IMPORTANTE: El examen lo haréis en un proyecto con **vuestro nombre** y cada ejercicio en un paquete distinto con nombre: **ejercicio1 y ejercicio2** respectivamente. **Se entregará el proyecto completo y cada programa debe compilar.**
Para que pueda ser corregido el examen, se tiene que cumplir lo descrito anteriormente, en caso contrario la nota en el ejercicio será de cero.

Además del correcto funcionamiento se valorará el estilo, claridad, optimización y organización del código como parte de la nota asignada en cada apartado. Deben cumplirse las bases de la POO.

Distribución de la nota:

1.- 2,5 puntos

2.- 7,5 puntos

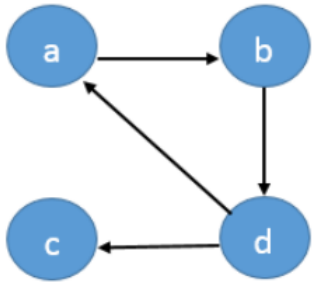
TOTAL: 10 puntos

1.- Se pide crear una clase denominada **MatrizAdyacencia** que nos permita la generación de matrices cuadradas de tamaño N.

El **constructor** recibirá el tamaño de la matriz y creará la matriz vacía. (***0,25 puntos***).

Crearás también un método llamado **iniciarMatriz**() para rellenarla de forma aleatoria con valores comprendidos entre 0 y 1. (***0,75 puntos***).

Una matriz cuadrada puede representar un grafo dirigido similar al siguiente:



	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

Como podemos observar en ambas imágenes, desde a podemos ir a b (valor 1 en la tabla de la derecha). Desde b a d , … Para ver los caminos que existen en una matriz, harás un método llamado **caminos()** (***1,5 puntos***), que mostrará en pantalla los caminos posibles de cada uno de sus vértices. Para poder mostrar los nombres de los vértices supón que habrá un máximo de 10 vértices en un grafo (a, b, c…, j). Puedes usar un interface para almacenar esa letras en forma de array unidimensional.

Ejemplo de **salida** para el grafo del dibujo donde /--> significa que no hay camino y ---> que sí lo hay**:**

Caminos posibles:

a /--> a
a ---> b
a /--> c
a /--> d

b /--> a
b /--> b
b /--> c
b ---> d

c /--> a
c /--> b
c /--> c
c /--> d

d ---> a
d /--> b
d ---> c
d /--> d

Nota: No se puede utilizar la palabra reservada **static** en ningún caso.

2.- Se pide desarrollar un programa para realizar las tareas más habituales en un almacén que tiene artículos que podemos clasificar de **hardware** y de **software**.

Ambos tipos de artículos tendrán:

- Un **código** de tipo cadena de caracteres, con justo 3 caracteres siempre. Da igual que sean numéricos o alfabéticos. Si no se cumple esto, no se da de alta el artículo y se vuelve a mostrar el menú. No tienes que controlar si el código se repite para más de un artículo.
- Un **nombre** (alfanumérico).
- Un **precio**.
- La **cantidad** en stock del artículo.

Para los artículos **Software**, se tendrá además que guardar el **tipo** de artículo software de qué se trata, por ejemplo:

"Sistema Operativo", "Juegos", "Antivirus", …

Para los artículos **Hardware**, se tendrá que guardar si se trata de un **periférico** o no (en un atributo de tipo booleano, indica al usuario que introduzca S o N y valida que lo introducido es correcto.

El programa empezará presentando el siguiente **menú** en el que tendrás que desarrollar cada una de sus opciones de la forma en qué se explica más abajo:

1. Introducir artículo de Software.
2. Introducir artículo de Hardware.
3. Mostrar todos los artículos ordenados descendentemente por precio.
4. Aumentar el precio de todos los artículos Software.

5. Borrar un artículo.

6. Salir.

Elija opción:

Para el desarrollo del proyecto debes crear **3 paquetes**:

Paquete *principal* (1,25 puntos):

El menú lo desarrollarás en un programa fuente (**Main.java**) que contendrá el método **main().** En el fichero Main.java **no tendrás más métodos que los necesarios para la presentación del menú y llamadas a los métodos de las demás clases**.

Acciones asociadas a las opciones de menú:

Opción 1: Se pedirán los datos para dar de alta un artículo de Software y se dará de alta.

Opción 2: Se pedirán los datos para dar de alta un artículo de Hardware y se dará de alta.

Opción 3: Se mostrarán los datos de cada uno de los artículos que hayamos almacenado hasta el momento en nuestro almacén (ordenados descendentemente por precio), especificando de qué tipo son (uno por línea). Al final, una vez mostrados todos se mostrará el valor total de los de tipo Software por un lado y los de tipo Hardware por otro, que hay en el almacén.

Opción 4: Se aumentará el precio de los artículos de tipo Software en un porcentaje que se pedirá al usuario.

Opción 5: Mostrará los datos de todos los artículos almacenados (ahora sin suma de importes) y pedirá al usuario el código del artículo que se desea borrar, que se eliminará de nuestro almacén en ese momento (si hay varios, se borran todos).

Opción 6: Terminará el programa cuando se elija esta opción.

Paquete *artículos* (4,75 puntos):

Para desarrollar las opciones del menú tendrás que usar una clase a la que llamarás **Almacen.java** en este paquete.

La clase **Almacen.java** contendrá la estructura de datos (un **ArrayList**) donde se guardan **todos** los artículos del almacén en memoria y todos los métodos necesarios para desarrollar las distintas opciones del menú sobre ese ArrayList. Vamos a suponer que no vamos a necesitar usar más que un almacén, por lo que declararemos todos los elementos de esta clase como estáticos. **(2,75 puntos)**

El **resto** de los ficheros fuente (.java) que desarrollarás en este paquete, serán para representar las clases necesarias para representar los distintos artículos, siempre **de la forma más adecuada en el**

paradigma de la Orientación a Objetos. Todos los atributos deben ser declarados como privados. **(2 puntos)**

Paquete lectura (1,5 puntos):

En este paquete crearás un fichero llamado **Lector.java**, que contendrá los métodos (**estáticos**) que recibiendo un objeto de la clase Scanner, **devolverán un objeto** del tipo adecuado a la opción de menú seleccionada.

Nota: Debes indicar con mensajes todas las posibilidades de error que se puedan dar en la aplicación.