

Programación Avanzada

Convocatoria Extraordinaria de Septiembre de 2005
Ingeniería Técnica de Telecomunicación (ITT-ST e ITT-SE)

Alumno (apellidos y nombre)	D.N.I.	Prácticas entregadas	Firma
		<input type="checkbox"/> Simulín <input type="checkbox"/> Nada <input type="checkbox"/> Trabajo sobre	

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

1. Cuál es la finalidad del uso del juego de caracteres Unicode:
 - a) conseguir mayor rapidez a la hora de visualizar los caracteres en pantalla.
 - b) ninguna, es un código más, tal como el ASCII y el EBCDIC.
 - * c) conseguir la portabilidad de los programas.
2. ¿Podemos escribir programas en Java utilizando el formato .DOC de Microsoft Word?
 - a) Sí, puesto que Word es el procesador de texto más extendido.
 - * b) No, pues Word siempre introduce caracteres especiales en el archivo no reconocibles por el compilador.
 - c) Depende de si escribimos textos con formato (negrita, cursiva, etc.) o no; en el primer caso no se podrá, pero en el segundo sí.
3. ¿Qué sucede si en una sentencia de control de flujo "switch (expresión) ... case" no ponemos la sentencia "break;" al final de cada cláusula "case"?
 - a) El depurador no detendrá la ejecución, pues se considera un punto de ruptura o "breakpoint".
 - * b) La ejecución continuará con el siguiente "case", aunque la expresión de "switch" no se cumpla.
 - c) No hay que poner nunca "break", pues si lo hacemos, la ejecución saltará de nuevo a la sentencia "switch".
4. En Java, el mecanismo llamado "recolección de basura" o "garbage collection" consiste en:
 - * a) limpieza de la memoria utilizada por los objetos, para eliminar los objetos que ya no se utilicen.
 - b) compactación de cada uno de los objetos para que ocupen menos memoria, mediante ZIP, ARJ, ...
 - c) es una llamada al sistema operativo para que aborde las tareas menos utilizadas pero que consumen algo de tiempo del procesador (CPU).
5. Si al compilar un programa obtenemos el error "ERROR: NullPointerException", ¿a qué puede ser debido?.
 - a) En Java no están permitidos los punteros.
 - b) Estamos intentando acceder a una posición de la memoria sin inicializar.
 - * c) Estamos accediendo a un método o atributo de un objeto que aún no se ha creado.
6. ¿Cuál es la clase raíz de la jerarquía de clases de Java?
 - a) Main
 - * b) Object
 - c) Java
7. Para crear un array de enteros (int) de dos dimensiones, ¿cuál de las siguientes sentencias es incorrecta sintácticamente?
 - * a) int n = new int[10][10];
 - b) int n[][] = new int[10][10];
 - c) int n[][] = { { 1, 2 }, { 3, 4 } };
8. Cuando una clase utiliza (implementa) una interfaz:
 - * a) debe definir todos los métodos que se declaran en la interfaz, o bien, ser declarada como abstracta.
 - b) la clase puede definir solamente los métodos que le interesan, no es necesario definir todos los métodos que se declaran en la interfaz, y ya podemos crear objetos de la clase.
 - c) la clase no necesita definir ninguno de los métodos, pues los "hereda" de la interfaz; tan solo sobreescribir los métodos que necesiten cambiar de comportamiento.
9. En una aplicación en modo gráfico, ¿qué es un "contenedor"?
 - * a) un componente más, pero que puede contener a otros componentes
 - b) es una librería de componentes
 - c) donde se depositan los objetos que ya no se utilizan, resultado del proceso de "recolección de basura"
10. Para poder atender los eventos de una aplicación gráfica debemos crear un manejador de eventos y registrararlo. Para crear el manejador de eventos debemos:
 - a) implementar ciertas interfaces "Listener" (ActionListener, MouseListener, etc.)
 - b) heredar de ciertas clases adaptadoras "Adapter" (ActionAdapter, MouseAdapter, etc.)
 - * c) de cualquiera de las dos formas anteriores

Teoría - Cuestionario (0.2 cada una, total 4.0 puntos)

1. Define los conceptos de "compilador" y "linkador".
¿Son necesarios en Java?
El compilador si. El linkador no. Ver apuntes.

2. ¿Qué significado tiene el incluir un punto (.) en el contenido de la variable CLASSPATH?.
Ver apuntes.

3. Pon un ejemplo de las posibilidades que tenemos de insertar comentarios en el código fuente.
Ver apuntes.

4. Explica en que consiste la técnica del "casting".
Cuando lo usamos con tipos primitivos, hay situaciones en las que su uso es obligatorio y otras en las que es opcional. Explica cuando y pon un ejemplo de cada una.
Ver apuntes. En caso de perder precisión, el uso de "casting" es obligatorio.

5. Hay situaciones en las que las variables se inicializan automáticamente a un valor por defecto y hay otras situaciones en las que no ocurre esto.
¿Podrías decir una situación de cada caso?.
Ver apuntes.

6. Define el concepto de propiedad o atributo.
Ver apuntes.

7. ¿En qué consiste la "sobrecarga de operadores"?.
¿La soporta Java?.
Java no la soporta. Ver apuntes.

8. ¿Cómo podemos saber en Java si un objeto ya ha sido creado o no?. ¿Qué ocurre si intentamos acceder a algún atributo o ejecutar algún método de un objeto que aun no ha sido creado?.
Ver apuntes.

9. Describe cuando se ejecuta el destructor de una clase.
Ver apuntes.

10. ¿Podrías explicar por qué los arrays y los objetos se pasan siempre por referencia a los métodos?.
Ver apuntes.

11. Describe brevemente 2 formas de recorrer todos los elementos de un objeto de la clase Vector?.
(NOTA: no es válido decir "hacia arriba y hacia abajo", se trata de explicar los 2 mecanismos que nos ofrece el lenguaje Java).
Con get() y con una enumeración. Ver apuntes.

12. ¿Cómo se crean paquetes?. ¿Cómo se utilizan los paquetes?.
Ver apuntes.

13. Explica la diferencia existente entre excepciones implícitas y excepciones explícitas. ¿Qué tiene que ver la cláusula "throws" con todo esto?.
Ver apuntes.

14. Diferencias entre los componentes de los paquetes gráficos AWT y Swing. ¿Qué nombre reciben en cada caso?.
Peso pesado y peso ligero. Ver apuntes.

15. Cuando desarrollamos aplicaciones de tipo gráfico con AWT, ¿para qué se utilizan los gestores de esquemas o "Layout"?.. Enumera alguno.
Ver apuntes.

16. Describe brevemente los pasos a seguir para crear una barra de menú con varios menús, y que cada menú tenga varias opciones de menú, y cómo capturar sus eventos, es decir, realizar alguna acción cuando seleccionemos cualquier opción de algún menú.
Ver apuntes.

17. ¿Qué significa que un cuadro de diálogo se muestre en modo "modal" o "no modal"?.
Ver apuntes.

18. ¿En qué situaciones se ejecuta automáticamente el método paint()?. ¿Existe alguna forma de forzar su ejecución?. Razona las respuestas.
Ver apuntes.

19. En Java podemos almacenar y leer objetos completos a un fichero. ¿Cómo se llama esta técnica?. ¿Qué clases se utilizan en ambos casos?.
Seriación. Ver apuntes.

20. Describe brevemente la estructura mínima que debe tener un applet (paquetes a importar, clase a derivar, método principal,...).
Ver apuntes.

Programación (5.0 puntos)

1. (2.0 puntos)

Escribir un programa en Java llamado "Nota" que reciba desde la línea de órdenes (consola del DOS) dos valores numéricos con decimales desde 0.0 hasta 10.0, correspondientes a la nota del examen y la nota de prácticas de un alumno. El programa deberá calcular la nota final del alumno, obteniéndose con la media ponderada de ambas notas del siguiente modo: la nota del examen influye un 80 % de la nota final, y la nota de prácticas, un 20 %. Una vez calculada la nota final, el programa proporcionará por

pantalla la nota final calculada, así como su calificación en forma de cadena de texto según los valores siguientes:

Rango de valores	Calificación
$\geq 0.0 \dots < 5.0$	SUSPENSO
$\geq 5.0 \dots < 7.0$	APROBADO
$\geq 7.0 \dots < 9.0$	NOTABLE
$\geq 9.0 \dots < 10.0$	SOBRESALIENTE
=10.0	MATRICULA DE HONOR

Restricciones:

- El programa deberá comprobar que el número de argumentos recibido desde la línea de órdenes es correcto, esto es, sólo se reciben dos y sólo dos argumentos. En caso de no especificar los argumentos necesarios, o ser incorrectos, el programa deberá mostrar un mensaje indicativo con la forma de uso.

```
C:\>java Nota  
Uso: java Nota <Nota examen> <Nota prácticas>
```

- Así mismo, puesto que los argumentos recibidos serán cadenas de texto, necesitaremos convertirlos a valores numéricos float. Ver los consejos más abajo. En caso de teclear un valor incorrecto, o valores fuera del rango 0.0 a 10.0, el programa mostrará un mensaje y finalizará.
- La función main debe llamar a una función llamada "media" que calcula la nota final, tal y como se ha expuesto. Esta deberá recibir dos valores float, así como retornar también un valor float. Siempre se realizará media, es decir, no es necesario tener un mínimo en cada una de las dos partes (Observación: sólo para este ejercicio).
- La función main, una vez calculada la nota final, invocará a otra función llamada "calificación", que recibirá un valor float que representa la nota final calculada, y retornará una cadena de texto (String) con la calificación según la tabla anterior. Para esta función, el alumno podrá utilizar las sentencias de control de flujo que estime conveniente.
- Finalmente, la función main mostrará el resultado. Algunos ejemplos de uso válidos serían los siguientes:

```
C:\>java Nota 5.5 8.2  
Nota final: 6.04 (APROBADO)  
C:\>java Nota 9.3 9.1  
Nota final: 9.26 (SOBRESALIENTE)
```

Consejos:

- Para convertir una cadena de texto (String) a un valor numérico float se puede utilizar la función: `Float.parseFloat (cadena)`. Esta función puede generar la excepción "NumberFormatException", que será conveniente capturar. Esto se puede producir si el usuario, por ejemplo, teclea alguna letra, o una coma en vez del punto decimal.
- Aunque no es obligatorio, se valorará si, a la hora de visualizar el resultado, se utiliza un número fijo de decimales. Como ayuda se proporciona el siguiente ejemplo:

```
import java.text.*; // Ojo: colocar en el lugar adecuado  
float n= 15.5;  
DecimalFormat f = new DecimalFormat("##0.00");  
System.println ("Valor: " + f.format (n)); // Nos imprime: 15.50
```

Solución:

```
/*
 * =====
 * Titulo: Aplicación modo texto
 * Fichero: Nota.java
 * =====
 */
import java.text.*; // Para utilizar DecimalFormat
public class Nota {
    // Función principal
    public static void main(String[] args) {
        // 1. Comprobar el número de argumentos ( args.length == 2 )
        // 2. Obtener los dos valores a operar desde los argumentos con parseFloat
        // 3. Comprobar que el rango de valores de ambos valores (0.0 a 10.0)
        // 4. Calcular nota final llamando a la función "media"
        // 5. Calcular calificación llamando a la función "calificación"
        // 6. Mostrar resultados (println). Si se muestran con decimales, habrá
        //    que crear un objeto de la clase DecimalFormat (paquete java.text.*)
        // 7. Fin
    }

    // Calculamos la nota final
    private static float media(float notaExamen, float notaPrácticas) {
        return ( (notaExamen*0.8F) + (notaPrácticas*0.2F) );
    }

    // Calculamos calificación
    private static String calificación (float notaFinal) {
        if (notaFinal < 5.0) {
            return ("SUSPENSO");
        } else if (notaFinal < 7.0) {
            // ...
        } else {
            return ("MATRÍCULA DE HONOR");
        }
    }
}
```

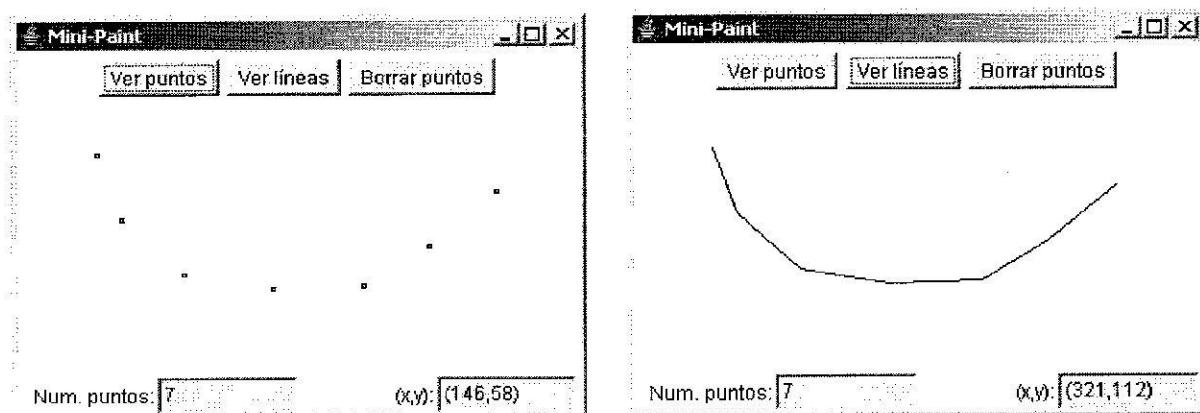
2. (3.0 puntos)

Escribir una aplicación gráfica en Java (AWT) que permita dibujar puntos en su zona de trabajo cada vez que el usuario haga "click" con el ratón. Estos puntos se almacenarán en un objeto de la clase Vector, es decir, este vector contendrá todas las coordenadas de los puntos en los que el usuario ha pulsado el ratón. Lo que se almacena en dicho vector serán objetos de la clase Point.

Aspecto gráfico:

La aplicación constará de:

- Una **barra de herramientas** en la zona superior con tres botones:
 - Ver puntos: desde que se pulse, los puntos se visualizarán como pequeños rectángulos: `drawRect(x,y,ancho,alto)`.
 - Ver líneas: desde que se pulse, se visualizarán las líneas que unen los puntos almacenados: `drawLine(x1,y1,x2,y2)`.
 - Borrar puntos: borra todos los puntos, esto es, todos los elementos del vector.
- Una **barra de estado** en la zona inferior, donde se nos muestre en todo momento el número de puntos que tenemos en el vector, y las coordenadas (x,y) del ratón.



Consejos:

- Para crear los objetos Point, existe un constructor: `Point (int x, int y)`.
- Para el Frame se puede utilizar como gestor de esquemas el "BorderLayout", poniendo un panel en el norte para la barra de herramientas, y otro panel en el sur para la barra de estado. A su vez, cada uno de estos paneles tendrá su propio gestor de esquemas.
- Para la barra de estado, se pueden utilizar objetos Label o TextField.

Solución

```
/*
 * =====
 * Titulo : Aplicación gráfica
 * Fichero: MiniPaint.java
 * =====
 */
import java.awt.*;
import java.awt.event.*;
import java.util.*; // Para utilizar la clase Vector
public class MiniPaint {
    // Función principal
    public static void main(String[] args) {
        // Creamos un objeto frame y lo mostramos (como siempre)
        // ...
    }
}

class FrameMiniPaint extends Frame
    implements ActionListener, MouseListener, MouseMotionListener {
    Panel ...;
    Button ...;
    TextField ...;
    boolean verPuntos;

    /** Vector para lista de puntos */
    Vector puntos;

    FrameMiniPaint (String titulo) {
        super(titulo);

        // 1. Establecer un gestor de esquemas: BorderLayout.
        // 2. Creamos el panel norte
        // 2a. Agregar el panel al norte del frame
        // 2b. Establecemos un gestor de esquemas al panel norte
        // 2c. Agregamos los 3 botones y registramos sus eventos
        // 2d. Creamos el panel sur
        // 2a. Agregar el panel al sur del frame
        // 2b. Establecemos un gestor de esquemas al panel sur
        // 2c. Agregamos los Label y TextField al panel sur

        // 3. Vreamos el vector (Lista de puntos)
        puntos = new Vector();

        // 4. Atender eventos de la ventana para poder cerrara (windowClosing)
        // 5. Registrarmos el gestor de eventos del ratón, tanto de movimiento
        //     (MouseMotionListener), como de click (MouseListener).
    }

    // Eventos de botones
    public void actionPerformed (ActionEvent e) {
        if (e.getSource() == buttonPuntos) {
            verPuntos = true;
            repaint();
        } else if (e.getSource() == buttonLineas) {
            verPuntos = false;
            repaint();
        } else if (e.getSource() == buttonBorrar) {
            puntos.clear();
            repaint();
        }
    }
}
```

```

// Acciones de dibujo
public void paint (Graphics g) {
    int i;
    Point p, p2;
    if (verPuntos) {
        for (i=0; i<puntos.size(); i++) {
            p = (Point) puntos.get(i);
            g.drawRect (p.x, p.y, 2, 2);
        }
    } else {
        for (i=0; i<puntos.size()-1; i++) {
            p = (Point) puntos.get(i);
            p2 = (Point) puntos.get(i+1);
            g.drawLine (p.x, p.y, p2.x, p2.y);
        }
    }
    // Mostramos el número de puntos (size) en el textfield inferior (setText)
}
// Eventos del ratón: pulsaciones botones
public void mouseClicked(MouseEvent e) {
    // 1. Obtener las coordenadas del ratón con e.getX() y e.getY()
    // 2. Crear un nuevo punto Point, añadirlo al vector
    // 3. Redibujar
}
// Eventos del ratón: movimiento
public void mouseMoved(MouseEvent e) {
    // 1. Obtener las coordenadas del ratón con e.getX() y e.getY()
    // 2. Establecer las coordenadas en el textfield inferior como texto.
    // 3. Redibujar
}
public void mouseDragged(MouseEvent e) {}
}
=====
```



Programación Avanzada

Convocatoria Ordinaria de Diciembre de 2006
Ingeniería Técnica de Telecomunicación (ITT-ST / ITT-SE)

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

1. ¿Por qué crees que los diseñadores del lenguaje Java lo han hecho tan parecido al lenguaje C++?
a) Para poder utilizar los mismos compiladores, indicando mediante algún parámetro el lenguaje concreto a utilizar.
* b) Java y C no tienen nada que ver, pero lo han hecho así para facilitar su aprendizaje.
c) Los diseñadores de Sun querían hacer una ampliación al lenguaje C, pero manteniendo la compatibilidad con los programas ya desarrollados.
2. ¿Cómo se llama la utilidad del JDK que nos permite generar páginas Web a partir de unos comentarios especiales insertados en el código fuente?
a) docuweb
* b) javadoc
c) javah
3. Cuando queremos expresar una expresión aritmética con valores constantes reales, por ejemplo: 3.14, ¿tenemos algún mecanismo de distinguir de forma explícita cuando queremos utilizar "float" o "double"?
a) Con la técnica del casting, poniendo (float) o (double) justo delante de la constante, dependiendo de su tipo.
b) Poniendo la letra F justo detrás para indicar que es un "float", y la letra D para indicar que es "double"; si no la especificamos, el compilador asume que es "double" (por defecto).
* c) Las dos anteriores
4. En programación estructurada (lenguaje C), una estructura o registro "struct" equivale (mas o menos) en POO a:
a) un atributo
* b) una clase
c) un método
5. ¿Cuál es la clase raíz de la jerarquía de clases de Java?
a) Main
* b) Object
c) Java
6. Para crear un array de enteros (int) de dos dimensiones, ¿cuál de las siguientes sentencias es incorrecta sintácticamente?
* a) int n = new int[10][10];
b) int n[][] = new int[10][10];
c) int n[][] = { { 1, 2 }, { 3, 4 } };
7. Una clase puede utilizar (implementar):
a) sólo una interfaz, pues en Java no se soporta la herencia múltiple.
* b) todas las interfaces que necesite.
c) tanto interfaces como objetos virtuales.
8. En una aplicación en modo gráfico, ¿qué es un "contenedor"?
* a) un componente más, pero que puede contener a otros componentes
b) es una librería de componentes
c) donde se depositan los objetos que ya no se utilizan, resultado del proceso de "recolección de basura".
9. ¿Para qué se utiliza una clase adaptadora?
a) es obligatoria para poder utilizar cualquiera de las interfaces "Listener"
* b) nos evita tener que definir todos los métodos que incorpora una interfaz de tipo "Listener"
c) las dos anteriores
10. ¿Es necesario compilar un programa java para ejecutarlo como applet?
a) Depende del navegador del usuario
b) No, porque los navegadores son capaces de interpretar código fuente de java insertado en su código
* c) Si, como cualquier otro programa escrito en java

Teoría - Cuestionario (0.2 cada una, total 3.0 puntos)

1. Define los conceptos de "compilador" y "enlazador" (linker). ¿Son necesarios en Java?

El compilador si. El linker no. Ver apuntes.

2. Define el concepto de "alcance" o "visibilidad" de un identificador (variable, constante u objeto) dentro de una clase.

Ver apuntes.

3. ¿Cuáles son las condiciones para que se realice una conversión automática de una variable de un tipo a otro tipo distinto? Pon un ejemplo.

Ver apuntes.

4. Cuando definimos una constante dentro de una clase, ¿por qué es conveniente especificar el modificador "static"? Pon un ejemplo de definición de una constante.

Para evitar que cada uno de los objetos de esta clase guarden una copia de la constante. Ver apuntes.

5. Define el concepto de método.

Ver apuntes.

6. Define el concepto de polimorfismo. Pon algún ejemplo.

Ver apuntes.

7. ¿Qué modificador de clase se utiliza para indicar que una clase X es abstracta?

Se utiliza el modificador "abstract": abstract class X { ... }

8. Los constructores y finalizadores no se heredan. Si quisieramos disponer en la subclase de los mismos constructores que en la superclase, ¿cómo lo haríamos?

Ver apuntes.

9. ¿Qué significa que un parámetro se pase a un método por valor o por referencia?

Ver apuntes.

10. ¿Para qué sirve la cláusula "throws" en la gestión de excepciones?

Ver apuntes.

11. ¿Es posible el uso de varios gestores de esquemas en un único contenedor?. Razona la respuesta.

Si. Ver apuntes.

12. Enumera las clases de que dispone Java para crear y gestionar menús desplegables en aplicaciones gráficas AWT, dando una mínima descripción de para qué se utiliza cada una de ellas (al menos 3). Ojo: no se pregunta por componentes o controles AWT.

Ver apuntes.

13. ¿Qué significa que un cuadro de diálogo se muestre en modo "modal" o "no modal"?

Ver apuntes.

14. Los métodos paint() y update() reciben un argumento de la clase Graphics. ¿Para qué sirve este objeto? Pon un ejemplo.

Ver apuntes.

15. En Java podemos almacenar y leer objetos completos a un fichero. ¿Cómo se llama esta técnica?. ¿Qué clases se utilizan en ambos casos?

Serilación. Ver apuntes.

Programación (6.0 puntos)

1. (1.0 puntos)

Escribir una clase en Java llamada "**Matriz**", que sirva para representar datos de tipo matriz. Esta clase, la cual deberá tener los datos y operaciones que se definen a continuación, se utilizará en los ejercicios siguientes; caso de no realizar este ejercicio, el alumno deberá suponer que esta clase existe, y con esta funcionalidad.

Datos (todos públicos)		
Dato	Tipo	Descripción
m	Array de enteros de dos dimensiones	Valores de los elementos de la matriz
filas	Entero	Número de filas de la matriz
cols	Entero	Número de columnas de la matriz

Constructores	
Descripción	Recibe
Crear una matriz con las filas y columnas especificadas. Además, almacena los valores recibidos en las variables filas y cols.	El número de filas y el número de columnas

Métodos		Recibe	Retorna
Método	Descripción		
generar	Generar de forma aleatoria todos los valores de la matriz, comprendidos entre 0 y el valor máximo recibido.	El valor máximo.	Nada
traspuesta	Crear y obtener la matriz traspuesta de la matriz recibida.	Nada	La matriz traspuesta.
toString	Construye una cadena de texto con los datos de la matriz, apareciendo cada fila de la matriz en una línea diferente.	Nada	La cadena construida

- Para **generar números aleatorios** se puede utilizar la función estática `random()`, de la clase `Math`. Esta función retorna un valor `double` comprendido entre 0.0 y 1.0 ($0.0 \leq n < 1.0$), de forma aleatoria. Por tanto, deberemos multiplicarlo por el límite superior del rango de números que se desea producir y sumar un valor para establecer el límite inferior. En este caso particular, el valor máximo es el que recibe la función `generar (max)`, y como el valor inferior es 0 no es necesario hacer nada más. Así obtendremos un valor entre 0 y `max-1`, pero como lo que nos interesa es entre 0 y `max` tendremos que hacer una ligera modificación. Finalmente, como lo que hemos obtenido es un `double` y necesitamos obtener un `int` para acceder al elemento concreto del vector, deberemos convertirlo de forma explícita a `int` (`casting`).

2. (1.0 puntos)

Escribir un programa en Java llamado "**MatrizTest**" que nos permita probar la clase "Matriz" del ejercicio anterior (se debe utilizar obligatoriamente dicha clase, aunque no se haya realizado el ejercicio anterior). El programa generará una matriz de forma aleatoria, y calculará su matriz traspuesta, imprimiendo después ambas matrices. El programa recibirá 3 argumentos desde línea de órdenes: 1) número de filas, 2) número de columnas de la matriz a generar, y 3) el valor máximo (max) que podrán tener los valores de la matriz (0..max).

Se deberá comprobar el número de argumentos recibido, y que éstos son válidos. Puesto que los argumentos recibidos serán cadenas de texto, necesitaremos convertirlos a valores numéricos enteros (`Integer.parseInt`). Asimismo, se deberá comprobar que los tres valores son mayor que 0 para poder continuar. En caso de no cumplir alguna de estas condiciones, el programa deberá mostrar un mensaje de error y terminar. Algunos ejemplos de uso son:

```
C:\>java MatrizTest  
Uso: java MatrizTest filas cols max  
  
C:\>java MatrizTest 2 3 A  
Argumentos incorrectos. Sólo valores numéricos > 0.  
  
C:\>java MatrizTest 2 3 9  
Matriz (2x3):  
4 8 2  
0 2 9  
Traspuesta (3x2):
```

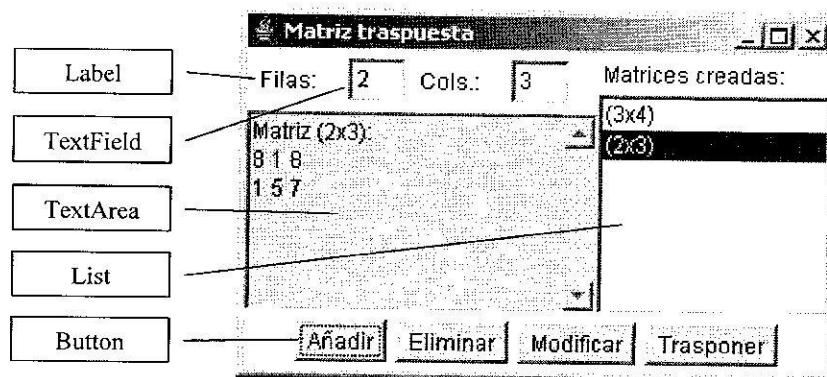
4	0
8	2
2	9

3. (2.5 puntos)

Escribir un programa gráfico en Java (AWT) llamado "**MatrizTestApp**" con el aspecto gráfico que se muestra a continuación, y la funcionalidad descrita. En resumen, se trata de crear matrices (objetos de la clase Matriz del primer ejercicio) y almacenarlas en un Vector, mostrando siempre las matrices creadas en una lista. De esta lista podremos eliminar cualquiera de las matrices creadas, modificarlas, o añadir la traspuesta de la que tengamos seleccionada. En la izquierda se muestra siempre la matriz seleccionada en la lista en forma de cadena (`toString`).

Aspecto gráfico:

- El TextArea se ha deshabilitado para no poder escribir en él (setEnabled).
- El programa debe de **cerrarse** al pulsar el botón "X" de la ventana.
- Se valorará el uso de los **gestores de esquemas** más adecuados.

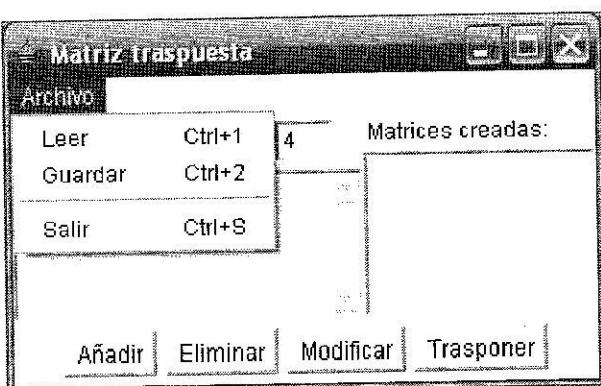


Funcionalidad del programa:

- Al pulsar [Añadir]: se creará un objeto de la clase Matriz con el número de filas y columnas especificadas (new), se asignarán de forma aleatoria los valores de la matriz, se añadirá nuevo el objeto al Vector de matrices creadas (add), se añadirá también a la lista de la derecha (add) una cadena con sus dimensiones (filas x cols), y, finalmente, se mostrará la matriz en la parte izquierda (setText y toString).
- Al pulsar [Eliminar], se eliminará la matriz seleccionada en la lista. Será necesario saber el índice del elemento de la lista seleccionado (getSelectedIndex) y eliminarlo, tanto del vector (remove), como de la lista (remove).
- Al pulsar [Modificar] nos aparecerá otra ventana (Dialog) que nos permitirá modificar cualquier valor de la matriz seleccionada. Al igual que en el caso anterior, es necesario saber el índice del elemento de la lista seleccionado, para acceder con dicho índice al vector y obtener la matriz correspondiente (get). Dicha matriz se pasará al cuadro de diálogo para que se pueda modificar, el cual aparecerá con tantas casillas (TextField) como elementos tenga la matriz a modificar. Las casillas aparecen ya llenas con los valores correspondientes (setText). En el ejemplo se muestra cómo aparecería la ventana para una matriz (3x4). Si el usuario pulsa [Aceptar] se deberá extraer cada uno de los valores de estas casillas (getText), asignarlo al elemento correspondiente de la matriz y cerrar el diálogo (dispose); si alguno de los valores tecleados fuese incorrecto (una letra, etc.), se ignorará dicho valor, aunque se continuará con el resto de valores. Por otro lado, si se pulsa [Cancelar], tan sólo cerraremos el diálogo, dejando la matriz intacta.
- Al pulsar [Trasponer] se añadirá una nueva matriz que será la traspuesta de la matriz seleccionada. Es decir, se hará lo mismo que en el caso de [Añadir], salvo que las dimensiones de la matriz dependen ahora de la matriz que ya tengamos seleccionada, y los datos no se generan de forma aleatoria sino que serán los traspuestos. Por tanto, también necesitamos obtener la matriz seleccionada, de la misma forma que en los casos anteriores.
- Al seleccionar una matriz de la lista de la derecha (con el ratón o teclas del cursor), mostraremos en la parte izquierda la matriz correspondiente en forma de texto (toString). Para ello habrá que capturar un evento (interface ItemListener), obtener la matriz seleccionada (de la misma forma que se ha explicado anteriormente), y asignar (setText) el texto de la matriz a la zona de la izquierda.

4. (1.5 puntos)

Realizar las modificaciones necesarias a la aplicación gráfica del ejercicio anterior (o a cualquier clase que sea necesario) para incorporar una barra de menús desplegable, con las opciones necesarias para: 1) guardar la lista de matrices (objeto de la clase Vector) creadas en un archivo, 2) leerlas de dicho archivo, y 3) salir del programa. Se valorará el uso de teclas rápidas o short-cuts.



Para **guardar el Vector**, se pedirá al usuario el nombre del fichero a guardar mediante el diálogo predefinido en Java para ello (clase `FileDialog`). En caso de no cancelar, se procederá finalmente a grabar el objeto Vector en el archivo especificado.

De la misma forma se procederá para **leer el Vector**, utilizando en cada caso las clases correspondientes. En este caso, además, tendremos que actualizar la lista de la derecha con las matrices leídas, seleccionando finalmente cualquiera de ellas.

Recuerda que `FileDialog` varía ligeramente cuando se trata de grabar o de leer.

Solución:

```
// =====
//   Titulo :    Matriz
//   Fichero:    Matriz.java
// =====

import java.io.*;      // Serializable

public class Matriz implements Serializable {
    public int m[][];
    public int filas, cols;

    Matriz (int filas, int cols) {
        m = new int[filas] [cols];
        this.filas = filas;
        this.cols = cols;
    }

    public void generar (int max) {
        for (int i=0; i<filas; i++) {
            for (int j=0; j<cols; j++) {
                m[i][j] = (int)(Math.random()*max)+1;
            }
        }
    }

    public Matriz traspuesta() {
        Matriz t = new Matriz(cols,filas);
        for (int i=0; i<filas; i++) {
            for (int j=0; j<cols; j++) {
                t.m[j][i] = m[i][j];
            }
        }
        return (t);
    }

    public String toString () {
        String s = "Matriz (" + filas + "x" + cols + "):\n";
        for (int i=0; i<filas; i++) {
            for (int j=0; j<cols; j++) {
                s = s + m[i][j] + " ";
            }
            s = s + "\n";
        }
        return (s);
    }
}

// =====
```

```

// =====
//   Titulo :    MatrizTest
//   Fichero:    MatrizTest.java
// =====

public class MatrizTest {

    // Función principal
    public static void main (String args[]) {
        int filas=0, cols=0, max=0;
        Matriz m, t;

        if (args.length != 3) {
            System.err.println("Error");
            System.exit(1);
        }
        try {
            filas = Integer.parseInt(args[0]);
            cols = Integer.parseInt(args[1]);
            max = Integer.parseInt(args[2]);
        } catch (NumberFormatException e) {
            System.err.println("Error");
            System.exit(1);
        }
        if ((filas<=0) || (cols<=0) || (max<=0)) {
            System.err.println("Error");
            System.exit(1);
        }
        // Generamos aleatoriamente la matriz m
        m = new Matriz (filas,cols);
        m.generar(max);
        System.out.println("Matriz generada ("+filas+"x"+cols+"): \n"+m);
        // Construimos la matriz traspuesta
        t = m.traspuesta();
        System.out.println("Matriz traspuesta ("+cols+"x"+filas+"): \n"+t);
    }
}

// =====

```

```

// =====
//   Titulo : MatrizTestAppLoadSave
//   Fichero: MatrizTestAppLoadSave.java
// =====

import java.awt.*;
import java.awt.event.*;
import java.util.Vector; // Importamos clase Vector
import java.io.*;

public class MatrizTestAppLoadSave {
    public static void main (String sArgs[]) {
        FrameMatrizTestLoadSave f;

        f = new FrameMatrizTestLoadSave ("Matriz traspuesta");
        f.setSize (320,200);
        f.show();
    }
}

// ActionListener: para las opciones de menús y submenús
// ItemListener: para detectar la selección de la matriz en la lista
class FrameMatrizTestLoadSave extends Frame implements ActionListener, ItemListener
{
    Panel      pCentro, pCentroNorte, pCentroCentro, pEste, pSur;
    Label      lblFilas, lblCols, lblMatrices;
    TextField  txtFilas, txtCols;
    TextArea   txtMatriz;
    List       lista;
    Button     bAñadir, bEliminar, bModificar, bTrasponer;

    Vector matrices;

    // --- Ejercicio 4 - Barra de menu (inicio) -----
    MenuBar menubar;
    Menu menu;
    MenuItem menuitem1, menuitem2, itemsalir;
    MenuShortcut shortcut;
    // --- Ejercicio 4 - Barra de menu (fin) -----

    FrameMatrizTestLoadSave (String titulo) {
        super(titulo);
        setLayout (new BorderLayout());

        // Panel centro
        pCentro = new Panel();
        pCentro.setLayout (new BorderLayout());
        pCentroNorte = new Panel();
        pCentroNorte.setLayout (new FlowLayout(FlowLayout.LEFT));
        pCentroCentro = new Panel();
        pCentroCentro.setLayout (new BorderLayout());
        pCentro.add (pCentroNorte, BorderLayout.NORTH);
        pCentro.add (pCentroCentro, BorderLayout.CENTER);
        lblFilas = new Label("Filas:");
        lblCols = new Label("Cols.:");
        txtFilas = new TextField("3");
        txtCols = new TextField("4");
        txtMatriz = new TextArea("",10,10,TextArea.SCROLLBARS_BOTH);
        txtMatriz.setEditable (false);
        pCentroNorte.add (lblFilas);
        pCentroNorte.add (txtFilas);
        pCentroNorte.add (lblCols);
        pCentroNorte.add (txtCols);
        pCentroCentro.add (txtMatriz);
    }
}

```

```

// Panel este
pEste = new JPanel();
pEste.setLayout (new BorderLayout());
lblMatrices = new JLabel("Matrices creadas:");
lista = new List();
lista.addItemListener(this);
pEste.add (lblMatrices, BorderLayout.NORTH);
pEste.add (lista, BorderLayout.CENTER);

// Panel sur
pSur = new JPanel();
pSur.setLayout (new FlowLayout());
bAñadir = new JButton("Añadir");
bEliminar = new JButton("Eliminar");
bModificar = new JButton("Modificar");
bTrasponer = new JButton("Trasponer");
bAñadir.addActionListener (this);
bEliminar.addActionListener (this);
bModificar.addActionListener (this);
bTrasponer.addActionListener (this);
pSur.add (bAñadir);
pSur.add (bEliminar);
pSur.add (bModificar);
pSur.add (bTrasponer);

// Añadimos los paneles al Frame
add (pCentro, BorderLayout.CENTER);
add (pEste , BorderLayout.EAST);
add (pSur , BorderLayout.SOUTH);

// Eventos de la ventana
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0); }});

// Creamos vector de matrices
matrices = new Vector();

// --- Ejercicio 4 - Barra de menu (inicio) -----
menu = new Menu("Archivo");
shortcut = new MenuShortcut(KeyEvent.VK_1); // CTRL+1
menuitem1 = new MenuItem("Leer", shortcut);
menu.add(menuitem1);
menuitem1.addActionListener(this);
//
shortcut = new MenuShortcut(KeyEvent.VK_2); // CTRL+2
menuitem2 = new MenuItem("Guardar", shortcut);
menu.add(menuitem2);
menuitem2.addActionListener(this);
//
menu.addSeparator();
//
shortcut = new MenuShortcut(KeyEvent.VK_S); // CTRL+S
itemsalir = new MenuItem("Salir", shortcut);
menu.add(itemsalir);
itemsalir.addActionListener(this);
//
menubar = newMenuBar();
menubar.add(menu);
setMenuBar(menubar);
// --- Ejercicio 4 - Barra de menu (fin) -----
}

```

```

// Al pulsar cualquier botón ...
public void actionPerformed (ActionEvent e) {
    Matriz m, t;
    int filas, cols, index;

    if (e.getSource() == bAñadir) {
        try {
            filas = Integer.parseInt(txtFilas.getText());
            cols = Integer.parseInt(txtCols.getText());
        } catch (NumberFormatException exc) {
            return;
        }
        m = new Matriz(filas, cols);
        m.generar(9);
        añadirMatriz(m);
    } else if (e.getSource() == bEliminar) {
        index = lista.getSelectedIndex();
        lista.remove(index);
        matrices.remove(index);
        txtMatriz.setText ("Matriz eliminada");
    } else if (e.getSource() == bModificar) {
        index = lista.getSelectedIndex();
        m = (Matriz) matrices.get(index);
        DlgMatrizLoadSave dlg = new DlgMatrizLoadSave (this,m);
        dlg.show();
        txtMatriz.setText (m.toString());
    } else if (e.getSource() == bTrasponer) {
        index = lista.getSelectedIndex();
        m = (Matriz) matrices.get(index);
        t = m.traspuesta();
        añadirMatriz (t);
    }
    // --- Ejercicio 4 - Barra de menu (inicio) -----
    if (e.getSource() == menuitem1) {
        leerMatrices();
    } else if (e.getSource() == menuitem2) {
        grabarMatrices();
    } else if (e.getSource() == itemsalir) {
        System.exit(0);
    }
    // --- Ejercicio 4 - Barra de menu (fin) -----
}

public void itemStateChanged (ItemEvent e) {
    int index;
    Matriz m;

    if (e.getSource() == lista) {
        index = lista.getSelectedIndex();
        m = (Matriz) matrices.get(index);
        txtMatriz.setText (m.toString());
    }
}

private void añadirMatriz (Matriz m) {
    matrices.add (m);
    lista.add ("(" +m.filas+"x"+m.cols+ ")");
    lista.select ( lista.getItemCount()-1 );
    txtMatriz.setText (m.toString());
}

```

```

// --- Ejercicio 4 - Barra de menu (inicio) -----
private void leerMatrices() {
    String fichero;
    FileDialog filedialog;

    filedialog = new FileDialog(this, "Seleccione archivo a leer", FileDialog.LOAD);
    //filedialog.setMode (FileDialog.LOAD);
    filedialog.show();
    fichero = filedialog.getFile();
    if (fichero != null) {
        try {
            FileInputStream fis = new FileInputStream(fichero);
            ObjectInputStream ois = new ObjectInputStream (fis);
            matrices = (Vector) ois.readObject();
            fis.close();
            actualizarLista();
        } catch (FileNotFoundException exc) {
        } catch (IOException exc) {
        } catch (ClassNotFoundException exc) {
        }
    }
}

private void actualizarLista() {
    Matriz m=null;

    for (int i=0; i<matrices.size(); i++) {
        m = (Matriz) matrices.get(i);
        lista.add ("(" +m.filas+"x"+m.cols+ ")");
    }
    // Seleccionamos el ultimo elemento
    if (lista.getItemCount() > 0) {
        lista.select ( lista.getItemCount()-1 );
        txtMatriz.setText (m.toString());
    }
}

private void grabarMatrices () {
    String fichero;
    FileDialog filedialog;

    filedialog = new FileDialog(this, "Seleccione archivo grabar", FileDialog.SAVE);
    //filedialog.setMode (FileDialog.SAVE);
    filedialog.show();
    fichero = filedialog.getFile();
    if (fichero != null) {
        try {
            FileOutputStream fos = new FileOutputStream(fichero);
            ObjectOutputStream oos = new ObjectOutputStream (fos);
            oos.writeObject(matrices); // Matriz debe ser Serializable
            fos.close();
        } catch (FileNotFoundException exc) {
        } catch (IOException exc) {
        } //} catch (NotSerializableException exc) {
    }
}
// --- Ejercicio 4 - Barra de menu (fin) -----
}

```

```

class DlgMatrizLoadSave extends Dialog implements ActionListener {
    Panel pCentro, pSur;
    TextField casilla[][];
    Button ok, cancel;
    Matriz m;

    DlgMatrizLoadSave (Frame f, Matriz m) {
        super (f, "Editar matriz", true);
        setSize (150,130);
        this.m = m;
        setLayout (new BorderLayout ());

        pCentro = new Panel ();
        pCentro.setLayout (new GridLayout(m.filas,m.cols));
        casilla = new TextField[m.filas][m.cols];
        for (int i=0; i<m.filas; i++) {
            for (int j=0; j<m.cols; j++) {
                casilla[i][j] = new TextField(""+m.m[i][j]);
                pCentro.add (casilla[i][j]);
            }
        }

        pSur = new Panel ();
        pSur.setLayout (new FlowLayout ());
        ok = new Button("Aceptar");
        cancel = new Button("Cancelar");
        ok.addActionListener (this);
        cancel.addActionListener (this);
        pSur.add (ok);
        pSur.add (cancel);

        add (pCentro, BorderLayout.CENTER);
        add (pSur,     BorderLayout.SOUTH);

        // Eventos de la ventana
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose(); }});
    }

    public void actionPerformed (ActionEvent e) {
        int v;

        if (e.getSource()==ok) {
            // Actualizamos la matriz
            for (int i=0; i<m.filas; i++) {
                for (int j=0; j<m.cols; j++) {
                    try {
                        v = Integer.parseInt(casilla[i][j].getText());
                        m.m[i][j] = v;
                    } catch (NumberFormatException exc) {
                    }
                }
            }
            dispose();
        }
    }
}
// =====

```

Programación Avanzada

Convocatoria Ordinaria de Septiembre de 2010
Ingeniería Técnica de Telecomunicación (ITT-ST / ITT-SE)

Teoría – Test

1 punto: 0.1 cada una, cada dos mal restan una bien

Teoría - Cuestionario

3 puntos: 0.2 cada pregunta

Programación

6 puntos

1. Ejercicio en modo texto

2.0 puntos

En matemáticas, la sucesión de Fibonacci es una sucesión infinita de números naturales en donde cada elemento se obtiene de la suma de los dos anteriores, siendo los dos primeros 0 y 1. Escribir un programa en Java que reciba un valor natural (entero mayor o igual que 0), y calcule el valor de esta sucesión para dicho valor. La función principal (main) recibirá un único argumento, y deberá invocar a dos funciones que realicen este mismo cálculo pero de dos formas distintas: (a) Recursiva y (b) Iterativa (bucles).

Sucesión de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

Como ayuda se proporciona la siguiente definición de la sucesión de Fibonacci de forma recursiva:

$$f(n) = \begin{cases} 0 & \text{si } n=0 \\ 1 & \text{si } n=1 \\ f(n-1) + f(n-2) & \text{si } n>1 \end{cases}$$

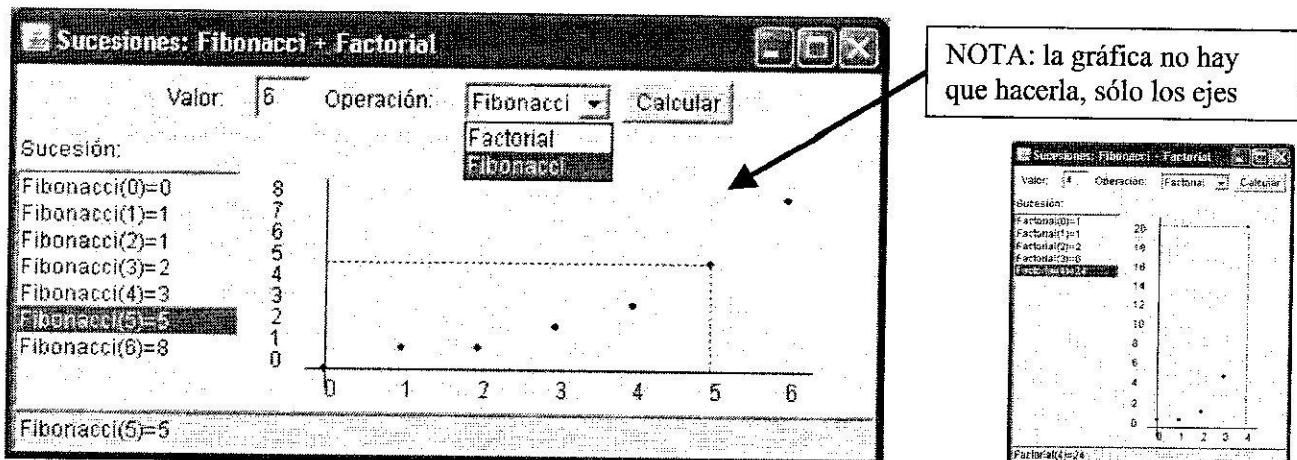
¿Cuántas llamadas recursivas se hacen cuando queremos evaluar $f(3)$, incluyendo la llamada a $f(3)$?

$f(3) = f(2) + f(1) = [f(1) + f(0)] + f(1) = [1+0] + 1 = 2$ Total = 4 llamadas

2. Ejercicio en modo gráfico

4.0 puntos

Escribir un programa gráfico en Java (AWT) llamado "SucesionesApp" para calcular sucesiones de valores para la función de Fibonacci y la función Factorial.



Funcionalidad del programa:

El usuario tecleará un valor en el campo (TextField) "Valor", seleccionará la operación a realizar (Factorial o Fibonacci) de la lista desplegable (Choice), y al pulsar el botón "Calcular" (Button) se llenará la lista desplazable (List) de la izquierda con todos los valores de esa función desde 0 hasta el valor introducido. Posteriormente, cuando el usuario seleccione un elemento de la lista, habrá que colocar dicho elemento en un texto (TextField) no editable (setEditable) o deshabilitado (setEnabled) en la barra de estado inferior. En cuanto a la gráfica de la derecha, tan solo habrá que dibujar las líneas de los ejes.

Implementación del programa (en el orden recomendado):

1. Interface gráfico:

- Hacer la ventana (Frame) con el interface gráfico mostrado. El programa deberá tener un **aspecto** lo más parecido posible al que se muestra en la figura. Se valorará el utilizar los **gestores de esquemas** más adecuados
- De la gráfica de la derecha, tan solo se dibujarán las **líneas de los ejes**, ajustados al tamaño actual de la ventana (getWidth() y getHeight()). Se dejará siempre un margen alrededor (p.e. 50 pixels): entre la lista de valores y el eje Y, entre el eje X y los bordes inferior y derecho de la ventana, etc. El eje X se dibujará en **color negro** y el eje Y en **color azul**. NOTA: no se pide hacer el resto de la gráfica: valores en los ejes, puntos, etc.

2. Eventos a capturar:

- Cerrar la ventana al pulsar el botón "X" de la misma.
- Que al pulsar el botón Calcular se llene la lista con los valores de la sucesión para la operación seleccionada. Se deberá implementar dos funciones para calcular el Factorial y el valor de Fibonacci, ya sea de forma recursiva o iterativa (una forma solamente). Si has realizado el ejercicio anterior no se puntúa la función para el cálculo de Fibonacci.
- Que al seleccionar un elemento de la lista aparezca ese mismo texto en la barra de estado.
- Que al cambiar el tamaño de la ventana se redibuje el contenido. En nuestro caso no se trata de redibujar la gráfica completa sino sólo las líneas de los ejes X e Y, ajustados al nuevo tamaño de la ventana; ver imagen de la derecha.

Solución:

```
// =====
//   Titulo :    Fibonacci
//   Fichero:    Fibonacci.java
// =====

package matematicas;

public class Fibonacci {

    public static void main (String args[]) {
        int n=0;      // Valor desde linea de ordenes
        long f1, f2;

        // Comprobamos el número de argumentos
        if (args.length != 1) {
            // Mostramos mensaje con la forma de uso
            System.out.println("Uso: Fibonacci n (con n>=0)");
            System.exit(1);
        }
        try {
            // Obtenemos el valor numérico del primer argumento
            n = Integer.parseInt(args[0]);
        } catch (NumberFormatException e) {
            System.err.println("Datos incorrectos");
            System.exit(1);
        }
        // Comprobamos que el valor sea natural (>=0)
        if (n<0) {
            System.out.println("El valor debe ser >= 0");
            System.exit(1);
        }
        // Imprimir la sucesión de fibonacci con el numero de iteraciones dado
        f1 = fibonacci_iterativa (n);
        System.out.println("Sucesion iterativa: " + f1);
        f2 = fibonacci_recursiva (n);
        System.out.println("Sucesion recursiva: " + f2);
    }

    static long fibonacci_iterativa (int n) {
        int i, j, t=0;

        //if (n==0) return (0);
        //if (n==1) return (1);
        i = 1;
        j = 0;
        for (k=1; k<=n; k++) {
            t = i + j;
            i = j;
            j = t;
        }
        return (t);
    }

    static long fibonacci_recursiva (int n) {
        if (n==0) return (0);
        if (n==1) return (1);
        return ( fibonacci_recursiva(n-1) + fibonacci_recursiva(n-2) );
    }
}
```

```

/*
=====
 Titulo: Fibonacci
 Fichero: FibonacciApp_v2.java
=====
 */

package matematicas;

import java.awt.*;
import java.awt.event.*;
import java.util.*;

public class FibonacciApp_v2 {
    /** Función principal: simplemente crea un objeto Frame */
    public static void main ( String Args[] ) {
        MiFrame f;

        f = new MiFrame();
        f.setSize ( 480, 400 );
        f.show();
    }
}

/* Clase que define nuestro objeto Frame (aplicacion)
   ActionListener: para los eventos de los botones */
class MiFrame extends Frame implements ActionListener, ItemListener, ComponentListener
{
    java.awt.List lstSucesion;
    Label     lblValor, lblOperacion, lblSucesion;
    TextField txtValor, txtBarraEstado;
    Choice    chOperacion;
    Button    calcular;
    Panel    p1, p2;

    long valores[];
    int n;

    MiFrame() {
        super("Sucesiones: Fibonacci + Factorial");
        // Gestor de esquemas
        this.setLayout(new BorderLayout());

        // Panel norte
        p1 = new Panel();
        this.add (p1, "North");
        p1.setLayout (new FlowLayout());
        lblValor = new Label("Valor:");
        txtValor = new TextField("6");
        lblOperacion = new Label("Operación:");
        chOperacion = new Choice();
        chOperacion.add ("Factorial");
        chOperacion.add ("Fibonacci");
        calcular = new Button("Calcular");
        p1.add (lblValor);
        p1.add (txtValor);
        p1.add (lblOperacion);
        p1.add (chOperacion);
        p1.add (calcular);

        // Panel oeste
        p2 = new Panel();
        this.add (p2, "West");
        p2.setLayout (new BorderLayout());
        lblSucesion = new Label("Sucesión:");

```

0.1

```

lstSucesion = new java.awt.List();
p2.add (lblSucesion, "North");
p2.add (lstSucesion, "Center");

/* Barra de estado (sur)
txtBarraEstado = new TextField("");
txtBarraEstado.setEditable(false);
add (txtBarraEstado, "South");

// Registraremos gestores de eventos
lstSucesion.addItemListener (this);
calcular.addActionListener (this);
this.addComponentListener (this);

// Gestor de eventos de la ventana
addWindowListener( new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
}

// Al pulsar los botones ...
public void actionPerformed (ActionEvent evt) {
    int op;
    String operacion;

    if (evt.getSource() == calcular) {
        try {
            n = Integer.parseInt(txtValor.getText());
        } catch (NumberFormatException e) {
            txtBarraEstado.setText ("Valor incorrecto");
            return;
        }
        // Comprobamos que sea > 0
        if (n <= 0) {
            txtBarraEstado.setText ("Valor debe ser > 0");
            return;
        }
        // Generamos la sucesion
        valores = new long[n+1];
        // Averiguamos la operacion
        op = chOperacion.getSelectedIndex();
        operacion = chOperacion.getSelectedItem();
        // Llenamos la lista que muestra valores de la sucesión
        lstSucesion.clear();
        txtBarraEstado.setText ("");
        for (int i=0; i<=n; i++) {
            switch (op) {
                case 0:      // Factorial
                    valores[i] = factorial_recursiva(i);
                    break;
                case 1:      // Fibonacci
                    valores[i] = fibonacci_recursiva(i);
                    break;
                default:;
            }
            lstSucesion.add (operacion + "(" + i + ")=" + valores[i]);
        }
        lstSucesion.select(0);
        repaint();
    }
}
}

```

```

// Al pulsar en la lista ...
public void itemStateChanged (ItemEvent evt) {
    // Actualizamos barra de estado
    txtBarraEstado.setText (lstSucesion.getSelectedItem());
    repaint();
}

// Al cambiar de tamaño la ventana ampliamos el gráfico
public void componentResized (ComponentEvent e) {
    repaint();
}

public void componentMoved      (ComponentEvent e) {}
public void componentHidden     (ComponentEvent e) {}
public void componentShown      (ComponentEvent e) {}

public long factorial_recursiva (int n) {
    if (n==0) return (1);
    if (n==1) return (1);
    return ( n * factorial_recursiva(n-1) );
}

private long fibonacci_recursiva (int n) {
    if (n==0) return (0);
    if (n==1) return (1);
    return ( fibonacci_recursiva(n-1) + fibonacci_recursiva(n-2) );
}

// Dibujar gráfica
public void paint (Graphics g) {
    int i, origenx, origeny;
    int altografico, anchografico, altoventana, anchoventana;
    int x, y, incrancho, incralto, incrvalor, numvalores;
    int seleccionado;
    long max, v;

    if (valores==null) return;
    anchoventana = this.getWidth();
    altoventana = this.getHeight();
    origenx = lstSucesion.getWidth() + 50;
    origeny = altoventana - 50;
    anchografico = anchoventana - origenx - 50;
    altografico = origeny - 100;
    g.setColor (Color.BLACK);

    // Eje X
    g.drawLine (origenx-10, origeny, origenx+anchografico+10, origeny);
    // Eje Y
    g.drawLine (origenx, origeny+10, origenx, origeny-altografico-10);
}

```

```

// Valores eje X
max = valores[n];
incrAncho = anchografico / n;
for (i=0; i<=n; i++) {
    g.drawString (""+i, origenx+(i*incrAncho), origeny+15);
}

// Valores eje Y
max = valores[n];
numvalores = Math.min(10, (int)max);
incrValor = (int) (max/numvalores);
incrAlto = (int) (altografico / numvalores);
v = 0;
for (i=0; i<=numvalores; i++) {
    g.drawString (""+v, origenx-30, origeny-(i*incrAlto));
    v += incrValor;
}

// Valores
seleccionado = lstSucesion.getSelectedIndex();
for (i=0; i<=n; i++) {
    x = origenx+(i*incrAncho);
    y = origeny- (int) (valores[i]*altografico/max);
    if (i==seleccionado) {
        g.setColor (Color.RED);
    } else {
        g.setColor (Color.BLACK);
    }
    g.fillOval (x-2, y-2, 4, 4);
    if (i==seleccionado) {
        Graphics2D g2d = (Graphics2D)g;
        Stroke stroke = new BasicStroke (1f, BasicStroke.CAP_ROUND,
            BasicStroke.JOIN_ROUND, 1f, new float[] {2f}, 0f);
        g2d.setStroke (stroke);
        g2d.drawLine (origenx, y, x, y);
        g2d.drawLine (x, origeny, x, y);
    }
}
}

// -----

```



Programación Avanzada

Convocatoria Ordinaria de Febrero de 2007

Ingeniería Técnica de Telecomunicación (ITT-ST)

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

1. Un programa escrito en lenguaje Java es multiplataforma; esto quiere decir que:
 - a) una vez compilado, se podrá ejecutar directamente en todas las plataformas existentes
 - b) sólo necesitaremos compilar el programa para cada plataforma concreta, pero sin modificar el código fuente.
 - * c) una vez compilado, se podrá ejecutar en otras plataformas para las que exista un intérprete especial de Java.
2. Para compilar un programa Java llamado "Hola.java" desde línea de órdenes escribiremos:
 - a) java Hola
 - * b) javac Hola.java
 - c) javac Hola
3. Java tiene un repertorio de operadores tan extenso como el lenguaje C. Estos se pueden agrupar en:
 - a) Aritméticos y sentencias de control de flujo.
 - * b) Asignación, aritméticos, de bits, lógicos y comparadores.
 - c) Aritmético-lógicos y de objetos.
4. En Java, el mecanismo llamado "recolección de basura" o "garbage collection" consiste en:
 - * a) limpieza de la memoria utilizada por los objetos, para eliminar los objetos que ya no se utilicen.
 - b) compactación de cada uno de los objetos para que ocupen menos memoria, mediante ZIP, ARJ, ...
 - c) es una llamada al sistema operativo para que aborde las tareas menos utilizadas pero que consumen algo de tiempo del procesador (CPU).
5. ¿Cómo se especifica en Java que una clase X es abstracta?
 - a) virtual X { ... }
 - * b) abstract X { ... }
 - c) las dos anteriores
6. ¿Es obligatorio definir algún constructor dentro de una clase?
 - a) Si, pues de lo contrario nunca podremos crear objetos
 - * b) No, ya que siempre podremos crear objetos al menos con el constructor por defecto
 - c) Depende de si la clase hereda algún constructor de alguna superclase
7. ¿Cuál de las siguientes sentencias es la única forma correcta de definir un array de dos dimensiones?
 - a) int n[10,10];
 - b) int n[10][10];
 - * c) int n[][] = { { 1, 2 }, { 3, 4 } };
8. Además de la jerarquía de clases en java tenemos una jerarquía de paquetes (packages). ¿cuál es el paquete raíz de dicha jerarquía?
 - a) El paquete raíz es "Object"
 - * b) El paquete raíz es "java"
 - c) No existe tal jerarquía de paquetes ya que los paquetes están totalmente aislados unos de otros.
9. Cuando una clase utiliza (implementa) una interfaz:
 - * a) debe definir todos los métodos que se declaran en la interfaz, o bien, ser declarada como abstracta.
 - b) la clase puede definir solamente los métodos que le interesan, no es necesario definir todos los métodos que se declaran en la interfaz, y ya podemos crear objetos de la clase.
 - c) la clase no necesita definir ninguno de los métodos, pues los "hereda" de la interfaz; tan solo sobreescribir los métodos que necesiten cambiar de comportamiento.
10. Para registrar un manejador de eventos creado desde los interfaces "Listener" utilizamos unos métodos "add...Listener" (addActionListener, addMouseListener, etc.). Cuando utilizamos clases adaptadoras:
 - * a) registraremos el manejador de eventos del mismo modo que cuando utilizamos los interfaces "Listener".
 - b) ya no es necesario registrar el manejador de eventos; ya se encargan de ellos las clases adaptadoras.
 - c) utilizaremos unos métodos equivalentes: "add...Adapter" (addActionAdapter, addMouseAdapter, etc.)

Teoría - Cuestionario (0.2 cada una, total 4.0 puntos)

1. Java es un lenguaje "Orientado a Objetos". ¿Sabrías explicar qué significa esto?

Ver apuntes.

2. Enumera las diferencias que conozcas entre variable local y variable de clase. ¿Es posible tener una variable local y una de clase con el mismo nombre?

Ver apuntes.

3. ¿Es posible convertir en Java una variable de tipo "boolean" a algún dato numérico de forma automática?. ¿Y empleando la técnica del "casting"? ¿Puede convertirse a una cadena de caracteres String?. Razona las respuestas.

No. No. Si. Ver apuntes.

4. ¿Qué ventajas nos aporta el uso de constantes en cualquier programa?. ¿Cuál es la sintaxis para declarar una constante?.

Evitar el uso de "números mágicos. Ver apuntes.

5. Enumera y pon un ejemplo de uso de sentencias de control de flujo de las que dispone el lenguaje Java (con 3 es suficiente). OJO: no se pregunta sobre el concepto de "flujo" o stream.

if-else, for, while, switch, break, continue. Ver apuntes.

6. Java dispone de una sentencia de bucle "while (condición) { ... }" y otra sentencia "do { ... } while (condición)". Si la condición es la misma en ambos, ¿son equivalentes ambas sentencias de control de flujo?.

No. Ver apuntes.

7. Define el concepto de objeto.

Ver apuntes.

8. En Java podemos definir una constante mediante el modificador de atributo "final". ¿Es posible utilizar este modificador para un método?. Razona la respuesta.

Si. Ver apuntes.

9. ¿Qué significa que una clase es "final"?.

Ver apuntes.

10. El uso de los arrays (matrices) en C y en Java es muy similar. ¿Sabrías decir alguna diferencia?.

Ver apuntes.

11. ¿Qué sentencias se utilizan para capturar los errores?. ¿Qué ventajas tiene el uso de estas sentencias en vez de utilizar las técnicas tradicionales?.

Ver apuntes.

12. Explica el comportamiento del manejador de excepciones por defecto cuando se produce un error, es decir, cuando nosotros no capturamos las excepciones y se produce un error, tanto para aplicaciones en modo texto como para aplicaciones gráficas y applets.

Ver apuntes.

13. Diferencias entre los componentes de los paquetes gráficos AWT y Swing. ¿Qué nombre reciben en cada caso?.

Peso pesado y peso ligero. Ver apuntes.

14. En una aplicación gráfica, ¿por qué es necesario establecer un tamaño con "setSize" antes de que se visualice (show)?.

Ver apuntes.

15. Cuando desarrollamos aplicaciones de tipo gráfico con AWT, ¿para qué se utilizan los gestores de esquemas o "Layout"? Enumera alguno.

Ver apuntes.

16. ¿Cuál es la relación que tienen las clases adaptadoras con las interfaces?.

Ver apuntes.

17. Describe brevemente los pasos a seguir para crear una barra de menú con varios menús, y que cada menú tenga varias opciones de menú, y cómo capturar sus eventos, es decir, realizar alguna acción cuando seleccionemos cualquier opción de algún menú.

Ver apuntes.

18. ¿En qué situaciones se ejecuta automáticamente el método paint()?. ¿Existe alguna forma de forzar su ejecución?. Razona las respuestas.

Ver apuntes.

19. ¿En qué consiste la técnica de seriación/deseriación de objetos?. ¿Para qué sirve declarar un atributo con el modificador transient?.

Ver apuntes.

20. En cuanto a la forma de ejecución, ¿cuál es la principal diferencia entre una aplicación (standalone) y un applet?

Ver apuntes.

Programación (5.0 puntos)

1. (1.0 puntos)

Escribir una clase en Java llamada “**Círculo**” y otra llamada “**Rectángulo**”, que representen objetos de esos nombres. Estas clases deberán tener los datos y operaciones que se definen a continuación.

Los datos para la clase Círculo son:	Los datos para la clase Rectángulo son:
<ul style="list-style-type: none">- x e y: centro del círculo.- radio: radio del círculo.	<ul style="list-style-type: none">- x e y: coordenadas de la esquina superior izquierda.- ancho y alto: ancho y alto del rectángulo.

Además, ambas clases tendrán un único **constructor** que recibe todos los datos anteriores en cada caso, y los **métodos** que se describen a continuación:

Método	Descripción	Recibe	Retorna
area	Calcular el área del objeto.	Nada	El área
toString	Construye una cadena de texto con los datos del objeto.	Nada	La cadena construida
dibujar	Dibuja el objeto gráficamente en el contexto gráfico recibido.	Graphics g	Nada

 Ayuda	Para dibujar figuras se utiliza la clase Graphics del paquete <code>java.awt</code> . <ul style="list-style-type: none">- <code>drawOval(x, y, ancho, alto)</code> --> Dibuja un círculo- <code>drawRect(x, y, ancho, alto)</code> --> Dibuja un rectángulo
---	--

NOTA: en este ejercicio no habrá función principal (`main`), ya que se utilizarán en los ejercicios siguientes; caso de no realizar este ejercicio, el alumno deberá suponer que estas clases existen, y con esta funcionalidad.

2. (2.0 puntos)

Este ejercicio consta de dos partes o clases: (1) `ListaFiguras` y (2) `TestFiguras`.

1. **ListaFiguras**: se trata de una clase que tenga una lista de figuras de las clases `Círculo` o `Rectángulo`. Las figuras se almacenarán en un objeto de la clase `Vector`. Además, esta clase tendrá los métodos siguientes:

Método	Descripción	Recibe	Retorna
añadir	Añadir un círculo o rectángulo generado al azar, con coordenadas y dimensiones también generadas al azar. Los valores estarán comprendidos entre 0 y una constante <code>VALOR_MAX</code> (cualquier valor).	Nada	Nada
tamaño	Retorna el número de figuras de la lista (número de elementos del vector).	Nada	Un entero
limpiar	Elimina todos los elementos de la lista.	Nada	Nada
listar	Construye una cadena de texto con el listado de todos los elementos de la lista.	Nada	La cadena construida
leer	Lee una lista de figuras grabada previamente desde el nombre de archivo especificado.	Nombre del archivo	Nada
grabar	Graba la lista de figuras actual en el archivo especificado.	Nombre del archivo	Nada
dibujar	Dibuja gráficamente todos los objetos de la lista en el contexto gráfico especificado.	Graphics g	Nada

 Ayuda	<p>Para la lista se utiliza un objeto de la clase <code>Vector</code> del paquete <code>java.util</code>.</p> <ul style="list-style-type: none"> - <code>add(Object o)</code> --> añade un elemento al vector. - <code>remove(Object o)</code> --> elimina un elemento del vector. - <code>removeAllElements()</code> --> elimina todos los elementos del vector. - <code>size()</code> --> Número de elementos del vector. - <code>get(int i)</code> --> Retorna el objeto de la posición i-ésima. - <code>elements()</code> --> retorna una enumeración. <p>Para generar números aleatorios se puede utilizar la función estática <code>random()</code>, de la clase <code>Math</code>.</p> <ul style="list-style-type: none"> - Esta función retorna un valor <code>double</code> comprendido entre 0.0 y 1.0 ($0.0 \leq n < 1.0$), de forma aleatoria. Por tanto, deberemos multiplicarlo por el límite superior del rango de números que se desea producir. - Para elegir al azar si es un <code>Círculo</code> o un <code>Rectángulo</code> hay que generar un valor aleatorio entre 0.0 y 2.0, y convertirlo a entero, obteniendo un valor 0 o 1. Se puede tomar cualquier criterio: 0-Círculo, 1-Rectángulo.
--	--

2. **TestFiguras**: este será un programa de prueba de todas las clases anteriores, y ya tendrá función principal (`main`). Recibe desde línea de órdenes 2 argumentos: un nombre de archivo y un número de figuras a generar aleatoriamente (mayor que 0). Lo que debe hacer es utilizar un objeto de la clase `ListaFiguras` anterior, y llamar el número de veces especificado al tercer método `añadir`, y grabar la lista al final con el método `grabar` y el nombre de archivo especificado. El programa debe comprobar que los argumentos recibidos desde línea de órdenes sean correctos (número y valores), mostrando un mensaje con la forma de uso en caso contrario. Ejemplos de uso:

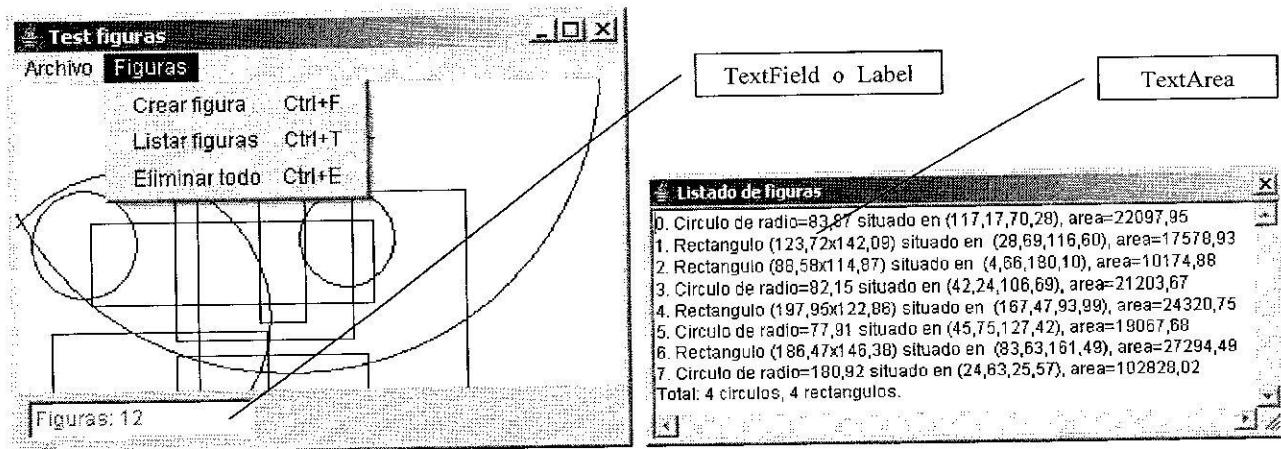
```
C:\>java TestFiguras
Uso: java TestFiguras <archivo> <numero-figuras>
C:\> java TestFiguras figuras.bin A
Argumentos incorrectos. Segundo argumento es numérico > 0.
C:\> java TestFiguras figuras.bin 50
Ok
```

3. (2.0 puntos)

Escribir un programa gráfico en Java (AWT) llamado "**TestFigurasApp**" con la barra de menú y aspecto gráfico que se muestra a continuación, y con la funcionalidad que se describe. En resumen, se trata de utilizar la clase **ListaFiguras** del ejercicio anterior para ir creando figuras **Círculo** o **Rectángulo** de una en una, de posición y tamaño aleatorio, y que aparezcan dibujadas en la ventana.

Aspecto gráfico:

- Las ventanas deben de **cerrarse/ocultarse** al pulsar el botón "X" de la ventana.
- En la parte inferior aparecerá el **número de figuras** creadas hasta el momento.
- Se valorará el uso de los **gestores de esquemas** más adecuados.
- Se valorará el uso de **teclas rápidas** en las opciones de los menús.



Funcionalidad del programa:

El usuario interactúa con el programa únicamente con las opciones de la barra de menú, las cuales se describen a continuación. En la mayoría de ellas se utilizan los métodos de la clase **ListaFiguras** que se indica. Hay que tener en cuenta también que para dibujar las figuras hay que invocar al método **dibujar** de **ListaFiguras**, lo cual se hace en el método **paint** de la ventana.

- **Archivo > Leer:** lee un archivo de figuras (método **leer** de **ListaFiguras**) y lo muestra en la ventana (forzando un refresco de la ventana), actualizando también la barra de estado. El nombre del archivo es constante (ver nota inferior ☺ si quieras subir o asegurarte una buena nota).
- **Archivo > Grabar:** graba las figuras (método **grabar** de **ListaFiguras**) actualmente mostradas en un archivo. El nombre del archivo es constante (ver nota inferior ☺).
- **Archivo > Salir:** finaliza el programa.
- **Figuras > Crear figura:** añade una nueva figura al dibujo de forma aleatoria (método **añadir** de **ListaFiguras**), dibujando todas las figuras (forzando un refresco) y actualizando la barra de estado.
- **Figuras > Listar figuras:** muestra un nuevo diálogo que muestra en una zona de texto (**TextArea**) el listado de las figuras que componen nuestro dibujo (método **listar** de **ListaFiguras**).
- **Figuras > Limpiar:** elimina todas las figuras del dibujo (método **limpiar** de **ListaFiguras**), actualizando la ventana (forzando un refresco) y la barra de estado.

☺ Todo lo anterior es lo que se pide, y con lo que puede obtenerse la máxima nota (10.0). No obstante, si te sobra tiempo y sabes hacerlo, puedes seguir con lo siguiente, lo cual puede ser substituto de alguna parte anterior que no hayas realizado correctamente. Se trata de mostrar un diálogo estándar **FileDialog** para pedir al usuario el nombre del fichero a leer y grabar, en lugar de utilizar un nombre fijo.

Solución

```
/** =====
 * Archivo : Circulo.java
 * Descripción: Gestión de figuras de tipo círculo
 * ===== */
import java.io.*;          // Para Serializable
import java.awt.*;          // Para Graphics

public class Circulo implements Serializable {
    private double x, y, radio;           // Datos

    Circulo (double x, double y, double r) { // Constructor
        this.x=x; this.y=y; this.radio=r;
    }

    public double area () {                // Área del círculo
        return (Math.PI*radio*radio);
    }

    public String toString () {
        String s = "Círculo de radio=" + radio +
            " situado en (" + x + "," + y + ")," + " area=" + area();
        return (s);
    }

    public void dibujar (Graphics g) {
        g.drawOval ((int)(x-radio), (int)(y-radio), (int)(radio*2), (int)(radio*2));
    }
}

/** =====
 * Archivo : Rectangulo.java
 * Descripción: Gestión de figuras de tipo rectángulo
 * ===== */
import java.io.*;          // Para Serializable
import java.awt.*;          // Para Graphics

public class Rectangulo implements Serializable {
    /** Coordenada x,y de la esquina superior izq. del rectángulo, y dimensiones */
    private double x, y, ancho, alto;           // Datos

    Rectangulo (double x, double y, double ancho, double alto) { // Constructor
        this.x=x; this.y=y;
        this.ancho=ancho; this.alto=alto;
    }

    public double area () {                // Área del rectángulo
        return (ancho*alto);
    }

    public String toString () {
        String s = "Rectángulo (" + ancho + "x" + alto +
            ") situado en (" + x + "," + y + ")," + " area=" + area();
        return (s);
    }

    public void dibujar (Graphics g) {
        g.drawRect ((int)x, (int)y, (int)ancho, (int)alto);
    }
}
```

```

/** =====
 * Archivo    : ListaFiguras.java
 * Descripción: Gestión de una lista de figuras Circulo o Rectangulo
 * ===== */
import java.util.*;
import java.io.*;
import java.awt.*;

/** Lista de figuras de tipo Circulo o Rectangulo*/
public class ListaFiguras {
    /** Valor maximo para los parametros de las figuras (x,y,radio,ancho,alto) */
    private static final double VALOR_MAX = 200.0;
    /** Vector para almacenar las figuras creadas */
    private Vector v = new Vector();

    /** Añadimos a la lista un Circulo o un Rectangulo generado al azar */
    public void añadir () {
        int     tipo;
        double x,y,radio,ancho,alto;
        Circulo cir;
        Rectangulo rec;

        // Elegimos al azar el tipo de figura (0:Circulo, 1:Rectangulo)
        tipo = (int) (Math.random()*2.0);
        // Creamos un nuevo objeto del tipo elegido
        x = Math.random()*(VALOR_MAX+1.0);
        y = Math.random()*(VALOR_MAX+1.0);
        switch (tipo) {
            case 0: // Circulo
                radio = Math.random()*(VALOR_MAX+1.0);
                cir   = new Circulo(x,y,radio);
                v.add (cir);
                break;
            case 1: // Rectangulo
                ancho = Math.random()*(VALOR_MAX+1.0);
                alto  = Math.random()*(VALOR_MAX+1.0);
                rec   = new Rectangulo(x,y,ancho,alto);
                v.add (rec);
                break;
            default:
        }
    }

    /** Numero de figuras del vector */
    public int tamaño () {
        return (v.size());
    }

    /** Eliminar todas las figuras creadas */
    public void limpiar () {
        v.removeAllElements();
    }

    /** Listamos las figuras que tenemos almacenadas en la lista */
    public String listar () {
        int i, nCir, nRec;
        Object o;
        String s="";

        nCir = nRec = 0;
        for (i=0; i<v.size(); i++) {
            o = v.get(i);
            if (o instanceof Circulo) nCir++;

```

```

        if (o instanceof Rectangulo) nRec++;
        s += i + " " + o.toString() + "\n";
    }
    s += "Total: " + nCir + " circulos, " + nRec + " rectangulos.";
    return (s);
}

/** Leemos del fichero especificado una lista de figuras ya creadas */
public void leer (String fichero) {
    try {
        FileInputStream fis = new FileInputStream(fichero);
        ObjectInputStream ois = new ObjectInputStream(fis);
        v = (Vector) ois.readObject();
        ois.close();
    } catch (FileNotFoundException e) {
        System.out.println("Error: Archivo <" + fichero + "> no encontrado.");
    } catch (IOException e) {
        System.out.println("Error en el dispositivo de E/S.");
    } catch (ClassNotFoundException e) {
    }
}

/** Grabamos en el fichero especificado la lista de figuras ya creadas */
public void grabar (String fichero) {
    try {
        FileOutputStream fos = new FileOutputStream(fichero);
        ObjectOutputStream oos = new ObjectOutputStream(fos);
        oos.writeObject(v);
        oos.close();
    } catch (IOException e) {
        System.out.println("Error en el dispositivo de E/S.");
    }
}

/** Dibujar las figuras en el contexto gráfico especificado */
public void dibujar (Graphics g) {
    int i;
    Object o;
    Circulo cir;
    Rectangulo rec;

    for (i=0; i<v.size(); i++) {
        o = v.get(i);
        if (o instanceof Circulo) {
            cir = (Circulo) o;
            cir.dibujar (g);
        } else if (o instanceof Rectangulo) {
            rec = (Rectangulo) o;
            rec.dibujar (g);
        }
    }
}

/*
=====
*/

```

```

/** =====
 * Archivo      : TestFiguras.java
 * Descripción: Programa de prueba de las clases Circulo, Rectangulo, y ListaFiguras
 * =====
public class TestFiguras {
    public final static void main (String args[]) {
        int i, n=0;
        String fichero;
        ListaFiguras lista = new ListaFiguras();

        // Comprobamos los parametros de una de las formas de uso
        if (args.length != 2) {
            System.out.println ("Forma de uso: TestFiguras <archivo> <N> (con N>0)");
            System.exit(1);
        }
        fichero = args[1];
        try {
            // Obtenemos el valor numérico (numero de figuras a crear)
            n = Integer.parseInt(args[2]);
        } catch (NumberFormatException e) {
            System.err.println("Valor incorrecto.");
            System.exit(1);
        }
        // El valor debe ser >= 0
        if (n < 0) {
            System.err.println("Valor debe ser >= 0");
            System.exit(1);
        }
        // Generamos el numero de figuras especificadas
        for (i=0; i<n; i++) {
            lista.añadir();
        }
        // Grabamos
        lista.grabar (fichero);
        System.out.println ("Ok");
    }
}

// =====
// Titulo :      TestFigurasApp
// Fichero:      TestFigurasApp.java
// =====
import java.awt.*;
import java.awt.event.*;

public class TestFigurasApp {
    public static void main (String sArgs[]) {
        TestFigurasFrame f;

        f = new TestFigurasFrame ("Test figuras");
        f.setSize (400,300);
        f.setVisible(true);
    }
}

```

```

class TestFigurasFrame extends Frame implements ActionListener
{
    /** Nombre del fichero donde se guardan las figuras */
    private final static String FICHERO = "figuras.bin";

    /** Lista de figuras creadas */
    private ListaFiguras lista = new ListaFiguras();

    // --- Barra de menu (inicio) -----
    MenuBar menubar;
    Menu menuArchivo, menuFiguras;
    MenuItem menuItemLeer, menuItemGrabar, menuItemSalir,
    MenuItem menuItemCrear, menuItemListar, menuItemEliminarTodo;
    MenuShortcut shortcut;
    // --- Barra de menu (fin) -----

    Panel pSur;
    TextField txtMensaje;

    TestFigurasFrame (String sTitle) {
        super(sTitle);
        setLayout (new BorderLayout());

        // --- Barra de menu (inicio) -----
        menuArchivo = new Menu("Archivo");
        shortcut = new MenuShortcut(KeyEvent.VK_L); // CTRL+L
        menuItemLeer = new MenuItem("Leer", shortcut);
        menuItemLeer.addActionListener(this);
        menuArchivo.add(menuItemLeer);
        //
        shortcut = new MenuShortcut(KeyEvent.VK_G); // CTRL+G
        menuItemGrabar = new MenuItem("Guardar", shortcut);
        menuItemGrabar.addActionListener(this);
        menuArchivo.add(menuItemGrabar);
        //
        menuArchivo.addSeparator();
        //
        shortcut = new MenuShortcut(KeyEvent.VK_S); // CTRL+S
        menuItemSalir = new MenuItem("Salir", shortcut);
        menuItemSalir.addActionListener(this);
        menuArchivo.add(menuItemSalir);
        // ---
        menuFiguras = new Menu("Figuras");
        shortcut = new MenuShortcut(KeyEvent.VK_F); // CTRL+F
        menuItemCrear = new MenuItem("Crear figura", shortcut);
        menuItemCrear.addActionListener(this);
        menuFiguras.add(menuItemCrear);
        //
        shortcut = new MenuShortcut(KeyEvent.VK_T); // CTRL+T
        menuItemListar = new MenuItem("Listar figuras", shortcut);
        menuItemListar.addActionListener(this);
        menuFiguras.add(menuItemListar);
        //
        shortcut = new MenuShortcut(KeyEvent.VK_O); // CTRL+O
        menuItemEliminarTodo = new MenuItem("Limpiar", shortcut);
        menuItemEliminarTodo.addActionListener(this);
        menuFiguras.add(menuItemEliminarTodo);
        //
        menubar = new MenuBar();
        menubar.add(menuArchivo);
        menubar.add(menuFiguras);
        setMenuBar(menubar);
        // --- Barra de menu (fin) -----
    }
}

```

```

// --- Barra de estado ---
pSur = new Panel();
pSur.setLayout (new FlowLayout(FlowLayout.LEFT));
add (pSur, BorderLayout.SOUTH);
txtMensaje = new TextField ("Figuras: 0      ");
txtMensaje.setEnabled(false);
pSur.add (txtMensaje);
pSur.setBackground (Color.LIGHT_GRAY);

// Eventos de la ventana
addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0); }});
}

// Al pulsar cualquier botón ...
public void actionPerformed (ActionEvent e) {
    if (e.getSource()==menuItemLeer) {
        lista.leer (FICHERO);
    } else if (e.getSource()==menuItemGrabar) {
        lista.grabar (FICHERO);
    } else if (e.getSource()==menuItemSalir) {
        System.exit(0);
    } else if (e.getSource()==menuItemCrear) {
        lista.añadir();
    } else if (e.getSource()==menuItemListar) {
        DlgFigurasListar dlg = new DlgFigurasListar (this, lista);
        dlg.setVisible(true);
    } else if (e.getSource()==menuItemEliminarTodo) {
        lista.eliminarTodo();
    }
    // Despues de casi todas las opciones hay que redibujar la imagen
    txtMensaje.setText ("Figuras: "+lista.tamaño());
    repaint();
}

// Redibujar el frame
public void paint (Graphics g) {
    lista.dibujar(g);
}
}

class DlgFigurasListar extends Dialog {
    TextArea txtArea;

    DlgFigurasListar (Frame f, ListaFiguras lista) {
        super(f, "Listado de figuras", true);
        setLayout (new GridLayout(1,1));
        setSize (150, 100);
        txtArea = new TextArea(lista.listar());
        add (txtArea);

        // Eventos de la ventana
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                setVisible(false); }});
    }
}

```





Dpto. Física y Arquitectura de Computadores

Área de Arquitectura y Tecnología de Computadores

Programación Avanzada

Convocatoria Febrero 2010 - ITT-ST

- # No se permiten teléfonos móviles.
- # Carpetas, mochilas, ... lejos.
- # Entregar todo el papel suministrado.
- # Escribir por ambas caras.

Nombre: _____ DNI: _____

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

Teoría - Cuestionario (0.2 cada una, total 4.0 puntos)

Programación (5.0 puntos)

1. Clases para puertas lógicas (1.25 puntos)

Escribir las siguientes clases para representar objetos de varios tipos de puertas lógicas: AND, NAND, OR, NOR, y NOT. Serán puertas lógicas de dos entradas (salvo la NOT que obviamente tendrá sólo una). Son clases muy básicas que no tienen ningún dato, ni ningún constructor, tan sólo tendrán las siguientes dos funciones:

Función	Descripción
public int valor (int a, int b);	Recibe los dos valores que tiene la puerta lógica en las entradas (0 o 1) y retorna el valor que tendrá en la salida (0 o 1). En el caso de la puerta NOT (por simplicidad), también recibe ambos valores, aunque sólo se utilizará el primero.
public String tablaVerdad ();	Retorna una cadena con la tabla de verdad de la puerta (siempre será la misma cadena).

IMPORTANTE:

1. Se valorará el uso de:
 - a. Herencia (uso de super)
 - b. Clases y funciones abstractas (las 5 clases mencionadas no pueden ser abstractas).
 - c. Bucles
 - d. El menor código repetido posible (con todo lo anterior se consigue).
2. En este ejercicio no hay función principal `main`. Estas clases se utilizarán en los siguientes ejercicios. Caso de no hacer este ejercicio, se supondrá que estas clases ya existen y tienen la funcionalidad descrita, independientemente de su implementación.

2. Programa "Logica" en modo texto (1.25 puntos)

Escribir un programa en Java llamado "Logica" que reciba desde línea de órdenes la función a realizar (AND, NAND, OR, NOR, NOT) y los valores de las entradas (0 o 1). Tras procesar los argumentos y chequear posibles excepciones, el programa mostrará por pantalla la tabla de verdad de la puerta indicada, así como el valor de la salida para las entradas especificadas, utilizando las clases del ejercicio anterior (creando un objeto de las clases que sean necesarias).

Se deberá comprobar que el número de argumentos recibidos es válido: 3 (2 si se trata de una puerta NOT), y que éstos son válidos. Para comparar cadenas, convendría convertirlas todas a mayúsculas (`toUpperCase`). Para los valores de las entradas (0 o 1), habrá que convertirlos a valores enteros (`parseInt`). En caso de detectar algún argumento incorrecto, el programa deberá mostrar un mensaje de error o la forma de uso y terminar (ver ejemplos).

Ejemplos:

C:\>java Logica Uso: Logica OPERACION v1 [v2]	C:\>java Logica AND 1 ERROR: se requieren dos argumentos
C:\>java Logica XOR 1 0 ERROR: operación no implementada	C:\>java Logica AND 1 1 a b s ----- 0 0 0 0 1 0 1 0 0 1 1 1 AND(1,1) = 1
C:\>java Logica NOT 1 a s ----- 0 1 1 0 NOT(1,0) = 0	

3. Programa "Logica" en modo gráfico (2.5 puntos)

Escribir un programa gráfico en Java (utilizando AWT) llamado "LogicaApp" con el aspecto gráfico que se muestra a continuación, y la funcionalidad descrita abajo. En resumen, se trata de hacer algo similar al ejercicio anterior, pero ahora en modo gráfico.

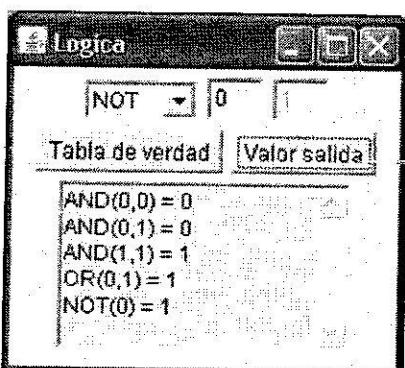


Fig. 1

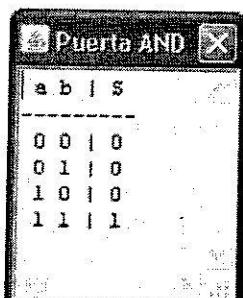


Fig. 2

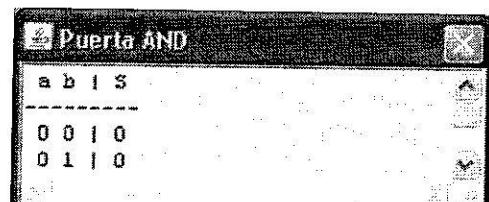


Fig. 3

Funcionalidad:

- La ventana (Frame) tendrá el título y los componentes mostrados (ver Fig. 1), y se debe poder cerrar al pulsar el botón "X". Se valorará el uso de los gestores de esquemas más adecuados.
- El usuario podrá seleccionar la operación a realizar de una lista desplegable (Choice).
- Los valores de las dos entradas se escriben en dos campos de texto (TextFields). Un detalle: cuando se selecciona la puerta NOT, puesto que sólo se utiliza el primer valor de las entradas, convendría deshabilitar el segundo nada más seleccionar dicha puerta de la lista.
- Al pulsar un botón (Button) [Tabla de verdad] se mostrará un dialogo (Dialog) en modo modal con la tabla de verdad de la puerta seleccionada (información que habrá que pasar el diálogo). Habrá que hacer uso de las clases del ejercicio 1. Tener en cuenta también el uso de gestores de esquemas en el diálogo para que el tamaño de la zona de texto (TextArea) se ajuste siempre al tamaño del mismo (ver Fig. 2 y 3). Un detalle: se puede observar que la fuente de letra de la tabla de verdad es distinta (Monospaced); se valorará si se establece dicha fuente (Font) a la zona de texto.
- Al pulsar otro botón [Valor salida], se obtendrá el valor de las entradas (getText), se convertirán a entero como en el ejercicio 2 (parseInt), y se comprobará que son correctas. Después se averiguará el valor de la salida, y se añadirá una línea más a la zona de texto (TextArea) de la parte inferior (setText). La línea será del estilo a las mostradas en la Fig. 1. Aquí también habrá que hacer uso de las clases del ejercicio 1.

Solución – Ejercicio 1:

```
// Puerta genérica
abstract class Puerta {
    public abstract int valor (int v1, int v2);
    public String tablaVerdad () {
        String s = " a b | S\n" + "-----\n";
        for (int a=0; a<=1; a++) {
            for (int b=0; b<=1; b++) {
                s += " " + a + " " + b + " | " + valor(a,b) + "\n";
            }
        }
        return (s);
    }
}

// Puerta AND
class AND extends Puerta {
    public int valor (int v1, int v2) {
        int s = ((v1==1) && (v2==1) ? 1 : 0);
        return (s);
    }
}

// Puerta NAND
class NAND extends AND {
    public int valor (int v1, int v2) {
        int s = (super.valor(v1,v2)==0 ? 1 : 0);
        return (s);
    }
}

// Puerta OR
class OR extends Puerta {
    public int valor (int v1, int v2) {
        int s = ((v1==1) || (v2==1) ? 1 : 0);
        return (s);
    }
}

// Puerta NOR
class NOR extends OR {
    public int valor (int v1, int v2) {
        int s = (super.valor(v1,v2)==0 ? 1 : 0);
        return (s);
    }
}

// Puerta NOT
class NOT extends Puerta {
    public int valor (int v1, int v2) {
        int s = ((v1==0) ? 1 : 0);
        return (s);
    }
    public String tablaVerdad () {
        String s = " a | S\n" + "-----\n";
        for (int a=0; a<=1; a++) {
            s += " " + a + " | " + valor(a,0) + "\n";
        }
        return (s);
    }
}
```

Solución – Ejercicio 2:

```
public class Logica {
    public static void main (String[] args) {
        String operacion;
        int v1=0, v2=0;
        Puerta p=null;

        n = args.length;
        if (n < 2) {
            // Si no hemos escrito los argumentos correctos mostramos mensaje
            // con la forma de uso
            System.out.println("Uso: Logica OPERACION v1 [v2]");
            System.exit(1);
        }
        // Extraemos la operacion (en mayúscula)
        operacion = args[0].toUpperCase();
        // Averiguamos si es una operación válida
        if (operacion.equals("AND") || operacion.equals("NAND") ||
            operacion.equals("OR") || operacion.equals("NOR")) {
            // Necesitamos 2 argumentos
            if (n < 3) {
                System.out.println ("ERROR: se requieren dos argumentos");
                System.exit(1);
            }
            // Extraemos los dos argumentos (como int o como char)
            try {
                v1 = Integer.parseInt(args[1]);
                v2 = Integer.parseInt(args[2]);
            } catch (NumberFormatException e) {
                System.err.println ("Argumento incorrecto; debe ser sólo 0 o 1");
                System.exit(1);
            }
            if ((v1<0) || (v1>1) || (v2<0) || (v2>1)) {
                System.out.println ("Argumento incorrecto; debe ser solo 0 o 1");
                System.exit(1);
            }
        } else if (operacion.equals("NOT")) {
            // Extraemos el único argumento (como int)
            try {
                v1 = Integer.parseInt(args[1]);
            } catch (NumberFormatException e) {
                System.err.println ("Argumento incorrecto; debe ser solo 0 o 1");
                System.exit(1);
            }
        } else {
            System.out.println("ERROR: operación no valida");
            System.exit(1);
        }
        // Creamos un objeto puerta de la clase correspondiente
        if (operacion.equals("AND")) p = new AND();
        if (operacion.equals("NAND")) p = new NAND();
        if (operacion.equals("OR")) p = new OR();
        if (operacion.equals("NOR")) p = new NOR();
        if (operacion.equals("NOT")) p = new NOT();
        // Averiguamos la salida
        System.out.println ( p.tablaVerdad() );
        System.out.println ( operacion + "(" + v1 + "," + v2 + ") = " +
                            p.valor(v1, v2) + "\n" );
    }
}
```

Solución – Ejercicio 3:

```
import java.awt.*;
import java.awt.event.*;

/** Clase principal */
public class LogicaApp {
    /** Función principal: simplemente crea un objeto Frame */
    public static void main ( String Args[] ) {
        MiFrame f;

        f = new MiFrame ();
        f.setSize (220,200);
        f.show();
    }
}

/* Clase que define nuestro objeto Frame (aplicación)
   ActionListener: para los eventos de los botones */
class MiFrame extends Frame implements ActionListener, ItemListener
{
    Panel      pSur, pCentro;      // Panel para mostrar resultados
    Choice     chOperacion;
    TextField  txtV1, txtV2;       // Area de texto para el display
    TextArea   txtSalida;
    Button     pbTabla, pbSalida;

    public MiFrame () {
        super ("Logica");
        // En el Frame establecemos BorderLayout como gestor de esquemas
        setLayout (new BorderLayout ());

        // Panel para el Display: en la parte norte del BorderLayout (NORTH)
        pCentro = new Panel();
        add (pCentro, BorderLayout.CENTER);
        pCentro.setLayout (new FlowLayout());
        chOperacion = new Choice();
        chOperacion.add ("AND");
        chOperacion.add ("NAND");
        chOperacion.add ("OR");
        chOperacion.add ("NOR");
        chOperacion.add ("NOT");
        chOperacion.addItemListener(this);
        txtV1 = new TextField ("0", 1);
        txtV2 = new TextField ("0", 1);
        pbTabla = new Button ("Tabla de verdad");
        pbSalida = new Button ("Valor salida");
        pbTabla.addActionListener (this);
        pbSalida.addActionListener (this);
        txtSalida = new TextArea ("", 5, 20);
        txtSalida.setEditable(false);
        pCentro.add (chOperacion);
        pCentro.add (txtV1);
        pCentro.add (txtV2);
        pCentro.add (pbTabla);
        pCentro.add (pbSalida);
        pCentro.add (txtSalida);

        // Eventos de la ventana
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose(); }});
    }
}
```

```

/* Gestor de eventos de la interfaz ActionListener */
public void actionPerformed (ActionEvent event) {
    Button boton = (Button) event.getSource();
    String operacion="", salida="";
    int v1=0, v2=0;
    Puerta p=null;

    // Obtenemos operacion seleccionada
    operacion = chOperacion.getSelectedItem();
    // Creamos un objeto puerta de la clase correspondiente
    if (operacion.equals("AND")) p = new AND();
    if (operacion.equals("NAND")) p = new NAND();
    if (operacion.equals("OR")) p = new OR();
    if (operacion.equals("NOR")) p = new NOR();
    if (operacion.equals("NOT")) p = new NOT();
    // Averiguamos el botón que hemos pulsado
    if (boton==pbTabla) {
        DlgDetalles dlg = new DlgDetalles (this, operacion, p);
        dlg.show();
    } else if (boton==pbSalida) {
        // Extraemos los dos argumentos (como int)
        try {
            v1 = Integer.parseInt(txtV1.getText());
            v2 = Integer.parseInt(txtV2.getText());
        } catch (NumberFormatException e) {
            return;
        }
        // Comprobamos que los valores son 0 o 1
        if ((v1<0) || (v1>1) || (v2<0) || (v2>1)) {
            return;
        }
        // Generamos texto con la linea resultante
        salida = txtSalida.getText() + operacion;
        if (p instanceof NOT) {
            salida += "(" + v1 + ")";
        } else {
            salida += "(" + v1 + "," + v2 + ")";
        }
        salida += " = " + p.valor(v1, v2) + "\n";
        txtSalida.setText (salida);
    }
}

// Eventos al seleccionar operación
public void itemStateChanged (ItemEvent e) {
    String operacion="";

    // Obtenemos operacion seleccionada
    operacion = chOperacion.getSelectedItem();
    // Habilitamos el segundo valor sólo si no es operacion NOT
    if (operacion.equals("NOT")) {
        txtV2.setEnabled (false);
    } else {
        txtV2.setEnabled (true);
    }
}

```

```

// Dialogo con los detalles
class DlgDetalles extends Dialog {
    TextArea tablaVerdad;
    Font fuente;

    DlgDetalles (Frame f, String operacion, Puerta p) {
        super (f, "Puerta " + operacion, true);
        setSize (150,130);
        // En el Frame establecemos GridLayout como gestor de esquemas,
        // para poder redimensionar el dialogo
        setLayout (new GridLayout(1,1));
        // Mostramos su tabla de verdad
        tablaVerdad = new TextArea(p.toString(), 6, 20, TextArea.SCROLLBARS_BOTH);
        // Establecemos una fuente
        fuente = new Font ("Monospaced", Font.PLAIN, 12);
        tablaVerdad.setFont (fuente);
        add (tablaVerdad);

        // Eventos de la ventana
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose(); }});
    }
}

```



Programación Avanzada

Convocatoria Ordinaria de Febrero de 2005
Ingeniería Técnica de Telecomunicación (ITT-ST)

Alumno (apellidos y nombre)	D.N.I.	Prácticas entregadas	Firma
		<input type="checkbox"/> Prácticas estándar <input type="checkbox"/> Nada <input checked="" type="checkbox"/> Trabajo sobre	

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

1. Los programas escritos en lenguaje Java no son tan rápidos como los del lenguaje C. Esto es debido a:
 - a) Java es un lenguaje totalmente compilado
 - b) Java es un lenguaje totalmente interpretado
 - * c) Java es un lenguaje mitad compilado y mitad interpretado
2. ¿Es posible definir constantes dentro de un método?
 - a) No, sólo es posible declararlas dentro de la clase (fuera de cualquier método).
 - b) Si, aunque tendrá el mismo efecto que a nivel de clase, es decir, será visible en toda la clase.
 - * c) Si, aunque sólo será visible dentro del método en el que se define.
3. Cuando una clase "Nieta" hereda de otra clase "Padre", la cual a su vez hereda de otra clase "Abuelo", si desde un método de la clase Nieta queremos ejecutar un método "M" de la clase Abuelo escribiremos:
 - a) anidando llamadas super, una por cada nivel: super.super.M();
 - * b) Con "casting": ((Abuelo) this).M
 - c) No es posible realizar dicha llamada, ya que sólo se pueden invocar métodos de la superclase directa (Padre).
4. En programación estructurada (lenguaje C), una estructura o registro "struct" equivale (mas o menos) en POO a:
 - a) un atributo
 - * b) una clase
 - c) un método
5. ¿Es obligatorio definir algún constructor dentro de una clase?
 - a) Si, pues de lo contrario nunca podremos crear objetos
 - * b) No, ya que siempre podremos crear objetos al menos con el constructor por defecto
 - c) Depende de si la clase hereda algún constructor de alguna superclase
6. La técnica de conversión denominada "casting" se puede utilizar para:
 - a) conversiones entre datos de tipos primitivos
 - b) conversiones entre objetos de clases diferentes
 - * c) los dos anteriores
7. Los paquetes de java se pueden almacenar descomprimidos en forma de una estructura de directorios, pero también se pueden almacenar comprimidos:
 - a) en ficheros *.ZIP
 - b) en ficheros *.JAR (Java Archives)
 - * c) los dos anteriores
8. En una aplicación en modo gráfico, ¿qué es un "contenedor"?
 - * a) un componente más, pero que puede contener a otros componentes
 - b) es una librería de componentes
 - c) donde se depositan los objetos que ya no se utilizan, resultado del proceso de "recolección de basura".
9. ¿Para qué se utiliza una clase adaptadora?
 - a) es obligatoria para poder utilizar cualquiera de las interfaces "Listener"
 - * b) nos evita tener que definir todos los métodos que incorpora una interfaz de tipo "Listener"
 - c) las dos anteriores
10. ¿Es necesario compilar un programa java para ejecutarlo como applet?
 - a) Depende del navegador del usuario
 - b) No, porque los navegadores son capaces de interpretar código fuente de java insertado en su código
 - * c) Si, como cualquier otro programa escrito en java

Teoría - Cuestionario (0.2 cada una, total 4.0 puntos)

1. ¿Qué diferencia existe entre el JDK y el JRE?.
Ver apuntes.

2. Define el concepto de "alcance" o "visibilidad" de un identificador (variable, constante u objeto).
Ver apuntes.

3. ¿Es posible convertir en Java una variable de tipo "boolean" a algún dato numérico de forma automática?. ¿Y empleando la técnica del "casting"?.
¿Puede convertirse a una cadena de caracteres String?. Razona las respuestas.
No. No. Si. Ver apuntes.

4. ¿Cuáles son las condiciones para que se realice una conversión automática de una variable de un tipo a otro tipo distinto?. Pon un ejemplo.
Ver apuntes.

5. ¿Qué ventajas nos aporta el uso de constantes en cualquier programa?.
Evitar el uso de "números mágicos. Ver apuntes.

6. Enumera y pon un ejemplo de sentencias de control de flujo de las que dispone el lenguaje Java.
if-else, for, while, switch, break, continue. Ver apuntes.

7. Describe las diferencias entre el paso de argumentos desde la línea de órdenes en un programa en C y en un programa en Java.
Ver apuntes.

8. ¿Qué significa que una clase es abstracta?.
Ver apuntes.

9. ¿Es posible definir un método como abstracto (abstract) o sólo está permitido que una clase sea abstracta?. Razona la respuesta.
Si. Ver apuntes.

10. Describe cuando se ejecuta el constructor de una clase.
Ver apuntes.

11. ¿Cuál es el criterio que sigue Java para el paso de parámetros por valor y por referencia?.
Ver apuntes.

12. Escribe una diferencia entre la clase "String" y la clase "StringBuffer".
Ver apuntes.

13. ¿Cómo se especifica en Java que una clase B implementa una interfaz X, ¿cómo se escribe dicha relación en Java?.
Se utiliza la cláusula "implements": class B implements X { ... }

14. ¿Para qué sirve la técnica de gestión de errores (excepciones)?.
Ver apuntes.

15. Diferencias entre los componentes de los paquetes gráficos AWT y Swing. ¿Qué nombre reciben en cada caso?.
Peso pesado y peso ligero. Ver apuntes.

16. ¿Existe alguna alternativa al uso de gestores de esquemas para colocar los componentes de una ventana gráfica?. Razona la respuesta.
Si. Ver apuntes.

17. Java tiene definidas varias clases adaptadoras para hacer más cómoda la programación. ¿Es posible definir nosotros nuevas clases adaptadoras?. Razona la respuesta.
Si. Ver apuntes.

18. Enumera las clases de que dispone Java para crear y gestionar menús en AWT, dando una mínima descripción de para qué se utiliza cada una de ellas.
Ver apuntes.

19. ¿Dónde se colocan las instrucciones para dibujar en un componente o un contenedor?. ¿De qué clase es el objeto necesario para poder dibujar?.
En el método paint() o update(). Ver apuntes.

20. ¿En qué consiste la técnica de serialización/deserialización de objetos?. ¿Para qué sirve declarar un atributo con el modificador transient?.
Ver apuntes.

Programación (5.0 puntos)

S

1. (1.0 puntos)

Escribir una clase en Java que incluya funciones (métodos) que nos permitan convertir cantidades de una moneda a otra. Las monedas que utilizaremos son euros, pesetas (España), marcos (Alemania), francos (Francia), liras (Italia), y escudos (Portugal). Para ello, nuestra clase tendrá dos funciones estáticas y una serie de constants con el valor equivalente de 1 euro a cada una de las monedas. En resumen, la clase tendrá lo siguiente:

Class Monedas	
Field Summary (Resumen de atributos y constantes)	
Constante "PESETAS", con valor real de simple precisión (float)	166.386 (= 1 euro)
Constante "MARCOS", con valor real de simple precisión (float)	1.956 (= 1 euro)
Constante "FRANCOS", con valor real de simple precisión (float)	6.6 (= 1 euro)
Constante "LIRAS", con valor real de simple precisión (float)	1936.270 (= 1 euro)
Constante "ESCUUDOS", con valor real de simple precisión (float)	200.482 (= 1 euro)
Method Summary (Resumen de métodos o funciones)	
euro2otra	Parámetros: valor en euros (float) y una cadena de texto indicando la moneda destino ("pesetas", "marcos", "francos", "liras", y "escudos").
Uso: convertir un valor en euros (float) a un valor (float) de otra moneda especificada.	Valor retornado: el valor (float) en la moneda especificada.
otra2euro	Parámetros: valor en la moneda (float) y una cadena de texto indicando dicha moneda ("pesetas", "marcos", ...).
Uso: convertir un valor de cualquier moneda (float) a euros (float).	Valor retornado: el valor (float) en euros.

Las dos funciones anteriores utilizarán las constantes definidas para realizar las posibles conversiones.

NOTA: Aunque esta clase "Monedas" la utilizaremos en los dos siguientes problemas de programación, no es obligatorio haberla realizado; podemos suponer que esta clase ya está disponible, eso sí, con la funcionalidad descrita, y teniendo en cuenta que las funciones son estáticas.

Solución:

```
// =====
// Titulo : Monedas
// Fichero: Monedas.java
// =====

public class Monedas {
    // Constantes equivalentes a 1 euro
    private final static float pesetas = 166.386F;      // Pesetas españolas
    private final static float marcos = 1.956F;          // Marcos alemanes
    private final static float francos = 6.6F;            // Francos franceses
    private final static float liras = 1936.270F;          // Liras italianas
    private final static float escudos = 200.482F;         // Escudos portugueses

    // Constantes para nombres de monedas (recomendado)
    public final static String sEuros = "euros";
    public final static String sPesetas = "pesetas";
    public final static String sMarcos = "marcos";
    public final static String sFrancos = "francos";
    public final static String sLiras = "liras";
    public final static String sEscudos = "escudos";
}
```

```

// Convertimos un valor de una moneda a euros
public static float otro2euros (float valor, String moneda) {
    float euros = valor / cambio(moneda);
    return (euros);
}

// Convertimos euros a un valor de otra moneda
public static float euros2otro (float euros, String moneda) {
    float valor = euros * cambio(moneda);
    return (valor);
}

// Obtenemos el valor para cambiar una moneda
private static float cambio (String moneda) {
    // Por seguridad, convertirmos siempre a minusculas
    moneda = moneda.toLowerCase();
    if (moneda.equals(sPesetas)) {
        return (pesetas);
    } else if (moneda.equals(sMarcos)) {
        return (marcos);
    } else if (moneda.equals(sFrancos)) {
        return (francos);
    } else if (moneda.equals(sLiras)) {
        return (liras);
    } else if (moneda.equals(sEscudos)) {
        return (escudos);
    } else { // Será euros
        return (1.0F);
    }
}
}

```

2. (1.5 puntos)

Escribir un programa llamado "EuroConv" que realice conversiones entre euros y las antiguas monedas nacionales de diversos países y viceversa.

Entrada: el programa recibirá 3 argumentos desde la línea de órdenes (DOS), que habrá que extraer de forma segura, esto es, que contemple la posibilidad de no teclear los argumentos necesarios, o de ser incorrectos. En caso de no ser todo correcto se deberá mostrar un mensaje indicativo y abortar el programa.

Los argumentos son los siguientes:

0. cantidad a convertir (float).
1. cadena de texto con la moneda de la cantidad anterior (euros, pesetas, marcos, francos, liras, escudos).
2. cadena de texto con la moneda de destino (euros, pesetas, marcos, francos, liras, escudos).

Ayuda: para convertir una cadena de texto (String) a un valor numérico float se puede utilizar la función: `Float.parseFloat (cadena)`. Esta función puede generar la excepción "NumberFormatException", que será conveniente capturar. Esto se puede producir si el usuario, por ejemplo, teclea alguna letra, o una coma en vez del punto decimal.

Salida: el programa imprimirá por pantalla el resultado de la conversión, aproximadamente tal y como aparece en los ejemplos de abajo. Aunque no es obligatorio, se valorará si a la hora de visualizar el resultado, se utiliza 3 decimales como salida. Como ayuda se proporciona el siguiente fragmento de código:

```

import java.text.*;
// Ojo: colocar en el lugar adecuado
float n = 15.5;
DecimalFormat f = new DecimalFormat("##0.00");
System.println ("Valor: " + f.format (n)); // Nos imprime: 15.50

```

Mecanismo: para realizar las conversiones, el programa debe utilizar necesariamente la clase "Monedas" del Programación Avanzada - Convocatoria Febrero 2005

~~void flat = Flat. porFlat(variables);~~

ejercicio anterior. Como ya se ha dicho, en el caso de no haberla realizado se deberá suponer que ya existe, con la funcionalidad descrita, y teniendo en cuenta que las funciones son estáticas.

Ejemplos de uso:

C:\>java EuroConv 1 euros pesetas
El valor de 1,000 euros = 166,386
pesetas

C:\>java EuroConv 166.386 pesetas euros
El valor de 166,386 pesetas = 1,000
euros

C:\>java EuroConv 1 euros liras
El valor de 1,000 euros = 1.936,270
liras

C:\>java EuroConv 1936.270 liras euros
El valor de 1.936,270 liras = 1,000
euros

También será posible realizar conversiones entre cualquiera de las antiguas monedas. Para ello, siempre se pueden hacer dos conversiones: 1. de la moneda original a euros, y 2. de euros a la moneda destino.

C:\>java EuroConv 1936.270 liras pesetas
El valor de 1936.270 liras = 166,386
pesetas

C:\>java EuroConv 200 euros pesetas
El valor de 200,000 euros = 33.277,199
pesetas

Solución:

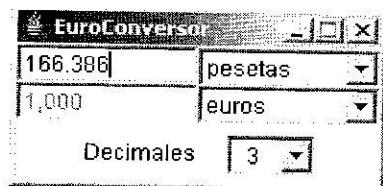
```
// =====  
// Titulo : EuroConv  
// Fichero: EuroConv.java  
// =====  
  
import java.text.*; // Para DecimalFormat  
  
public class EuroConv {  
  
    // Función principal  
    public static void main (String args[]) {  
        int n; // Número de argumentos desde la línea de órdenes  
        float cantidad = 0.0F; // Valor numérico del primer argumento  
        String origen, destino; // Monedas origen y destino  
        float euros, valor; // Valor en la moneda destino  
  
        // Comprobamos el número de argumentos: debe ser 3 (args.length)  
        // ...  
        // Obtenemos el valor numérico del argumento args[0] (parseFloat + try..catch)  
        // ...  
        // Obtenemos moneda origen y destino (args[1] y args[2] directamente)  
        // ...  
        // Convertimos a euros  
        euros = Monedas.otro2euros(cantidad, origen);  
        // Ya lo tenemos en euros; lo convertimos a la moneda destino  
        valor = Monedas.euros2otro(euros, destino);  
        // Mostramos el resultado  
        DecimalFormat f = new DecimalFormat ("###,##0.000");  
        System.out.println("El valor de "+f.format(cantidad)+" "+origen+" = "+  
                           f.format(valor)+" "+destino);  
    }  
}
```

Marc las mues
de una "1. 1"

3. (2.5 puntos)

Escribir un programa gráfico en Java utilizando la librería gráfica AWT llamado "EuroConvApp", con el aspecto que aparece en la derecha. Se trata del mismo programa del ejercicio anterior, pero esta vez, de forma gráfica.

Aspecto gráfico:



- Es una ventana (Frame) que consta de una zona de texto (TextField) para poder teclear la cantidad a convertir y una lista desplegable (Choice) para seleccionar la moneda origen, la cual podrá ser (en este orden): "euros", "pesetas", "marcos", "francos", "liras", y "escudos". Por defecto, deberá aparecer seleccionada "pesetas".
- De igual modo, tendremos otro par de componentes, una zona de texto (TextField) no editable (setEnabled) para que aparezca la cantidad convertida, y una lista similar a la anterior para seleccionar la moneda destino. Por defecto, deberá aparecer seleccionada "euros".
- Finalmente, en la parte inferior aparecerá una lista desplegable (Choice) con las posibles cifras decimales a utilizar para mostrar la cantidad resultante: 0, 1, 2, y 3. Por defecto aparecerá seleccionado el 3.
- Se valorará el uso de los gestores de esquemas más adecuados.

Acciones a realizar:

- La aplicación deberá poder cerrarse con el botón "X".
- Al cambiar la moneda seleccionada, tanto la moneda origen como la destino, se deberá extraer la cantidad tecleada en la zona de texto superior, realizar la conversión o conversiones adecuadas según las monedas origen y destino, y mostrar el resultado en la zona de texto inferior con el número de cifras decimales que se haya establecido en la lista de abajo. Para realizar las conversiones se debe utilizar la clase "Monedas" del primer ejercicio. Para convertirlo a cadena con un número concreto de cifras decimales ver el ejemplo del ejercicio anterior.
- Al teclear cualquier dígito (o letra) en la zona de texto superior, también se deberá ir actualizando la cantidad en la moneda destino. Para ello habrá que implementar la interface KeyListener (keyPress, keyReleased, keyTyped).
- Al cambiar el número de cifras decimales habrá que volver a actualizar la cantidad en la moneda destino. Como vemos, esto hay que realizarlo varias veces, por lo que se recomienda escribir una función separada para ello e invocarla donde sea necesario.

Solución:

```
// =====
// Titulo : EuroConvApp
// Fichero: EuroConvApp.java
// =====

import java.awt.*;
import java.awt.event.*;
import java.text.*;           // Para DecimalFormat

public class EuroConvApp {
    public static void main (String args[]) {
        FrameEuroConv f;

        f = new FrameEuroConv ("EuroConversor");
        f.setSize (200,100);
        f.setResizable (false);
        f.show();
    }
}
```

```

class FrameEuroConv extends Frame implements ItemListener, KeyListener {
    DecimalFormat formatos[] = {           // Formatos segun numero de decimales
        new DecimalFormat("#,#0"),         new DecimalFormat("#,#0.0"),
        new DecimalFormat("#,#0.00"),       new DecimalFormat("#,#0.000") };

    Panel      pCentro, pSur;
    TextField txtValorOrigen, txtValorDestino;
    Choice     chMonedaOrigen, chMonedaDestino;
    Choice     choiceDecimales;

    FrameEuroConv (String sTitle) {
        super(sTitle);

        // Creación de componentes - Panel Centro: datos
        txtValorOrigen = new TextField("");
        chMonedaOrigen = new Choice();
        chMonedaOrigen.add (Monedas.sEuros);
        chMonedaOrigen.add (Monedas.sPesetas);
        chMonedaOrigen.add (Monedas.sMarcos);
        chMonedaOrigen.add (Monedas.sFrancos);
        chMonedaOrigen.add (Monedas.sLiras);
        chMonedaOrigen.add (Monedas.sEscudos);
        chMonedaOrigen.select (1);      // Por defecto, pesetas
        //
        txtValorDestino = new TextField("");
        txtValorDestino.setEnabled(false);
        chMonedaDestino = new Choice();
        chMonedaDestino.add (Monedas.sEuros);
        chMonedaDestino.add (Monedas.sPesetas);
        chMonedaDestino.add (Monedas.sMarcos);
        chMonedaDestino.add (Monedas.sFrancos);
        chMonedaDestino.add (Monedas.sLiras);
        chMonedaDestino.add (Monedas.sEscudos);
        chMonedaDestino.select (0);    // Por defecto, euros

        // Creación de componentes - Panel Sur: numero cifras decimales
        Label lblDecimales = new Label("Decimales");
        choiceDecimales = new Choice();
        for (int d=0; d<formatos.length; d++) {
            choiceDecimales.add (" "+d+" ");
        }
        choiceDecimales.select(3);    // Por defecto, 3 decimales

        // Registramos eventos listas y texto
        txtValorOrigen.addKeyListener(this);
        chMonedaOrigen.addItemListener(this);
        chMonedaDestino.addItemListener(this);
        choiceDecimales.addItemListener(this);

        // Colocacion de componentes en la ventana
        setLayout (new BorderLayout());
        pCentro = new Panel();
        add (pCentro, BorderLayout.CENTER);
        pCentro.setLayout (new GridLayout(2,2));
        pCentro.add (txtValorOrigen);
        pCentro.add (chMonedaOrigen);
        pCentro.add (txtValorDestino);
        pCentro.add (chMonedaDestino);
        pSur = new Panel();
        add (pSur, BorderLayout.SOUTH);
        pSur.setLayout (new FlowLayout());
        pSur.add (lblDecimales);
        pSur.add (choiceDecimales);
        // Eventos de la ventana

```

(*) Si se hubiese definido un array con los nombres de las monedas, las listas chMonedaXXX se podrían llenar con un bucle.

```

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                System.exit(0); }});

}

// Eventos al ...
public void itemStateChanged (ItemEvent e) {
    // ... cambiar la moneda origen o destino, o decimales ...
    // No es necesario averiguar el objeto que ha generado el evento (getSource)
    actualizarValores();
}

// Eventos al ir escribiendo el valor (solo se necesita implementar 1 de los 3)
public void keyTyped      (KeyEvent e) { actualizarValores(); }
public void keyPressed     (KeyEvent e) { actualizarValores(); }
public void keyReleased    (KeyEvent e) { actualizarValores(); }

// Actualizamos los valores de los textos
private void actualizarValores() {
    float valorOrigen=0.0F, valorEuros=0.0F, valorFinal=0.0F;
    String textoFinal, monedaOrigen, monedaDestino;
    int nDecimales;

    // Obtenemos el valor a convertir
    try {
        valorOrigen = Float.parseFloat(txtValorOrigen.getText());
    } catch (NumberFormatException e) {
    }
    // Realizamos la conversion
    monedaOrigen = chMonedaOrigen.getSelectedItem();
    monedaDestino = chMonedaDestino.getSelectedItem();
    valorEuros = Monedas.otro2euros (valorOrigen, monedaOrigen);
    valorFinal = Monedas.euros2otro (valorEuros , monedaDestino);
    // Obtenemos el formato correcto segun numero de decimales
    nDecimales = choiceDecimales.getSelectedIndex();
    textoFinal = formatos[nDecimales].format(valorFinal);
    // Establecemos el texto resultante
    txtValorDestino.setText(textoFinal);
}
}

```

Programación Avanzada
Ingeniería Técnica de Telecomunicación (ITI-SE)

Teoría – Test (0.1 cada una, total 1.0 punto. Cada dos mal restan una bien)

1. El compilador de Java necesita que especifiquemos las rutas en donde puede encontrar nuestras clases. Esto lo podemos hacer de varias formas; ¿en la至上 respuesta incorrecta:
a) en la variable de entorno CLASSPATH.
b) como un parámetro a la hora de invocar a la máquina virtual (JVM).
c) como un parámetro a la hora de invocar al compilador.
2. j) Por qué crees que los diseñadores del lenguaje Java lo han hecho tan parecido al lenguaje C++?
a) Para poder utilizar los mismos compiladores, indicando mediante algún parámetro el lenguaje concreto a utilizar.
b) Java Y C no tienen nada que ver, pero lo han hecho así para facilitar su aprendizaje.
c) Los diseñadores de Sun querían hacer una ampliación al lenguaje C, pero manteniendo la compatibilidad con los programas ya desarrollados.
3. Los programas escritos en lenguaje Java no son tan rápidos como los del lenguaje C. Esto es debido a:
a) Java es un lenguaje totalmente compilado
b) Java es un lenguaje totalmente interpretado
c) Java es un lenguaje mitad compilado y mitad interpretado
4. i)Cómo se llama la utilidad del JDK que nos permite generar páginas Web a partir de unos comentarios especiales insertados en el código fuente?
a) docuweb
b) javadoc
c) javain
5. En Java, el mecanismo llamado "recogeción de basura" o "garbage collection" consiste en:
a) limpieza de la memoria utilizada por los objetos, para eliminar los objetos que ya no se utilicen.
b) compactación de cada uno de los objetos para que ocupen menos memoria, mediante ZIP, ARI, ...
c) es una llamada al sistema operativo para que aborde tareas menos utilizadas pero consumen tiempo del procesador (CPU).
6. En programación estructurada (lenguaje C), una estructura o registro "struct" equivale (más o menos) en POO a:
a) un atributo
b) una clase
c) un método
7. j)Cómo se especifica en Java que una clase B implementa una interfaz X?, dicho de otro modo, ¿Cómo se escribe dicha relación en Java?. i)Cómo afecta esto a la clase B?

Programación (6.0 puntos)

1. (2.0 puntos) – Consultad las ayudas de la última página

Escribir una clase en Java llamada "Diccionario", que sirva para almacenar una lista de palabras y realizar consultas posteriores. Esta clase, la cual deberá tener los datos y operaciones que se definen a continuación, se utilizará en el ejercicio siguiente; caso de no realizar este ejercicio, el alumno deberá suponer que esta clase existe.

4. **Lista de palabras:** habrá un array de String "palabras" con una serie de palabras iniciales, y un objeto de la clase Vector "diccionario", tal que las palabras del array se añadirán al Vector durante la creación de un objeto de esta clase Diccionario. Este Vector será el que finalmente se utilice para todo lo demás.
4. **Método "añadir":** permitirá añadir palabras al objeto Vector "diccionario" por parte del usuario, no utilizando ya el array "palabras". Además, el método debe comprobar si una palabra ya existe en el vector; en ese caso no se hace nada.
4. **Método "listar":** se trata de recorrer el vector (no el array) e imprimir por la pantalla cada palabra del diccionario. Este método recibe además un parámetro entero "longitud"; tal que si su valor es menor o igual que 1, se listará sólo las palabras de dicha longitud, y si el valor es menor o igual que 0, se mostrarán todas.
4. **Método "buscar":** consiste en buscar una palabra en el vector (no en el array) de forma aleatoria del diccionario y retornarla. Al igual que en el método "listar", este método recibe un parámetro entero "longitud"; tal que si su valor es menor o igual que 0, se retornará cualquier palabra (la primera palabra seleccionada al azar); en otro caso, la función deberá comprobar que la palabra tiene la longitud pedida, y deberá seguir reinteniéndolo hasta que la encuentre o hasta que lo haya intentando un número máximo de veces, para evitar bucles infinitos (si no hubiese ninguna). Si Finalmente, si no se hubiese encontrado ninguna palabra de la longitud solicitada, retornará "null".

- Programación Avanzada – _____
- a) El paquete raíz es "Object".
b) El paquete raíz es "Java".
c) No existe tal jerarquía de paquetes ya que los paquetes están totalmente aislados unos de otros.

Teoría - Cuestionario (0.2 cada una, total 3.0 puntos)

1. Define el concepto de "alcance" o "visibilidad" de un a una variable dentro de una clase.
2. i)Cuáles son las condiciones para que se realice una conversión automática de una variable de un tipo a otro tipo distinto? Pón un ejemplo.
3. Cuando definimos una constante dentro de una clase, ¿Por qué es conveniente especificar el modificador "static"? Pon un ejemplo de definición de una constante.
4. Enumera y pon un ejemplo de uso de sentencias de control de flujo de las que dispone el lenguaje Java (con 3 es suficiente). OJO: no se pregunta sobre el concepto de "flujo" o stream.
5. Define el concepto de método.
6. i)Qué significa que un parámetro se pase a un método por valor o por referencia?
7. El uso de los arrays (matrices) en C y en Java es muy similar. i)Sabrás decir alguna diferencia?
8. i)Cómo se especifica en Java que una clase B implementa una interfaz X?, dicho de otro modo, ¿Cómo se escribe dicha relación en Java?. i)Cómo afecta esto a la clase B?
9. j)Para qué sirve la cláusula "throws" en la gestión de excepciones?
10. En una aplicación gráfica, i)por qué es necesario establecer un tamaño con "setSize" antes de que se visualice (show)?
11. j)Es posible el uso de varios gestores de esquemas en un único contenido? Razona la respuesta.
12. i)Cuál es la relación que tienen las clases adaptadoras con las interfaces?
13. i)Qué significa que un cuadro de diálogo se muestre en un fichero. i)Cómo se llama esta técnica?. j)Qué clases se utilizan en ambos casos?
14. En Java podemos almacenar y leer objetos completos a través de una página Web de Internet. i)Se necesita tener ordenador para poder ejecutar un applet que se descarga desde una página Web de Internet. j)Se necesita tener instalada la máquina virtual de Java?.
15. i)Qué software se necesita tener instalado en nuestro ordenador para poder ejecutar un applet que se descarga desde una página Web de Internet. i)Se necesita tener instalada la máquina virtual de Java?.

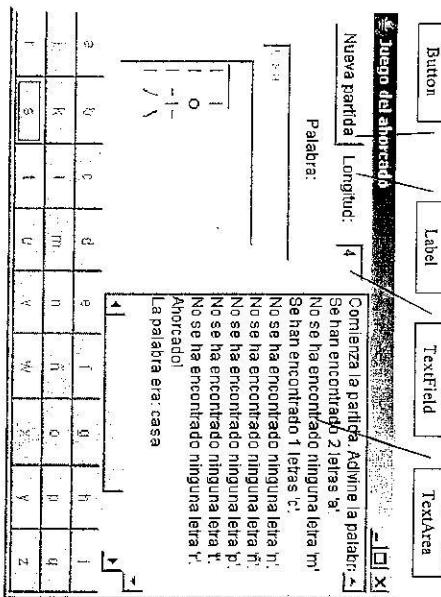
2. (4,0 puntos) – Consultad las ayudas de la última página

Escribir un programa gráfico en Java (AWT) llamado "AhorcadoApp" para jugar al clásico juego del ahorcado.

Descripción del juego: se trata de adivinar letra a letra la palabra oculta; las letras que aun no se han adivinado aparecen como guiones “_”. Cada vez que el jugador prueba una letra, si ésta aparece en la palabra, los guiones se transforman en dicha letra; en caso contrario, si la letra no aparece en la palabra, pierde una oportunidad y se le cuelga “un trozo”. La parte que se cuelga según el número de fallos es: 1-Cabeza, 2-Tronco, 3-Brazo izq., 4-Brazo dcho, 5-Pierna izq., y 6-Pierna dcha. Si el jugador pierde las 6 oportunidades que tiene y no ha adivinado la palabra estará colgado entero y habrá perdido.

Aspecto gráfico:

- El programa deberá tener un aspecto lo más parecido posible al que se muestra en la figura.
- Es necesario deshabilitar ciertos componentes: (1) el JTextField de la palabra a adivinar, (2) el TextArea dc la hora, y (3) aunque todos los botones de las letras estén inicialmente habilitados, cada vez que se pulse un botón, éste deberá deshabilitarse para evitar jugar dos veces la misma letra.
- El programa debe de cerrarse al pulsar el botón “X” de la ventana.
- Se valorará el uso de los gestores de esquemas más adecuados.



Funcionalidad del programa:

- **Inicio del juego:** cada vez que se comience una partida (pulsando el botón “Nueva Partida”), el programa obtendrá una palabra al azar de la longitud especificada, utilizando la clase “Diccionario” del primer ejercicio (habrá que declarar y crear un objeto de dicha clase y utilizar el método “Buscar”). Una vez iniciada la partida, el usuario puede ver la palabra que tiene que adivinar (con guiones) y el estado de la hora (talos que ha cometido hasta el momento).
- Elección de una letra: cada vez que el jugador prueba una letra (pulsando uno de los 27 botones inferiores), el programa busca la letra en la palabra y actúa en consecuencia: si la letra aparece en la palabra, se substituyen todos los guiones por dicha letra; en caso contrario, se incrementa el número de fallos. Después, se actualiza todo ello en la ventana. Además, para evitar que el usuario intente dos veces la misma letra, conviene también deshabilitar el botón pulsado.
- Mensajes: en la parte derecha, hay una zona de mensajes, donde se informa al usuario del resultado de cada elección de letra, y el resultado del juego (Palabra adivinada o Ahorcado). Se valorará la presentación de los mensajes.
- Fin del juego: el juego continuará hasta que el jugador haya adivinado la palabra completa (no hay guiones en la palabra oculta), o se le haya colgado completamente (hay fallado 6 veces).
- **Reinicio del juego:** al pulsar el botón “Nueva partida”, se borrarán todos los mensajes del juego anterior, se inicializarán las variables que sean precisas (número de fallos, etc.), se habilitarán/deshabillitarán los componentes necesarios, se asignarán valores a otros, y volveremos a empezar

AYUDAS (Ejercicio 1)

- ❖ No olvides escribir el constructor (es necesario).
- ❖ Lo que hay que hacer es una clase y no un “programa”, es decir, no hay función “main”.
- ❖ A la hora de manejar un objeto de la clase “Vector”:
 - Para añadir objetos: add().
 - Para obtener un elemento de una posición del vector: get(i).
 - Para conocer el número de elementos: size().
 - Para saber si el vector ya tiene un objeto “”, contains(o).
 - Para recorrer el vector puedes utilizar cualquiera de los dos métodos explicados en clase.
- ❖ Para generar números aleatorios se puede utilizar la función estática random(), de la clase Math. Esta función retorna un valor double comprendido entre 0.0 y 1.0 ($0.0 \leq n < 1.0$), de forma aleatoria. Por tanto, deberemos multiplicarlo por el límite superior del rango de números que se desea producir y sumar un valor para establecer el límite inferior. En este caso particular, el valor máximo es el número de palabras (NumPal) del vector, y como el valor inferior es 0 no es necesario hacer nada más. Así obtendremos un valor entre 0 y NumPal-1, que es lo que nos interesa. Finalmente, como lo que hemos obtenido es un double y queremos obtener un int para acceder al elemento concreto del vector, debetremos convertirlo de forma explícita a int (casting).
- ❖ Para saber la longitud de una cadena “s”: s.length().
- ❖ **AYUDAS (Ejercicio 2)**
- ❖ Los botones de las letras se pueden añadir en un panel aparte, el cual tenga el gestor de esquemas más apropiado. Puesto que son 27 botones, para no repetir código a la hora de crearlos se puede utilizar un array con las 27 letras, lo cual nos sirve para hacer un bucle.
- ❖ Una vez que ya se haya generado la palabra de la longitud adecuada, para llevar la cuenta de las letras acertadas se puede crear un array de char con tantos elementos como letras tenga dicha palabra (longitud de esa cadena), e inicializar todos los elementos con guiones “_”. Para esto último puedes hacer un bucle, o utilizar la función Arrays.fill (array, ‘_’); que inicializa todos los elementos de la array a un objeto Button con b = (Button)e.getSource();
- ❖ Cuando se capturan los eventos al pulsar los botones de las letras, para ahorrar código se puede tener en cuenta dos cosas: (1) El parámetro “e” del evento se puede convertir a un objeto Button con b = (Button)e.getSource(); y (2) Se puede obtener la etiqueta del botón con b.getLabel(). Todo esto sirve para obtener la letra pulsada con dos líneas de código, así como para deshabilitar el botón pulsado: b.setEnabled(false).
- ❖ Cúando se quiera buscar una letra en la palabra oculta, se buscará en la cadena “s” de la palabra, todas las ocurrencias del carácter “c”. Una posibilidad sencilla es hacer un bucle que recorra la cadena desde el principio (i=0) hasta la longitud de la cadena – 1 (s.length()-1), y varia extrayendo los distintos caracteres con charAt(i), si ese carácter es el que se busca, en la posición “i” del array de guiones comentado anteriormente habrá que substituir el guion por ese carácter. Alternativamente también puedes utilizar la función i=s.indexOf(c), lo cual nos da la posición del primer carácter “c” dentro de la cadena “s”, aunque la primera sugerencia es más fácil.
- ❖ En el caso de haber recorrido toda la palabra buscando un carácter y no haber encontrado ninguna ocurrencia del mismo, tendremos que incrementar el número de fallos, lo cual influye en el estado de la hora.
- ❖ Para mostrar el estado de la hora, es conveniente hacer una función aparte, que reciba el número de fallos actuales como argumento y retorne una cadena de texto con la hora y partes del cuerpo colgadas en función de ese valor. Esta cadena de texto se asignará al TextArea correspondiente (setText()). Si no hay fallos, sólo aparecerá la hora, si el número de fallos es 6, el jugador ha perdido (FIN DEL JUEGO).
- ❖ Para mostrar la palabra con guiones, como lo tenemos (si hemos seguido un consejo anterior) es un array de chars, tenemos que convertirlo a String para poder asignarlo al JTextField correspondiente (setText()); esta conversión se puede hacer con: s = new String (array). Además, esa cadena nos permite averiguar también rápidamente si nos quedan guiones con s.indexOf(‘_’) (en vez de hacer un bucle recorriendo el array de chars), si el resultado es > 0 es que todavía hay guiones (según jugando); en caso contrario, es que el jugador ya ha adivinado la palabra completa (FIN DEL JUEGO).
- ❖ Para añadir mensajes al TextArea de la derecha, utilizar append(...); para limpiar el texto al comenzar una nueva partida, utilizar setText("") .