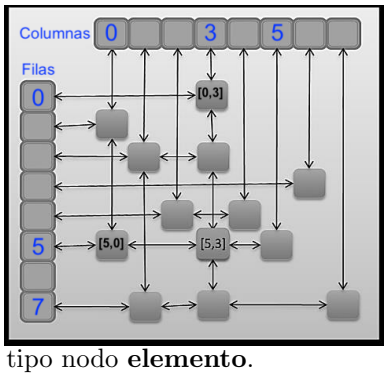
Tecnología de la Programación **18 de enero de 2022**
Duración: 2h:30' Evaluación Final.

NOTA IMPORTANTE: Aquellos ejercicios, que, a pesar de estar resueltos correctamente, no estén razonados por el alumno, se considerarán no válidos. Comenta el código. Controla los parámetros de entrada **sin usar assert**. Justifica, en su caso, el valor de retorno de los métodos.

Apellidos y Nombre: .....



Ejercicio 1 (6 puntos + 1 punto extra) En **álgebra lineal numérica** una matriz dispersa o matriz rala o matriz hueca es una matriz de **gran tamaño** en la que la mayor parte de sus elementos son **cero**. Con matrices de gran tamaño los métodos tradicionales para almacenar la matriz en la memoria de una computadora o para la resolución de sistemas de ecuaciones lineales necesitan una gran cantidad de memoria y de tiempo de proceso. Se hace necesario, pues, diseñar tipos de datos y algoritmos específicos para estos fines cuando las matrices son dispersas.

En este ejercicio tendrás que implementar este TDA-Matriz-Dispersa usando **una simplificación** de la estructura de la figura adjunta.

- EL TDA consta de nodos. Estos a su vez puede ser de tipo nodo **cabecera** o de tipo nodo **elemento**.
- Un nodo cabecera consta de los siguientes campos:
  - **index:** indica un índice fila/columna. Los índice empiezan en cero.
    En la figura: Filas muestra los nodos fila (del 0 al 7). Columnas muestra los nodos columnas (del 0 al 7).
  - **down:** o es None, o apunta a un nodo cabecera (si es un nodo fila) o a un nodo elemento (si es un nodo columna).
  - **right:** o es None, o apunta a un nodo cabecera (si es un nodo columna) o a un nodo elemento (si es un nodo fila).

  Ejemplo: los nodos cabecera para el elemento [5, 3] (ver figura) son:
  - Nodo cabecera fila: (index=5, down=nodo cabecera fila siguiente con index 6, right=nodo elemento [5, 0])
  - Nodo cabecera columna: (index=3, down=nodo elemento [0, 3], right=nodo cabecera derecha con index 4)
- Un nodo elemento consta de los siguientes campos:
  - **indices:** una tupla formado por un índice fila y un índice columna.
  - **value:** un valor numérico.
  - **down:** apunta a un nodo elemento o es None.
  - **right:** apunta a un nodo elemento o es None.

  Ejemplo: el nodo elemento para el elemento [5, 3] (ver figura) son:
  - indices=(5, 3), down=nodo elemento [7,3], right=nodo elemento [5, 5], value=puede ser cualquier valor.

  Notar que en la figura hay más atributos, pero se insiste en que solo se implementará **una simplificación** de la figura.

La implementación se realizará en varias partes (módulos) y cada paso se desarrollará con el inicio de **un folio diferente**. **No** está permitido el uso de **assert** en todo el programa.

Parte 1: (2 puntos) Especifica cómo se deben definir las **clases** `Nodo`, `NodoCabecera` y `NodoElemento`; así como sus constructores. Recuerde redefinir estas clases, si fuera necesario, añadiendo cuantos métodos sean necesarios y en función de lo que se pide en las partes siguientes.
NO escriba la interface Getter/Setter (se supone que ya le viene dada). Se asume, que para cada atributo de la forma `._attributo` que indiques en `.__init__()` existirá las correspondientes propiedades `attributo`.

Parte 2: (2 punto) Especifica también las **clases** que usas para lanzar tus propios **errores**, y que se indican en las demás partes. Recuerde redefinir estas clases, si fuera necesario, añadiendo cuantos métodos y clases sean necesarios y en función de lo que se pide en las demás partes.

Parte 3: (2 puntos) Implementa la clase `Matriz`, **usando solo los *nodos elementos* y los *nodos cabecera fila*** y con los siguientes operadores:

- `Matriz(filas, columnas)`: Construye una matriz vacía (sin nodos elementos) para el número de filas y columnas indicadas.
- `.add(i, j, valor)`: Añade el nodo elemento con **indices=(i, j)** si dicho nodo no existe y se le asignará **.value=valor**. De existir el nodo, se sustituirá dicho valor. Lanzará un error propio si los índices están fuera de rango.
  - Implementa explícitamente la inserción de un nodo en la lista enlazada, **sin recurrir a delegaciones de otros TDAs**.
  - Justifique claramente las delegaciones que hace esta clase a las clase de la jerarquía de nodos.
- ⋆ Recuerda definir claramente los tipos de errores que generarás en esta parte (leer partes anteriores).

Parte 4: (1 punto Extra) Construye un programa principal que utilice una interface de texto (en consola) con las siguientes opciones: `0. Salir; 1. Crear matriz; 2. Añadir elemento.`

- El programa principal deberá de controlar los errores.
- Implemente la arquitectura MVC para la máxima nota.

Ejercicio 2 (4 puntos + 1 punto extra)
Implementa las clases (atributos y métodos) que se piden en cada apartado, indicando de forma clara, y si fuera necesario, las clases abstractas, métodos abstractos, tipo de variables (de instancia o de clase), tipo de métodos (de instancia, de clase o estáticos).

1. **(0.5 puntos)** Todo guerrero tiene un nombre, pero no se pueden construir instancias de guerreros. Existen dos tipos de guerreros, los griegos y los troyanos, que sí pueden construirse. Construya las clases `Guerrero`, `Griego`, `Troyano`.

2. **(1 punto)** Escribe la clase `Caballo`. Consta de dos atributos de instancia. El primero hace referencia a su capacidad y el segundo hace referencia a una pila (stack) de ocupantes (que serán guerreros). Consta también del siguiente método:

   `.buscar(self, nombre: str)->bool` retorna `True` si se encontró a un guerrero con el nombre dado o `False` en otro caso.

   ⋆ Si necesitara los métodos de una pila, dispone de los métodos `.push()`, `.pop()`, ... (consulte las chuletas oficiales).

   ⋆ NO implemente ningún método del TDA Pila.

3. **(1.5 puntos)** Existen dos tipos de caballos, `CaballoMamifero` y `CaballoTroya`, y que debe de construir:

   - Cualquier caballo mamífero tiene siempre una capacidad de hasta 2 jinetes y se construyen sin jinetes que lo monten.
   - Un caballo de Troya se construye suministrando su capacidad para n-jinetes junto con una lista de guerreros. Deberá de tener en cuenta que (a) la capacidad n no podrá superar una capacidad máxima fija establecidad para cualquier caballo de Troya y (b) de la lista de guerreros solo pueden entrar en la pila los guerreros griegos - se descarta cualquier troyano.

4. **(1 punto)** Se te suministra la interface `IMontar` con los métodos abstractos `montar()` y `desmontar()` y cuyos parámetros y sus tipos debes deducir de lo que se te pide.

   - El método `desmontar` debe quitar al último ocupante que subió al caballo.
   - El método `montar` debe añadir un ocupante al caballo, siempre que no se supere su capacidad de ocupantes. Además deberá de tener en cuenta que en un caballo de Troya su parámetro de entrada será un griego (no se admiten troyanos) y en un caballo mamífero se admite cualquier guerrero (no importa si se mezclan troyanos con griegos).

   ¿Cómo y dónde deben implementarse estos métodos en la jerarquía de clases de todos los caballos teniendo en cuenta lo que se indica y sin que ninguno de los métodos use la sentencia **if** o **isinstance()**? ¿Cuál debería ser la signatura - parámetros, retorno y sus tipos - sin que se viole el principio de Liskov?

   En este apartado no copie todo lo que ya haya escrito sobre las clases de los apartados anteriores, **solo debe indicar** el nombre de la clase y la inclusión de lo que ahora realice.

5. **(1 punto extra)** Un `Ejercito` consta de un atributo que guarda una lista de guerreros (pueden ser griegos o troyanos). Todo ejército se **construye** invocando a su método `.load(fichero: str) -> List[Guerrero]` que consiste en leer el contenido de un fichero de texto. Cada línea del fichero tiene la secuencia `Griego nombreGuerreroGriego` o la secuencia `Troyano nombreGuerreroTroyano`, donde `nombreGuerreroXXX` es una secuencia de caracteres sin espacios. El método retorna una lista de guerreros, que puede ser vacía. Implementa dicho método controlando la excepción `FileNotFoundError`.

   ¿Cómo es el constructor de esta clase y su método `.load()`?