Universidad Autónoma de Yucatán

Facultad de Matemáticas

Construcción de Software

Proyecto Final

Profesor: Edwin Jesús León Bojórquez

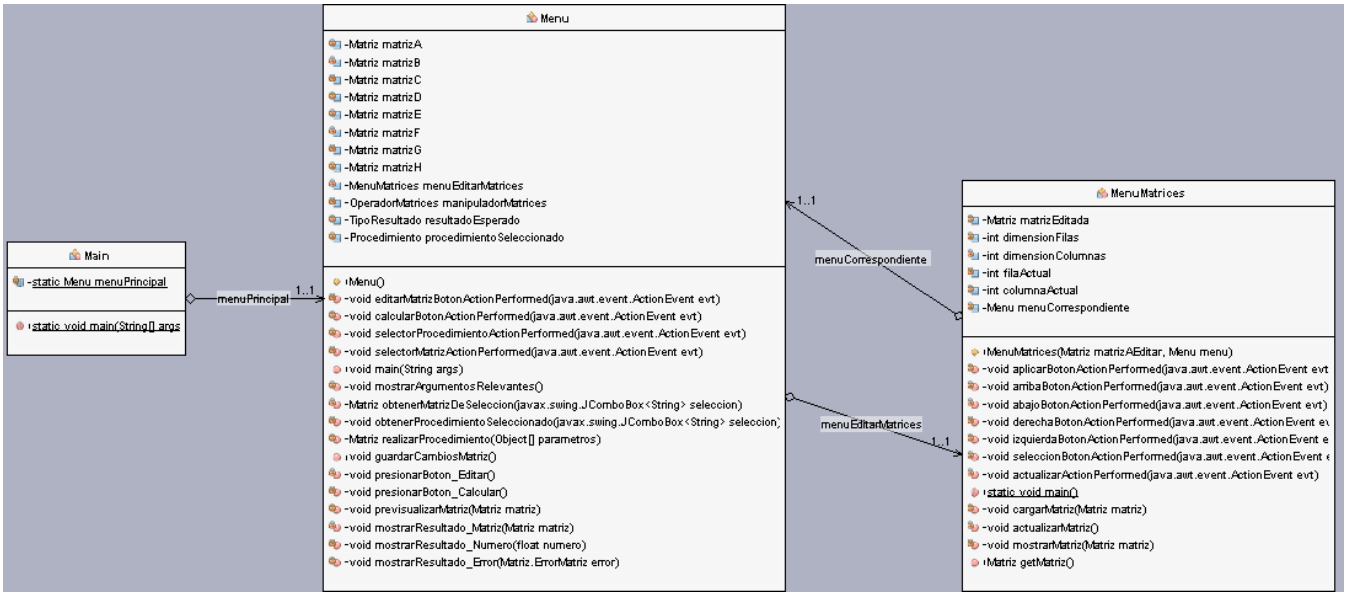
Alumno: Amaury Morales

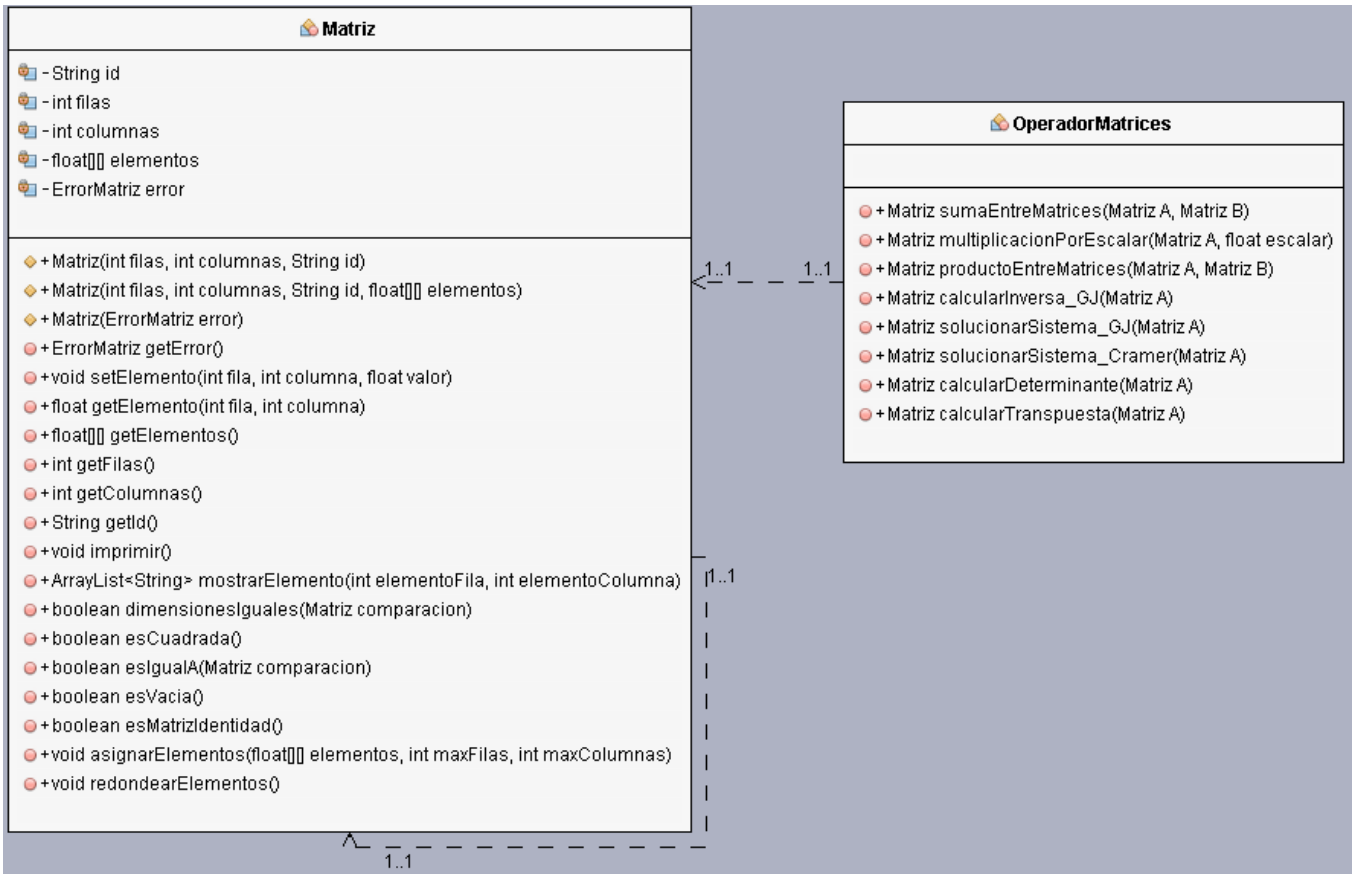
05/12/2021

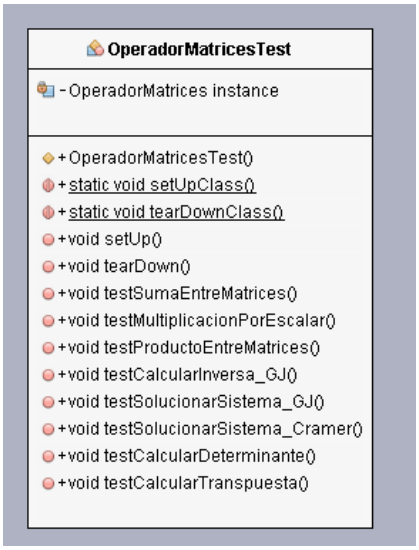
Contenidos

Construcción del código

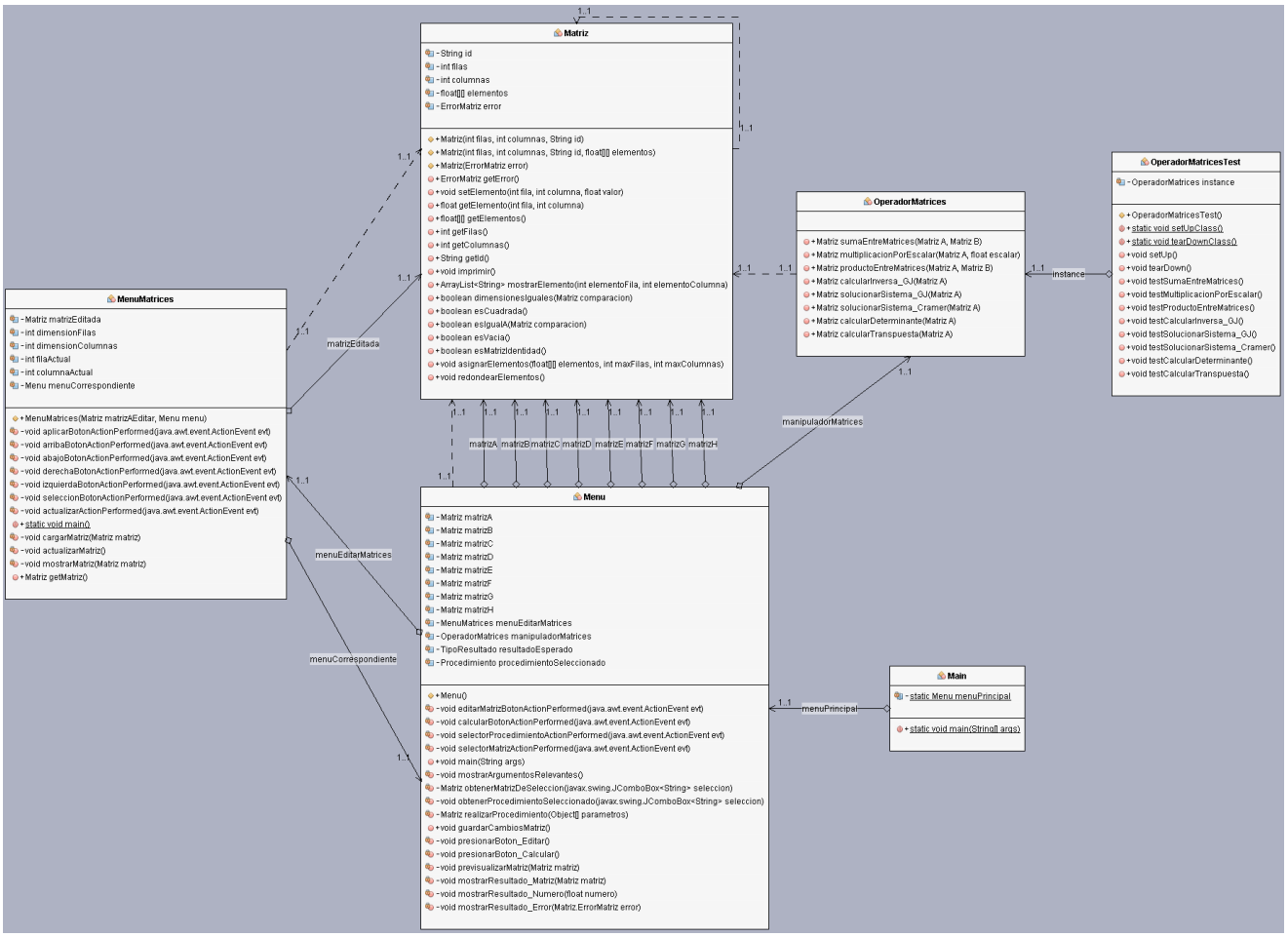
Modelado del dominio del problema







UML Completo



Checklist de las prácticas de construcción utilizadas

El checklist se dividirá en varias partes, por lo que son múltiples checklist, una por cada tipo de técnica.

Técnicas de construcción de variables y tipos de datos fundamentales

No	Practica	Uso	Comentario
1	Does each routine check input parameters for validity?	X	Solo cuando es necesario
2	Does the code declare variables closet to where they're first used?	X	Solo cuando es posible
3	Does the code initialize variables as they're declared, if possible?	X	Solo cuando es posible
4	Does the code initialize variables close to where they're first used, if it isn't possible to declare an initialize them at the same time?	X	
5	Are counters and accumulators initialized properly and, if necessary, reinitialized each time they are used?	X	
6	Are variables reinitialized properly in code that's executed repeatedly?	X	
7	Does the code compile with no warnings from the compiler (and have you turne don all the available warnings?)	X	
8	If your language uses implicit declarations, have you compensated for the problems they cause?		No se utilizan declaraciones implícitas por la naturaleza de Java
9	Do all variables have the smallest scope possible?		Probablemente no, se hizo un esfuerzo para optimizar el scope de las variables, pero como todo en este mundo, se puede mejorar
10	Are references to variables as close together as possible, both from each reference to a variable to the next reference and in total live time?	X	Se intento juntarlas, pero en algunos pasos no se puede. Se podría mejorar
11	Do control structures correspond to the data types?	X	
12	Are all the declared variables being used?	X	
13	Are all variables bound at appropriate times-that is, are you striking a conscious balance between the flexibility of late binding and the increased complexity associated with late binding?	X	Yo considero que si hay un balance decente.
14	Does each variable have one and only one purpose?	X	
15	Is each variable meaning explicit, with no hidden meanings?	X	

16	Does the name fully and accurately describe what the variable represents?	X	
17	Does the name refer to the real-world problem rather than to the program-ming-language solution?	X	
18	Is the name long enough that you don't have to puzzle it out?	X	
19	Are computed-value qualifiers, if any, at the end of the name?		No aplicable
20	Does the name use Count or Index instead of Num?	X	
21	Are loop index names meaningful (something other than i, j, or k if the loop is more than one or two lines long or is nested)?	X	
22	Have all "temporary" variables been renamed to something more mean-ingful?	X	
23	Are boolean variables named so that their meanings when they're true are clear?	X	
24	Do enumerated-type names include a prefix or suffix that indicates the cat-egory—for example, Color_ for Color_Red, Color_Green, Color_Blue, and so on?		No es necesario ya que para referenciar al enum se llama la variable en donde se define el enum y después el nombre del enum Ejemplo: Color.RED en lugar de escribir Color.COLOR_RED por redundancia
25	Are named constants named for the abstract entities they represent rather than the numbers they refer to?	X	
26	Does the convention distinguish among local, class, and global data?	X	No aplicable para variables globables (no hay)
27	Does the convention distinguish among type names, named constants, enumerated types, and variables?	X	
28	Does the convention identify input-only parameters to routines in lan-guages that don't enforce them?		No aplicable
29	Is the convention as compatible as possible with standard conventions for the language?	X	
30	Are names formatted for readability?	X	
31	Does the code use long names (unless it's necessary to use short ones)?	X	No tan largos, pero algunas veces si
32	Does the code avoid abbreviations that save only one character?	X	

33	Are all words abbreviated consistently?	X	
34	Are the names pronounceable?	X	
35	Are names that could be misread or mispronounced avoided?	X	
36	Are short names documented in translation tables?		No aplicable.
37	Avoided names that are misleading?	X	
38	Avoided names with similar meanings?	X	
39	Avoided names that are different by only one or two characters?	X	
40	Avoided names that sound similar?	X	
41	Avoided names that use numerals?	X	No aplica para las variables generadas automáticamente para los JFrame
42	Avoided names intentionally misspelled to make them shorter?	X	
43	Avoided names that are commonly misspelled in English?		No aplicable.
44	Avoided names that conflict with standard library routine names or with pre-defined variable names?	X	
45	Avoided totally arbitrary names?	X	
46	Avoided hard-to-read characters?	X	
47	Does the program use a different type for each kind of data that might change?		
48	Are type names oriented toward the real-world entities the types represent rather than toward programming-language types?		No aplicable. No se han agregado nuevos tipos de datos
49	Are the type names descriptive enough to help document data declarations?		No aplicable. No se han agregado nuevos tipos de datos
50	Have you avoided redefining predefined types?		No aplicable. No se han agregado nuevos tipos de datos
51	Have you considered creating a new class rather than simply redefining a type?	X	

Técnicas de organización de sentencias

No	Practica	Uso	Comentario
52	Does the code make dependencies among statements obvious?	X	
53	Do the names of routines make dependencies obvious?	X	En contexto de algebra lineal, debería ser obvio
54	Do parameters to routines make dependencies obvious?	X	

55	Do comments describe any dependencies that would otherwise be unclear?	X	
56	Have housekeeping variables been used to check for sequential dependen-cies in critical sections of code?		No aplicable. No se utilizan variables para hacer housekeeping
57	Does the code read from top to bottom?	X	
58	Are related statements grouped together?	X	
59	Have relatively independent groups of statements been moved into their own routines?	X	

Técnicas de construcción de estructuras de control de flujo

No	Practica	Uso	Comentario
60	If the loop is a for loop, does the code inside it avoid monkeying with the loop index?	X	
61	Is a variable used to save important loop-index values rather than using the loop index outside the loop?	X	
62	Is the loop index an ordinal type or an enumerated type—not floating-point?	X	
63	Does the loop index have a meaningful name?	X	
64	Does the loop avoid index cross-talk?	X	
65	Does the loop end under all possible conditions?	X	
66	Does the loop use safety counters—if you've instituted a safety-counter standard?	X	
67	Is the loop's termination condition obvious?	X	
68	If break or continue are used, are they correct?	X	
69	Does each routine use return only when necessary?	X	
70	Do returns enhance readability?	X	Probablemente.
71	Does the recursive routine include code to stop the recursion?	X	Hay casos base y casos de salida
72	Does the routine use a safety counter to guarantee that the routine stops?	X	
73	Is recursion limited to one routine?	X	
74	Is the routine's depth of recursion within the limits imposed by the size of the program's stack?	X	Probablemente
75	Is recursion the best way to implement the routine? Is it better than simple iteration?		De hecho, la única función que utiliza la recursión es el determinante porque se basa en sub-matrices. Si se hiciera con

			ciclos normales seria menos eficiente y más difícil de entender.
76	Are gotos used only as a last resort, and then only to make code more read-able and maintainable?		No aplicable.
77	If a goto is used for the sake of efficiency, has the gain in efficiency been measured and documented?		No aplicable.
78	Are gotos limited to one label per routine?		No aplicable.
79	Do all gotos go forward, not backward?		No aplicable.
80	Are all goto labels used?		No aplicable.
81	Do expressions use true and false rather than 1 and 0?	X	
82	Are boolean values compared to true and false implicitly?	X	
83	Are numeric values compared to their test values explicitly?	X	
84	Have expressions been simplified by the addition of new boolean vari-ables and the use of boolean functions and decision tables?		Las expresiones que lo necesitan no son tan complicadas y se entienden fácilmente
85	Are boolean expressions stated positively?	X	
86	Do pairs of braces balance?	X	
87	Are braces used everywhere they're needed for clarity?	X	
88	Are logical expressions fully parenthesized?	X	
89	Have tests been written in number-line order?	X	
90	Do Java tests uses a.equals(b) style instead of a == b when appropriate?	X	
91	Are null statements obvious?	X	
92	Have nested statements been simplified by retesting part of the condi-tional, converting to if-then-else or case statements, moving nested code into its own routine, converting to a more object-oriented design, or have they been improved in some other way?		Se hizo el intento, pero aun considero que hay muchas maneras de mejorar los algoritmos, pero no se me ocurre como.
93	If a routine has a decision count of more than 10, is there a good reason for not redesigning it?	X	

Técnicas de construcción de procedimientos

No	Practica	Uso	Comentario
94	Is the reason for creating the routine sufficient?	X	

95	Have all parts of the routine that would benefit from being put into routines of their own been put into routines of their own?	X	
96	Is the routine's name a strong, clear verb-plus-object name for a procedure or a description of the return value for a function?	X	
97	Does the routine's name describe everything the routine does?	X	
98	Have you established naming conventions for common operations?	X	
99	Does the routine have strong, functional cohesion—doing one and only one thing and doing it well?	X	
100	Do the routines have loose coupling—are the routine's connections to other routines small, intimate, visible, and flexible?	X	
101	Is the length of the routine determined naturally by its function and logic, rather than by an artificial coding standard?	X	Aunque algunos algoritmos podrían ser más cortos.
102	Does the routine's parameter list, taken as a whole, present a consistent interface abstraction?	X	Probablemente.
103	Are the routine's parameters in a sensible order, including matching the order of parameters in similar routines?	X	
104	Are interface assumptions documented?		No aplica.
105	Does the routine have seven or fewer parameters?	X	
106	Is each input parameter used?	X	
107	Is each output parameter used?	X	
108	Does the routine avoid using input parameters as working variables?	X	
109	If the routine is a function, does it return a valid value under all possible circumstances?	X	Se cambiaron los retornos nulos a una forma de retornar un mensaje de error cuando no se puede realizar la función.

Técnicas de construcción de clases

No	Practica	Uso	Comentario
110	Have you thought of the classes in your program as abstract data types and evaluated their interfaces from that point of view?	X	Específicamente la clase Matriz
111	Does the class have a central purpose?	X	

112	Is the class well named, and does its name describe its central purpose?	X	
113	Does the class's interface present a consistent abstraction?		No aplicable.
114	Does the class's interface make obvious how you should use the class?		No aplicable.
115	Is the class's interface abstract enough that you don't have to think about how its services are implemented? Can you treat the class as a black box?		No aplicable.
116	Are the class's services complete enough that other classes don't have to meddle with its internal data?	X	
117	Has unrelated information been moved out of the class?	X	
118	Have you thought about subdividing the class into component classes, and have you subdivided it as much as you can?	X	No creo que necesite alguna subdivisión
119	Are you preserving the integrity of the class's interface as you modify the class?	X	
120	Does the class minimize accessibility to its members?	X	Todo se escribe como privado por defecto
121	Does the class avoid exposing member data?	X	
122	Does the class hide its implementation details from other classes as much as the programming language permits?	X	Probablemente se pueda encapsular mejor, pero no se mucho de Java
123	Does the class avoid making assumptions about its users, including its derived classes?	X	
124	Is the class independent of other classes? Is it loosely coupled?	X	
125	Is inheritance used only to model "is a" relationships—that is, do derived classes adhere to the Liskov Substitution Principle?		No aplicable.
126	Does the class documentation describe the inheritance strategy?		No aplicable.
127	Do derived classes avoid "overriding" non-overridable routines?		No aplicable.
128	Are common interfaces, data, and behavior as high as possible in the inheritance tree?		No aplicable.
129	Are inheritance trees fairly shallow?		No aplicable.

130	Are all data members in the base class private rather than protected?	X	
131	Does the class contain about seven data members or fewer?	X	
132	Does the class minimize direct and indirect routine calls to other classes?	X	
133	Does the class collaborate with other classes only to the extent absolutely necessary?	X	
134	Is all member data initialized in the constructor?	X	
135	Is the class designed to be used as deep copies rather than shallow copies unless there's a measured reason to create shallow copies?		No aplicable.
136	Have you investigated the language-specific issues for classes in your specific programming language?	X	Probablemente.

Técnicas de documentación de código

No	Practica	Uso	Comentario
137	Can someone pick up the code and immediately start to understand it?	X	Es posible.
138	Do comments explain the code's intent or summarize what the code does, rather than just repeating the code?	X	
139	Is the Pseudocode Programming Process used to reduce commenting time?	X	
140	Has tricky code been rewritten rather than commented?	X	He hecho muchos cambios al código, pero probablemente todavía haya algunas cosas que puedan ser cambiadas
141	Are comments up to date?	X	
142	Are comments clear and correct?	X	Claridad es subjetiva de quien lo lea Pero considero que esta correcto hasta donde puedo entender
143	Does the commenting style allow comments to be easily modified?	X	
144	Does the code avoid endline comments?	X	
145	Do comments focus on why rather than how?	X	
146	Do comments prepare the reader for the code to follow?	X	
147	Does every comment count? Have redundant, extraneous, and self-indul-	X	Algunos comentarios redundantes no se eliminaron por creer que todo el proyecto

	gent comments been removed or improved?		necesitaba estar comentado extensivamente
148	Are surprises documented?	X	No hay sorpresas, pero funcionalidades raras han sido comentadas
149	Have abbreviations been avoided?	X	
150	Is the distinction between major and minor comments clear?	X	
151	Is code that works around an error or undocumented feature commented?	X	
152	Are units on data declarations commented?	X	
153	Are the ranges of values on numeric data commented?	X	
154	Are coded meanings commented?	X	
155	Are limitations on input data commented?	X	
156	Are flags documented to the bit level?		No aplicable.
157	Has each global variable been commented where it is declared?		No aplicable.
158	Has each global variable been identified as such at each usage, by a naming convention, a comment, or both?		No aplicable.
159	Are magic numbers replaced with named constants or variables rather than just documented?	X	
160	Is each control statement commented?	X	Solo lo necesario
161	Are the ends of long or complex control structures commented or, when possible, simplified so that they don't need comments?		
162	Is the purpose of each routine commented?	X	
163	Are other facts about each routine given in comments, when relevant, including input and output data, interface assumptions, limitations, error corrections, global effects, and sources of algorithms?	X	Solo lo necesario
164	Does the program have a short document, such as that described in the Book Paradigm, that gives an overall view of how the program is orga-nized?	X	Este documento probablemente se considere como ello ya que contiene los diagramas UML
165	Is the purpose of each file described?	X	
166	Are the author's name, e-mail address, and phone number in the listing?		Solo el nombre

Estándares de codificación

No	Practica	Uso	Comentario

167	i and j are integer indexes.	X	También “k” entra en algunos casos
168	Constants are in ALL_CAPS separated by underscores.		No aplicable.
169	Class and interface names capitalize the first letter of each word, including the first word—for example, ClassOrInterfaceName.	X	
170	Variable and method names use lowercase for the first word, with the first letter of each following word capitalized—for example, variableOrRoutineName.	X	
171	The underscore is not used as a separator within names except for names in all caps.	X	
172	The get and set prefixes are used for accessor methods.	X	

Legibilidad y calidad del código

El código de java se encuentra dentro del directorio del proyecto

Pruebas de Aceptación

El proyecto consiste en desarrollar una aplicación que permita realizar operaciones con matrices. El programa deberá presentar una interfaz que permita al usuario seleccionar el tipo de operación a realizar y deberá mostrar el resultado de la operación.

Las siguientes operaciones deben formar parte de la funcionalidad de la aplicación:

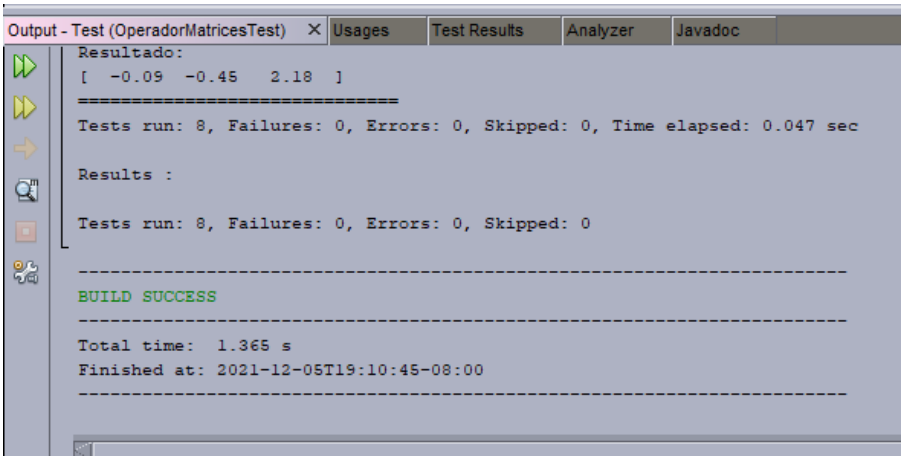
1. Suma de matrices.
2. Multiplicación de matrices por una escalar.
3. Multiplicación de matrices.
4. Obtención de la inversa de una matriz por el método de Gauss-Jordan.
5. Transpuesta de una matriz.
6. Solución de un sistema de ecuaciones por el método de Gauss-Jordan.
7. Obtención del determinante de una matriz.
8. Solución de un sistema de ecuaciones por el método de Cramer.

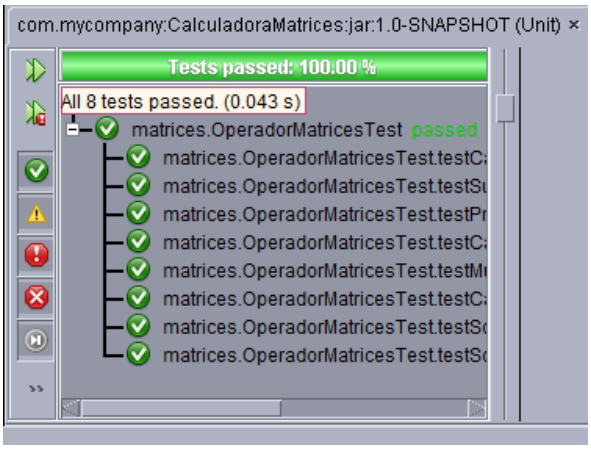
De las pruebas realizadas manualmente probando el programa y las unitarias con JUnit, el programa debería de cumplir con realizar las 8 funcionalidades de la aplicación aparte de mostrar una interfaz para poder modificar matrices y el seleccionar cuales de las matrices se van a utilizar como argumento para las operaciones mencionadas anteriormente.

La clase principal es Main.java. Puede que la interfaz sea un poco rara de entender al principio, pero explorándola por unos minutos se entiende como se utiliza sin explicaciones extensas.

Empleo de herramientas de construcción

JUnit





El archivo de las pruebas unitarias es OperadorMatricesTest.java

JavaDoc

El JavaDoc fue generado con la herramienta Doxygen y se encuentra dentro de la carpeta “doxygen” del directorio raíz del proyecto.

Lecciones aprendidas

Sobre JUnit

Yo no solía hacer pruebas unitarias de ningún tipo. Tengo experiencia creando sistemas dinámicos donde el usuario es un factor importante y cualquier menor fallo es una catástrofe; por lo que creía que la mejor forma de probar un programa es utilizando el recurso humano porque hay cosas que solo un humano puede detectar y una maquina no.

Esto es razonable hasta que me doy cuenta que también probaba cosas una y otra vez de manera manual que no requerían del recurso humano, y que cada versión del programa debería retornar los mismos resultados para saber que nada se ha roto.

Es ahí en donde vi que las pruebas unitarias, aunque probablemente no sirvan para detectar patrones muy complejos del usuario final, si sirven para probar cosas que se pueden calcular y predecir entre diferentes versiones del programa.

En este proyecto, si no fuera por las pruebas unitarias, estaría tratando de introducir las mismas matrices una y otra vez para verificar que después de cada cambio realizado el algoritmo funcione o siga funcionando.

Esto me hubiera consumido muchísimo tiempo de no ser por las pruebas unitarias, que hacen el trabajo sucio cada vez que se corre el programa de manera automática. Así ya puedo saber que sirve, que no sirve y lo que dejo de servir con cada cambio/compilación realizada.

Sobre JavaDoc

Yo también documento código, pero no me había percatado que tampoco debía de perder tiempo en generar un archivo PDF en el cual se encuentre toda la documentación del programa si se puede hacer de forma automática con una herramienta como JavaDoc. Ahora cada vez que hago cambios a los comentarios, no es necesario ir a la documentación a cambiar algo, si no que simplemente genero el JavaDoc y todo se actualiza conforme a lo que este escrito en el código fuente.

Cuando tienes comentarios que utilizan nombres de clases o de campos, esto ahorra tu tiempo de forma increíble, ya que JavaDoc también genera sus descripciones y las vincula.

En conclusión, ambas herramientas son increíbles para ahorrar tiempo durante el desarrollo de cualquier proyecto, y aunque estas herramientas estén disponibles solo para Java, estoy seguro de que muchos otros lenguajes tienen su propios frameworks de pruebas unitarias o de documentación automática.

Conclusión del curso

Antes de este curso creía que mi código era decente. Ahora cada vez que programo algo o veo código, me acuerdo de todas las técnicas que vimos durante el curso y tengo el deseo interno de mejorarlo, puesto que hay una diferencia abismal entre un código bien hecho y uno hecho a medias.

Aunque admito que en este proyecto el código realizado no sea perfecto, o siquiera bueno, al menos me quedo con lo aprendido para que pueda mejorar mi técnica con la práctica y escribir mejor código cuando exista la posibilidad de dedicarle más tiempo a su construcción, y tal vez en algo que sea menos complejo que algebra lineal.