



TAREA PARA PROG08: Aplicaciones de las Estructuras de Almacenamiento CURSO 2023-24

ENUNCIADO

La tarea no se dará por entregada si el proyecto tiene errores de sintaxis, no compila, no se ejecuta correctamente, si lo entregado no se corresponde con lo pedido o si no se ha realizado un mínimo de la tarea, es decir, si está vacía o casi.

En esta tarea hay que entregar:

- Dos proyectos en NetBeans llamados
 - **Tarea_Prog08_Apellido1Apellido2Nombre_Matrices**
 - **Tarea_Prog08_Apellido1Apellido2Nombre_Muebles**
- Un documento **PDF** llamado: **Tarea_Prog08_Apellido1Apellido2Nombre.pdf**, en el que pondremos **captura de pantallas** en la que aparezca como **fondo** vuestra conexión **al Papas** los resultados de la ejecución del ejercicio (tal y como se vería con Netbeans), para así mostrar que funcionan (debe aparecer una prueba ejecutada del ejercicio).

ENUNCIADO: PARTE I: MATRICES

Se trata de realizar un proyecto Java en NetBeans llamado:

Tarea_Prog08_Apellido1Apellido2Nombre_Matrices.

Este proyecto realizará una serie de operaciones con matrices dimensionales cuadradas.

Crearemos dos paquetes:

- **modelo:** dentro de este paquete crearemos la clase **Matriz**
- **app:** dentro de este paquete crearemos la clase **AppMatrices** (será un JFrame)

Clase Matriz:

Realizaremos una clase llamada Matriz que tendrá:

- Atributo privado:
 - **dimensión** : almacenará la dimensión de la matriz (máximo 7)
- Constructores:
 - **public Matriz()**: crea la matriz de 7x7 y con todas las posiciones de la matriz a 0
 - **public Matriz(int limite)**: crea la matriz de dimensión: limite x limite (por ejemplo, si limite tiene 3: sería 3x3) y en las posiciones de la matriz, valores aleatorios de 2 dígitos que no sean 0 ($\text{Math.random() * 99 + 1}$)
- Métodos Públicos:
 - **public int getDimension()** : nos devuelve el valor del atributo dimensión de la tabla
 - **public void setDimension(int d)** : almacena en el atributo dimensión el valor d
 - **public int[][] getMatriz()** : devuelve la matriz entera
 - **public void setValor(int i, int j, int valor)**: introduce **valor** en la posición **i,j** de la matriz
 - **public int getValor(int i, int j)**: devuelve el valor de la posición i, j de la matriz
 - **public Float diagonal()**: calcula la suma de las posiciones de la diagonal principal de la matriz (posiciones i,i de todas las filas).
 - **public Matriz transpuesta()**: nos devuelve la matriz transpuesta, que consiste en pasar las filas a columnas. Todas las posiciones de la columna 1 pasan a las posiciones de la fila 1, posiciones de la columna 2 pasan a las posiciones de la fila 2,...
 - **public Matriz suma(Matriz m)**: nos devuelve la matriz sumando el valor de cada posición de la matriz con la misma posición de la matriz m.

- public Matriz **multiplicar**(Matriz m): nos devuelve la matriz multiplicando el valor de cada posición con la misma posición de la matriz m.
- public **sumaFila**(int i): nos devuelve la suma de todos los valores de la fila i.
- Public **sumaColumna**(int i): nos devuelve la suma de todos los valores de la columna i.

Formulario : AppMatrices

Realizaremos un formulario JFrame en NetBeans llamado: **AppMatrices**, con el siguiente formato o parecido (no tiene porqué tener el mismo diseño, pero si el mismo contenido):

- Declararemos dos matrices del tipo Matriz: **matrizA** y **matrizB**.
- **Métodos del JFrame:**
 - **private void llenarJTable(Matriz m, JTable jt):** con este método llenaremos el JTable jt con los valores que tiene la matriz m.
 - **private void limpiarJTable(JTable jt):** con este método limpiaremos el JTable tj, asignando la cadena vacía("") a todas las posiciones 7x7.
- **Elementos de JFrame:**
 - **Elemento JTextField para la dimensión de la matriz :** nombre: **txtDimension**: introduciremos la dimensión de la matriz, que no podrá estar entre 1 y 7 (ambos incluidos).
 - **Elemento JTable para el resultado:** Nombre: **JTableResultado**. es un JTable de dimensión 7 filas y 7 columnas, de tipo Integer, donde se visualizarán los resultados.
 - **Elemento: JTable para la MatrizA:** Nombre: **JTableMatrizA**: es un JTable de dimensión 7 filas y 7 columnas, de tipo Integer para visualizar el contenido de matrizA.
 - **Elemento: JTable para la MatrizB:** Nombre: **JTableMatrizB**: es un JTable de dimensión 7 filas y 7 columnas, de tipo Integer para visualizar el contenido de matrizB.
 - **Botón Limpiar A:** Nombres: **btnLimpiarA**. Al pulsar este botón, se limpia el jTableMatrizA, llamando al método **limpiarJTable** y pasándole como entrada el jTableMatrizA. Además todos los elementos de la matriz se pondrán a nulos.
 - **Botón Llenar A:** Nombre: **btnLlenarA**. Al pulsar este botón, se realizarán las acciones:
 - Si txtDimension es nulo, visualizaremos con JOptionPane el mensaje "Dimensión no tiene valor" y llevaremos el foco a txtDimension
 - Si txtDimension tiene un valor <1 y >7, visualizaremos con JOptionPane el mensaje "La dimensión debe de ser entre 1 y 7" y llevaremos el foco a txtDimension
 - Se inicializará la matrizA con el constructor Matriz(txtDimension)
 - Se llenará el JTable jTableMatrizA con los valores de la matrizA, llamando al método llamado **llenarJTable** al que le pasaremos como entrada la matriz A y la jTableMatrizA

- **Botón: Diagonal A:** Nombre: **btnDiagonalA**: al pulsar este botón
 - Si la matrizA está a nula, visualizaremos con JOptionPane el mensaje “La matriz A no tiene datos” y llevaremos el foco a txtDimension
 - Se calculará la diagonal principal de la matrizA, llamando al método diagonal de la clase Matriz.
- **Botón Transpuesta A:** Nombre: **btnTranspuestaA**: al pulsar este botón:
 - Si la matrizA está a nula, visualizaremos con JOptionPane el mensaje “La matriz A no tiene datos” y llevaremos el foco a txtDimension
 - Si ambas matrices están a nulo visualizaremos con JOptionPane el mensaje “Las matrices no tienen datos”
 - Se calculará la matriz transpuesta de la matrizA llamando al método **transpuesta** de la clase Matriz después llamaremos al método **llenarJTable** para visualizar el resultado.
- **Botón A+B:** Nombre: **btnSuma**: al pulsar este botón:
 - Si matrizA o Matriz B son nulas, visualizaremos con JOptionPane el mensaje “Las matrices no tienen valores”
 - Si la dimensión de la matrizA y de la matrizB no son iguales, visualizaremos con JOptionPane el mensaje “Las matrices no tienen la misma dimensión”
 - Se calculará la suma de ambas matrices : matrizA y matrizB, llamando al método sumar de la clase Matriz, después llamaremos al método **llenarJTable** para visualizar el resultado de la suma.
- **Botón A*B:** Nombre: **btnMultiplicar**: al pulsar este botón:
 - Si matrizA o Matriz B son nulas, visualizaremos con JOptionPane el mensaje “Las matrices no tienen valores”
 - Si la dimensión de la matrizA y de la matrizB no son iguales, visualizaremos con JOptionPane el mensaje “Las matrices no tienen la misma dimensión”
 - Se calculará la multiplicación de ambas matrices : matrizA y matrizB, llamando al método multiplicar de la clase Matriz, después llamaremos al método **llenarJTable** para visualizar el resultado de la multiplicación.
- **Botón Limpiar B, Botón Llenar B, Botón: Diagonal B, Botón Transpuesta B:** el funcionamiento de estos botones son iguales que para la matriz A, pero con la matriz B.
- Tendremos la posibilidad de sumar las filas o las columnas de una de las matrices. Para hacerlo tendremos tres JComboBox:
 - **Elemento JComboBox:** nombre: **cmbElegirMatriz**: con él podemos elegir la matriz de la que queremos sumar filas o columnas, podremos elegir Matriz A o Matriz B.
 - **Elemento JComboBox:** nombre: **cmbFilaColumna**: para elegir si queremos sumar las filas o las columnas
 - **Elemento JComboBox:** nombre: **cmbNroFilaColumna**: para elegir el nº de fila o columna: 1,2, 3, 4, 5, 6 o 7. **(Si elegimos una fila o columna mayor que la dimensión saldrá un mensaje de error)**
- Una vez seleccionado en los combos, podemos pulsar en el botón llamado: **btnVisualizarSuma** y visualizaremos la suma de la fila o columna seleccionada de la matriz seleccionada. Para ello utilizaremos los métodos **sumaFila(i)** y **sumaColumna(i)** de la clase Matriz. Hay que controlar que las matrices no estén vacías, en cuyo caso visualizaremos un mensaje con un mensaje con JOptionPane.

ENUNCIADO : PARTE II: LISTA DE MUEBLES

Se trata de realizar un proyecto Java en NetBeans llamado:

Tarea_Prog08_Apellido1Apellido2Nombre_Muebles

El proyecto gestionará la venta de una serie de muebles. Los datos que necesitamos almacenar de los muebles son:

- **Código Mueble:** alfanumérico (patrón **patronCodigo**)
- **Descripción del Mueble:** alfanumérico (patrón: **patronAlfanumerico**)
- **Precio Unitario:** real (patrón **patronNumeroReal**)
- **Unidades Almacen:** entero (patrón: **patronNumeroEntero**)
- **unidadesMinimas:** entero (patrón: **patronNumeroEntero**)
- **tipoMueble:** 1 carácter (Valores corretos:H: Hogar/D:Despacho/C:Colegios)

En dicho proyecto crearemos dos paquetes:

- **app:** en el que crearemos un formulario JFrame llamado : **JFGestionMuebles.java**
- **modelos:** dentro de él crearemos una clase llamada: **Mueble.java**

Clase: Mueble.java:

Será una clase que implente la **interfaz: Comparable**

public class Mueble implements Comparable<Mueble>

Y que reescribirá el método **compareTo** y **equals** para que podamos ordenar y comparar por el código del mueble.

Además de los métodos anteriores, tendrá los métodos getters y setters para los atributos y dos constructores, uno sin parámetros y otro con parámetros.

Habrà un método llamado **getTotalAlmacen** que será el resultado de multiplicar las unidades en almacén por el precio unitario del objeto:

public float getTotalAlmacen ()

Formulario: AppGestionMuebles.java

En este formulario (JFrame) gestionaremos la venta de los muebles, declararemos una lista (ArrayList) llamada **listaMuebles** en la que iremos almacenando los objetos **Mueble** que introduciremos desde el formulario, con la condición de que el código del mueble no puede estar repetido.

El formulario tendrá el siguiente formato (o parecido):

| CODIGO MUEBLE | DESCRIPCIÓN | PRECIO UNITARIO | UNIDADES ALMACÉN | UNIDADES MÍNIMAS | TOTAL ALMACEN |
|---------------|-------------|-----------------|------------------|------------------|---------------|
|---------------|-------------|-----------------|------------------|------------------|---------------|

Componentes del **AppGestionMuebles:**

- Un jTextField para cada uno de los atributos del objeto Mueble
- Un combo para el tipoMueble.
- Botones: Añadir, Buscar, Eliminar, Modificar, Limpiar Datos, Limpiar Todos, Visualizar Todos y Salir.
- Un jTable llamado **jTableMuebles** en el que iremos visualizando los muebles introducidos.

Crearemos los siguientes **patrones** (Públicos):

- **patronCodigo:** que nos sirva para controlar que el código sea correcto. Éste es correcto si tiene 1 o 2 letras, seguido de un guión, después 4 dígitos.
- **patronAlfanumérico:** que nos sirva para controlar que solo le tecleen letras, números y espacios en blanco.
- **patronAlfabetico:** que nos sirva para controlar que solo le tecleen letras y espacios en blanco.
- **patronNumeroReal:** que nos sirva para controlar que solo le tecleen: dígitos enteros, como mínimo 1, seguido de un punto y después dígitos decimales, como mínimo 1.
- **patronNumeroEntero:** que nos sirva para controlar que solo le tecleen: dígitos enteros, como mínimo 1.

Funcionamiento de los jTextField:

- **txtCodigoMueble:** programaremos el evento actionPerformed, en el que controlaremos:
 - Si el txtCodigoMueble está vacío, visualizaremos un mensaje con **JOptionPane.showMessageDialog** y volveremos el foco al código
 - Si no coincide con el patronCodigo, visualizaremos un mensaje con **JOptionPane.showMessageDialog** y volveremos el foco al código
- **txtDescripcion:** programaremos el evento actionPerformed, en el que controlaremos:
 - Controlaremos que no esté vacío, si lo está, visualizaremos un mensaje con **JOptionPane.showMessageDialog** y volveremos el foco al txtDescripcion.
 - Si la descripción no coincide con el patronAlfanumerico, pasaremos el foco a txtDescripcion.
- **Para txtPrecioUnitario, txtUnidadesAlmacen, txtUnidadesMinimas:** se realizará igual que la descripción, comparando cada uno con su patrón
- Las unidades en almacén deben ser mayores que las unidades mínimas.
- Cuando tecleemos el valor en unidadesAlmacen (comprobaremos que precio unitario también tenga valor, si no lo tiene visualizaremos un mensaje), llamaremos al método **getTotalAlmacen** y lo devuelto se lo pasaremos al componente **txtTotalAlmacen** (que no podrá ser editable, posicionándonos en las propiedades, deseleccionaremos editable).

Funcionamiento de los botones:

- **Botón Añadir: btnInsertar :** programaremos el evento actionPerformed:
 - Comprobaremos que haya algún campo nulo, si es así, visualizaremos un mensaje y no insertaremos
 - Comprobaremos nuevamente que todos los patrones se cumplen, si alguno no se cumple visualizaremos un mensaje, no insertaremos y pondremos el foco en el que no lo cumpla.
 - Buscaremos el código del mueble en el ArrayList **listaMuebles** (método: **buscarMueble**).
 - Si es nulo (no existe el mueble), visualizaremos un mensaje y llevaremos el foco al txtCodigoMueble.
 - Si existe, llamaremos al método **llenarMueble**, que pasará los valores de los jTextField a un objeto Mueble y lo devolverá como salida.
 - Añadir el objeto mueble a la lista
 - Ordenamos la lista
 - Visualizaremos de nuevo el JTable con la lista ordenada.
 - Visualizar mensaje: MUEBLE GRABADO
- **Botón Buscar: btnBuscar:** programaremos el evento actionPerformed:
 - Si el txtCodigoMueble está vacío, visualizaremos un mensaje con **JOptionPane.showMessageDialog** y volveremos el foco a txtCodigoMueble
 - Buscaremos el código del mueble (**buscarMueble**).
 - Si es nulo (no existe), visualizaremos un mensaje y llevaremos el foco a txtCodigoMueble.
 - Si no es nulo (existe), realizaremos un método llamado **visualizarMueble**, que pasará los datos del objeto devuelto por buscarMueble a los jTextField y al combo del tipo de mueble.
- **Botón Eliminar: btnEliminar:** programaremos el evento actionPerformed, en el que controlaremos:
 - Si el txtCodigoMueble está vacío, visualizaremos un mensaje con **JOptionPane.showMessageDialog** y volveremos el foco a txtCodigoMueble
 - Buscaremos el código del mueble (**buscarMueble**).
 - Si es nulo (no existe), visualizaremos un mensaje NO EXISTE ESE CÓDIGO MUEBLE llevaremos el foco a txtCodigoMueble.
 - Si no es nulo (existe), realizaremos una llamada al método **visualizarMueble**, que pasará los datos del objeto devuelto por buscarMueble a los jTextField. Mediante: **JOptionPane.showConfirmDialog**, se preguntará si se Desea eliminar el mueble.
 - Si la respuesta es **JOptionPane.YES_OPTION**:
 - Buscaremos de nuevo el registro
 - Lo eliminamos de la lista

- La ordenamos
- Visualizaremos de nuevo el jTable.
- **Botón Modificar: btnModificar:** programaremos el evento actionPerformed, en el que controlaremos:
 - El proceso normal es que primero busquemos el mueble(botón buscar), modifiquemos lo que se desee y después pulsar el botón modificar.
 - Mediante un JOptionPane.showConfirmDialog, se preguntará si se Desea modificar el mueble.
 - Si se responde que si:
 - Que el codigoMueble exista (por si se ha modificado)
 - Controlaremos que no haya ninguno a nulo
 - Que todos coincidan con su patrón
 - Actualizaremos la lista con los valores que haya en los jTextField
 - Ordenaremos la lista
 - Visualizaremos de nuevo el jTable.
- **Botón LimpiarTodos: btnLimpiarTodos:** limpia el jTable entero.
- **Botón VisualizarTodos: btnVisualizarTodos:** ordena la lista y la visualiza.
- **Botón LimpiarDatos: btnLimpiarDatos:** limpia todos los jTextField (asignándoles la cadena vacía "" o "0").

Funcionamiento de jTableMuebles:

Para ver el funcionamiento, mira el ejemplo de Gestión de Productos, dentro del proyecto:

Prog08_EjemplosInicialesProductos, la clase: **JFrameGestionProductosJTable.java** que os he dejado en la plataforma en: UT8 Enunciados y Ejemplos

Recursos necesarios para realizar la Tarea.

- Ordenador personal.
- JDK y JRE de Java SE.
- Entorno de desarrollo NetBeans con las funcionalidades necesarias para desarrollar y emular midlets.
- Para ver el funcionamiento, mirad en el ejemplo de Gestión de Productos con jTable, que os he dejado en la plataforma Ejercicios resueltos de la unidad: Prog08_Ejercicios y Enunciados.

Criterios de puntuación. Total 10 puntos.

Para poder empezar a aplicar estos criterios es necesario que **LOS PROYECTOS NO TENGA ERRORES DE SINTAXIS, COMPILE Y SE EJECUTE CORRECTAMENTE.**

En caso de no hacerlo, la puntuación será directamente de **1,00**.

1. Parte I: Proyecto Matrices: 4 puntos

2. Parte II: Proyecto Listas: 6 puntos

En la puntuación total se podrá descontar por lo siguiente: **No** se han incluido **comentarios** describiendo el funcionamiento de todos los métodos y atributos en ambas clases, como se ha pedido en el enunciado. **No** se ha entregado el **informe** explicativo o se trata de un informe explicativo insuficiente

Qué hay que entregar

- Proyectos NetBeans compridos funcionando y **sin errores**.
Tarea_Prog08_Apellido1Apellido2Nombre_Matrices.rar
Tarea_Prog08_Apellido1Apellido2Nombre_Muebles.rar
- Además de los proyectos NetBeans, se deberá entregar un **informe** en el que se podrá como fondo vuestra conexión al Papas y en primer plano la ejecución de cada proyecto.
Tarea_Prog08_Apellido1Apellido2Nombre.pdf.
- Se **comprimirán tres archivos en uno solo:**
Tarea_Prog08_Apellido1Apellido2Nombre.rar