidades de producto que hay en el estante de
   *  la columna y fila indicados
   */
  public int numUnidades(int columna, int fila) throws Inexistente  {...}

  /** Cambia el numero de unidades de producto que hay en el estante
   *  de la columna y fila indicados
   */
  public void rellenaProducto(int columna, int fila, int nuevaCantidad)
    throws Inexistente  {...}

  /** Quita una unidad al estante de la columna y fila indicadas.
   *  Lanza NoHay si el numero de unidades es cero
   */
  public void quitaUnidad(int columna, int fila) throws NoHay, Inexistente
  {...}

  /** Esta operacion realiza una accion de compra de un objeto
   */
  public void accionCompra() {...}

}
```

Lo que se pide es realizar parte de esta clase. En primer lugar se pide escribir los atributos, todos privados. Serán los siguientes:

- `num`: un array bidimensional para guardar el número de unidades de cada estante. La primera dimensión o índice representa la fila (entre `0` y `maxFilas-1`) y la segunda dimensión la columna (entre `0` y `maxColumnas-1`)
- `rob`: una referencia a un objeto de la clase `Robot`
- `tec`: una referencia a un objeto de la clase `Teclado`

También se pide escribir el constructor, el método `quitaUnidad`() y el método `accionCompra`().

El constructor debe hacer lo siguiente:

- copiar los parámetros en los atributos correspondientes
- crear el array `num` con el tamaño indicado en la descripción de ese atributo.

El método `quitaUnidad`() debe hacer lo siguiente:

- Comprobar que la fila y la columna están comprendidas entre cero y `maxFilas-1` y entre cero y `maxColumnas-1`, respectivamente. En caso contrario lanzar `Inexistente`.
- Si el número de unidades de producto del estante correspondiente a la fila y la columna indicadas es cero, lanzar `NoHay`.
- Si no se ha lanzado ninguna de las dos excepciones, decrementar en una unidad el número de unidades de producto del estante correspondiente a la fila y la columna indicadas.

Por último, el método `accionCompra()` debe hacer lo siguiente:

- Encender la luz de "preparado" del teclado.
- Si hay tecla pulsada en el teclado (se sabe con `hayTeclaPulsada()`), seguir; si no, terminar el método.
- Leer la fila y columna de la tecla pulsada del teclado (con `filaTeclaPulsada()` y `columnaTeclaPulsada()`) y mover el robot a la posicion y altura de esa columna y fila, respectivamente. La posicion y altura se obtienen con los métodos del mismo nombre.
- Iniciar un lazo mientras el robot no haya alcanzado su destino (se sabe mediante `destinoAlcanzado()`). Dentro de este lazo, encender la luz "en operación" en el teclado.
- Al acabar el lazo anterior, recoger un objeto con el robot (`recogeObjeto()`) y quitar una unidad al producto (`quitaUnidad()`)
- Si en cualquier momento de la operación se lanza una de las excepciones `NoPulsada`, `NoAlcanzable`, o `Inexistente`, encender la luz "Averiado" y terminar el método.
- Si en cualquier momento de la operación se lanza la excepción `NoHay`, encender la luz "Error" y terminar el método.