Examen de Prácticas de Programación
Ingeniería Informática

Junio 2007

1) (2 puntos) Escribir en Java la implementación de los métodos
`public void escribeMatrizEnFichero(double[][] m, String nomFich) ...`
`public double[][] leeMatrizDeFichero(String nomFich) ...`

Los métodos deben servir para escribir y leer matrices de cualquier dimensión, se deja a elección del alumno la forma en que se almacenan las dimensiones de la matriz en el fichero.

El método de escritura deberá escribir los números con cuatro decimales y ocupando un número fijo de caracteres de forma que queden ordenados por columnas. El método de lectura debe ser capaz de leer ficheros generados por `escribeMatrizEnFichero` y también ficheros escritos "a mano" en los que no se respete esta ordenación por columnas.

Los métodos deberán detectar todos los errores que sea posible y lanzar las excepciones correspondientes.

Solución:

```
public void escribeMatrizEnFichero(double [][] m, String nomFich)
                                        throws IOException {
  PrintWriter sal = null;
  try {
    // abre el fichero de texto para escritura
    sal = new PrintWriter(new FileWriter(nomFich));

    // escribe el número de filas y columnas de la matriz en las
    // dos primeras líneas del fichero
    sal.println(m.length);    // filas
    sal.println(m[0].length); // columnas

    // escribe los números ordenados por columnas
    for(int f=0; f<m.length; f++) {
      for(int c=0; c<m[0].length; c++) {
        // escribe los números con 20 caracteres y 4 decimales
        sal.printf("%20.4f ",m[f][c]);
      }
      sal.println(); // fin de fila
    }
  } finally {
    if (sal != null) {
      // cierra el fichero
      sal.close();
  }
}
```

```
public double[][] leeMatrizDeFichero(String nomFich)
             throws ErrorEnDimensiones, ErrorEnComponentes,
                                    FileNotFoundException {
  double[][] m;
  Scanner ent = null;
  try {
    // abre el fichero de texto para lectura. Utilizamos la clase
    // Scanner porque facilita la lectura de los datos
    ent = new Scanner(new FileReader(nomFich));

    // lee las dimensiones de la matriz
    int numFilas, numColumnas;
    try {
      numFilas = ent.nextInt();
      numColumnas = ent.nextInt();
    } catch (Exception e) {
      throw new ErrorEnDimensiones();
    }

    // crea una matriz de las dimensiones apropiadas
    m = new double[numFilas][numColumnas];

    // lee cada uno de los componentes de la matriz
    try {
      for(int f=0; f<numFilas; f++)
        for(int c=0; c<numColumnas; c++)
          m[f][c]=ent.nextDouble();
    } catch (Exception e) {
      throw new ErrorEnComponentes();
    }


  } finally {
    if (ent!=null) {
      // si se abrió el scanner lo cierra
      ent.close();
    }
  }

  // retorna la matriz
  return m;
}
```

2) (1.5 puntos) Se dispone de la clase `Alumno` ya realizada cuya interfaz es la mostrada a continuación:

```
public class Alumno {
  public Alumno(String nombre, int curso, Estudios estudios)
  {...}

  otros métodos no relevantes para el problema planteado
}
```

Además existe la clase enumerada `Estudios`:

```
public enum Estudios {Físicas, Matemáticas, Informática}
```

Se pide implementar en Java un método que pida al usuario los datos de un alumno (utilizando una ventana de lectura del paquete fundamentos) y retorne un objeto de la clase `Alumno` con dichos datos. El método deberá permitir al usuario reintentar la entrada de datos hasta que sean correctos, es decir, hasta que el curso sea un número entre 1 y 5 y los estudios correspondan a uno de los tres valores válidos.

Solución:

```
public Alumno pideDatosAlumno() {
  String nombre=null; int curso=0; Estudios estudios=Físicas;
  String error;

  // crea la ventana de lectura
  Lectura l = new Lectura("Introduce datos");
  l.creaEntrada("Nombre", "");
  l.creaEntrada("Curso", "1");
  l.creaEntrada("Estudios", "Físicas");

  // bucle que se repite hasta que los datos sean correctos
  while(true) {
    try {
      l.esperaYCierra();
      // lee los datos introducidos por el usuario
      nombre=l.leeString("Nombre");
      curso=l.leeInt("Curso");
      estudios=Estudios.valueOf(l.leeString("Estudios"));
      if (curso >=1 && curso <= 5)
        break; // datos correctos: abandona el lazo
      // si llega hasta aquí es porque el curso no es correcto
      error = "Curso incorrecto";
    } catch (IllegalArgumentException e){
      // los estudios no son correctos
      error = "Estudios incorrectos";
    }
    // pone el mensaje de error y sigue en el lazo
    Mensaje msj = new Mensaje("Error");
    msj.escribe(error);
  }
  // crea el alumno y le retorna
  return new Alumno(nombre, curso, estudios);
}
```

3) (1.5 puntos) Escribir la salida por consola que se obtendría con la ejecución del método `main` de la clase `ValorObjs`. Indicar de forma clara las líneas en blanco que formen parte de la salida (por ejemplo sustituyéndolas por una línea horizontal).

```
public class MiClase {
  public double d;
  public Integer i;

  public MiClase(double d, Integer i) {
    this.d=d;
    this.i=i;
  }

  public MiClase() {
  }

  public void muestra(String msj) {
    System.out.println(msj+": (d:"+d+" i:"+i+")");
  }
}

public class ValorObjs {
    public static void main() {
        MiClase obj1=new MiClase();
        MiClase obj2=new MiClase(1.0, new Integer(2));

        obj1.muestra("obj1");
        obj2.muestra("obj2");

        MiClase[] arrayObjs=new MiClase[2];
        int[] arrayInts=new int[2];

        System.out.println("\narrayObjs[0]:"+arrayObjs[0]+
            " arrayObjs[1]:"+arrayObjs[1]);
        System.out.println("arrayInts[0]:"+arrayInts[0]+
            " arrayInts[1]:"+arrayInts[1]);

        arrayObjs[0]=new MiClase(2.0, new Integer(3));
        arrayObjs[1]=obj2;

        obj1.muestra("\nobj1");
        obj2.muestra("obj2");
        arrayObjs[0].muestra("arrayObjs[0]");
        arrayObjs[1].muestra("arrayObjs[1]");

        obj1.d=11.0;
        obj2.d=12.0;

        obj1.muestra("\nobj1");
        obj2.muestra("obj2");
        arrayObjs[0].muestra("arrayObjs[0]");
        arrayObjs[1].muestra("arrayObjs[1]");

        for(int i: arrayInts)
            i+=1;
        System.out.println("\narrayInts[0]:"+arrayInts[0]+
            " arrayInts[1]:"+arrayInts[1]);
    }
}
```

Solución:

```
obj1: (d:0.0 i:null)
obj2: (d:1.0 i:2)

arrayObjs[0]:null arrayObjs[1]:null
arrayInts[0]:0 arrayInts[1]:0

obj1: (d:0.0 i:null)
obj2: (d:1.0 i:2)
arrayObjs[0]: (d:2.0 i:3)
arrayObjs[1]: (d:1.0 i:2)

obj1: (d:11.0 i:null)
obj2: (d:12.0 i:2)
arrayObjs[0]: (d:2.0 i:3)
arrayObjs[1]: (d:12.0 i:2)

arrayInts[0]:0 arrayInts[1]:0
```

4) (5 puntos) Se pretende desarrollar un programa que permita gestionar el préstamo de las películas de un vídeo club. Para ello se dispone de las clases `Película` y `Cliente`, ya realizadas y cuya interfaz es:

```
public class Película {
   /** crea una película en estado no prestada */
   public Película(String título) {...}
   /** retorna el título de la película */
   public String título() {...}
   /** presta la película al cliente indicado, si la película ya
    *  se encontraba prestada lanza YaPrestada */
   public void presta(Cliente cliente) throws YaPrestada{...}
   /** marca una película prestada como devuelta. Si no se
    *  encontraba prestada lanza NoPrestada, y se encontraba
    *  prestada, pero a un cliente distinto de "cliente" lanza
    *  ClienteIncorrecto */
   public void devuelve(Cliente cliente) throws NoPrestada,
     ClienteIncorrecto {...}
}
```

No va a ser necesario utilizar ningún método de la clase `Cliente`, por lo que no se muestra su interfaz.

Se pide implementar en Java la clase `VídeoClub` con la siguiente funcionalidad:

- Atributos: cuatro listas enlazadas (`LinkedList`), cada una para almacenar las películas correspondientes a un género (ver clase enumerada `Género`). Dentro de las listas las películas deberán estar ordenadas por orden alfabético en base a su título (de dicha ordenación se encarga el método `añadePelícula`).
- método público `añadePelícula`: recibe como argumentos la película (objeto de clase `Película`) y su género (valor de la clase `Género`). Añade la película a la lista correspondiente a su género en la posición que le corresponde por orden alfabético (utilizar el método `compareTo` de los strings para comparar por orden alfabético los títulos de las películas).
- método privado `buscaPelícula`: recibe como argumentos el título de la película (`String`) y su género (valor de la clase `Género`) y busca la película en la lista correspondiente. Retorna la película buscada (objeto de tipo `Película`) o `null` en el caso de que no la encuentre. El método deberá sacar partido del hecho de que las listas están ordenadas alfabéticamente para detectar cuando una película no existe sin necesidad de recorrer toda la lista.
- método público `prestaPelícula`: recibe como argumentos el título de la película (`String`), su género (valor de la clase `Género`) y el cliente al que se va a prestar la película (objeto de la clase `Cliente`). Utiliza el método `buscaPelícula` para buscar la película y, en el caso de que exista, llama al método `presta` para la película encontrada. Si la película no existe lanza la excepción `NoExiste`. No deberá tratar la excepción YaPrestada (que puede ser lanzada por el método `presta` de la clase Película), sino que deberá dejar que se propague hacia fuera.
- En una implementación real la clase `VideoClub` tendría también métodos para eliminar una película del video club y para anotar que una película prestada ha sido devuelta, no se pide la realización de dichos métodos con el fin de limitar la duración del examen.

Las excepciones y la clase enumerada `Género` se encuentran declaradas en clases aparte:

```
public class ClienteIncorrecto extends Exception {}
public class NoPrestada extends Exception {}
public class YaPrestada extends Exception {}
public class NoExiste extends Exception {}
public enum Género {Drama, Acción, Terror, Comedia}
```

Solución:

```
import java.util.LinkedList;

public class VideoClub {

  // Listas de películas (una por género)
  private LinkedList<Película> pelisTerror =
    new LinkedList<Película>();
  private LinkedList<Película> pelisDrama =
    new LinkedList<Película>();
  private LinkedList<Película> pelisAcción =
    new LinkedList<Película>();
  private LinkedList<Película> pelisComedia =
    new LinkedList<Película>();

  /** añade una película a la lista correspondiente a su género
   *  en la posición que le corresponde por orden alfabético */
  public void añadePelícula(Película p, Genero género) {
    // elige la lista en función del género
    LinkedList<Película> lista = null;
    switch (género) {
    case Drama:
      lista=pelisDrama;
      break;
    case Terror:
      lista=pelisTerror;
      break;
    case Acción:
      lista=pelisAcción;
      break;
    case Comedia:
      lista=pelisComedia;
      break;
    }
    // busca la posición en la que hay que añadir la película. El
    // lazo se realiza hasta llegar al final de la lista o hasta
    // encontrar una película que, por orden alfabético, vaya
    // después de la película a añadir.
    int i=0;
    while (i<lista.size() &&
        p.título().compareTo(lista.get(i).título())<0) {
      i++;
    }
```

```
        // añade la película en al posición adecuada
        lista.add(i,p);
    }

    /** busca una película en la lista correspondiente al género
     *  indicado */
    private Película buscaPelícula(String títuloPelícula,
        Genero género) {
        // elige la lista en función del género
        LinkedList<Pelicula> lista = null;
        switch (género) {
        case Drama:
          lista=pelisDrama;
          break;
        case Terror:
          lista=pelisTerror;
          break;
        case Acción:
          lista=pelisAcción;
          break;
        case Comedia:
          lista=pelisComedia;
          break;
        }

        // busca una película con el título deseado
        for(Pelicula p:lista)
          if (p.título().equals(títuloPelícula)) {
            // película encontrada
            return p;
          }

        // si llega hasta aquí es porque no la ha encontrado
        return null;
    }


    /** Presta una película (identificada por su título y género) a
     *  un cliente */
    public void presta(String títuloPelícula, Genero género,
        Cliente cliente) throws YaPrestada, NoExiste {
        // busca la película
        Película p = buscaPelícula(títuloPelícula, género);

        // si no existe ninguna película con ese título
        // lanza NoExiste
        if (p==null)
          throw new NoExiste();

        // la película existe: se presta al cliente
        p.presta(cliente); // si la película ya está prestada este
                           // método lanza YaPrestada
    }
}
```