TAP 2020-21. Laboratorio.

2. Matrices

Si queremos resolver problemas de álgebra matricial con Java, lo razonable es buscar en la web alguna librería que lo permita.

Como el objetivo es aprender a desarrollar con Java, lo que haremos en el laboratorio es escribir parte de lo necesario para trabajar con matrices. NO HAY QUE MIRAR PRIMERO QUÉ CONTIENEN LAS LIBRERÍAS DISPONIBLES EN INTERNET, solamente se trata de practicar los elementos básicos del lenguaje, no de dar una solución óptima.

¿Y cuál es el elemento central para trabajar con matrices?... pues las matrices en sí mismas… así que se trata de escribir la clase "`Matrix`". Para ello ha de darse respuesta a las siguientes preguntas:

a) ¿Extenderá alguna clase existente? Existe "`Number`" y una matriz podría quizás considerarse un tipo de número con el que realizar operaciones aritméticas como con cualquier otro. Así que se ha de comenzar por ver qué aporta "`Number`" y valorar si es algo que nuestras Matrices deban asumir (usar "`extends`").

b) ¿Cómo se almacenarán los valores de la matriz? Aquí no vamos a complicarnos en absoluto, porque por el momento la solución de que disponemos es la de los arrays bidimensionales, y por plantear un caso concreto los definiremos de "doubles" (`double[][]`).

c) ¿Cómo se construirán nuestras matrices? Tenemos varias opciones:

   i) Vacías (constructor al que se le indican sólo las dimensiones). Después podrán irse asignando los valores uno a uno.

   ii) Leyendo los valores de un fichero en el disco del ordenador. Esto resultará muy útil, pero por el momento no hemos visto cómo hacerlo con Java. (así que no hacemos nada al respecto)

   iii) también es razonable que podamos construir algunas matrices "estándares" como por ejemplo la identidad, una con valores aleatorios en un rango, o una diagonal aportando sólo los elementos de dicha diagonal, etc. Es recomendable que estas matrices se obtengan a partir de métodos de la clase (no de constructores), de modo que su nombre indique el tipo de matriz que queremos, como por ejemplo "`Matrix.getIdentity(int dim)`" (la llamada al constructor se hace dentro del método y se devuelve la matriz construida e inicializada convenientemente),

d) Tendremos que atender a la herencia de Object. En particular un toString() nos vendrá bien para comprobar cómo funciona lo que hagamos.

e) ¿Qué podremos hacer con estas matrices? Pues para esta práctica haremos lo que no resulte demasiado complejo[1] de entre todo lo que nos parezca pertinente:

   i) Las operaciones aritméticas básicas[2]: sumar, restar, multiplicar y dividir. El método para la división lo dejaremos vacío (al menos inicialmente) porque es algo complejo.

   ii) Otras operaciones propias de las matrices: trasponer, obtener la traza, el determinante, las operaciones aritméticas con un número real, etc.

---

1 La posibilidad de invertir matrices y por tanto de implementar la división, así como otras operaciones (calcular valores y vectores propios, etc.), ya está en nuestra mano, pero la dificultad está en el algoritmo. Puede ser un buen ejercicio para casa localizar información de cómo hacerlo e implementarlo.

2 Tanto para estas operaciones como para cualquier otro caso aplicable, las situaciones de excepción/error se han de tratar con el mecanismo ya visto.

iii) Test de condiciones: si es cuadrada, si es diagonal, si es simétrica, si es la identidad, si es escalar, si es idempotente ($M=M^2=M^3=\ldots$),…

iv) Los métodos de acceso a elementos (getters y setters de la posición (i,j)). Ojo a que (como tantas veces) pueden producirse errores si se aportan índices incorrectos.

En muchos de los métodos indicados la duda es: ¿Hacemos métodos que reciben todos los operandos de la operación o preferimos asumir el "`this`" como uno de ellos? Para no entrar en valoraciones de una u otra aproximación elegimos una sin más: no usaremos el "`this`", plantearemos los métodos como operadores que reciben en sus parámetros todos los operandos sobre los que actúan.

Una vez hecho esto haremos una clase de prueba que, en su main, compruebe el funcionamiento de todas las capacidades dadas a nuestra clase "Matrix".

Esta clase podrá mejorar más delante conforme aprendamos más Java.

- Podremos leer/escribir matrices en ficheros o "recogerlas/enviarlas" de/a otros "origenes/destinos".
- Podremos tratar con matrices que no sean necesariamente de doubles.

La estructura planteada puede complicarse para dar más versatilidad a la solución, como por ejemplo planteando diferentes tipos de representación interna en función del tipo de la matriz en lo que se conoce como matrices "densas" y "dispersas" (dense/sparse), que serían dos subclases de una abstracta, o bien dos implementaciones de un interfaz.
