UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

"DESARROLLO DE APLICACIONES EN JAVA
PARA VULCANOLOGÍA"

T E S I S P R O F E S I O N A L

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A:

MARIO EDUARDO ABURTO GUTIÉRREZ

DIRECTOR DE TESIS: ING. CARLOS ALBERTO ROMÁN ZAMITIZ

CIUDAD UNIVERSITARIA, MÉXICO D. F. MAYO DE 2014

A mi madre, María Antonia Gutiérrez Enríquez,

por todo el amor que me ha dado.

Lo que soy se lo debo a ella.

A mi padre, Ángel Aburto Monjardín,

por apoyarme siempre y ayudarme

a superar los momentos más

difíciles de mi vida.

Agradecimientos

Al Ing. Carlos Alberto Román Zamitiz, gracias por la paciencia que me tuvo y el apoyo que me brindó para la realización de esta tesis.

Al Dr. Juan Manuel Espíndola Castro y a la Dra. Araceli Zamora Camacho les agradezco su calidez humana mientras realizaba mi servicio social en el Instituto de Geofísica, en donde desarrollé el proyecto en el que se basa esta tesis.

Por último, agradezco a la Universidad Nacional Autónoma de México, así como a la Facultad de Ingeniería, por todo lo que he aprendido en ellas durante estos años de estudio. Espero poder retribuirlas algún día.

Índice de contenido

I. Introducción

A. El conocimiento científico y las tecnologías de la información

El ser humano está en contacto constante con la tecnología, ya sea en el hogar, en el trabajo o en la escuela. Es increíble las cosas que son posibles en la actualidad gracias al desarrollo tecnológico: viajes espaciales, fábricas completamente automatizadas, videojuegos cada vez más realistas, operaciones médicas asistidas con robots, etc. Antes de continuar quisiera definir primero lo que es la tecnología. El diccionario de la real academia española define tecnología como el "Conjunto de teorías y de técnicas que permiten el aprovechamiento práctico del conocimiento científico". Como se puede apreciar la tecnología depende del conocimiento científico. Sin embargo, también el conocimiento científico depende en gran medida del uso de la tecnología, como se profundizará más adelante.

Un área específica de la tecnología que ha crecido exponencialmente en las últimas décadas es la de las tecnologías de la información. Las redes sociales, los teléfonos inteligentes, las tabletas y los libros electrónicos son ejemplos de productos tecnológicos que vemos todos los días. En los últimos años, se ha dado un gran avance en cuanto a la facilidad de uso de las tecnologías de la información. Algunos productos de las tecnologías de la información son tan intuitivos que ya no es necesario la lectura de manuales para su uso. Pero no sólo la vida cotidiana de las personas ha cambiado con estas tecnologías. También la industria y la investigación se han beneficiado de ellas.

La investigación científica en la actualidad se apoya en gran medida en programas computacionales. Estos programas ayudan a los científicos a procesar datos, hacer cálculos, realizar consultas de información y por supuesto, a redactar los resultados de sus investigaciones. Además, los mismos científicos en ocasiones requieren crear sus propios programas para probar o simular modelos matemáticos para sus investigaciones.

B. La vulcanología y las tecnologías de la información

La vulcanología es una rama de la geología que se encarga del estudio de los volcanes, así como otros fenómenos geológicos relacionados a éstos. Su objetivo último es la predicción de erupciones, lo cual hasta el momento es todavía algo muy lejano. Para su estudio utiliza métodos geofísicos, geodésicos, geoquímicos y geológicos, que se apoyan en tecnologías electrónicas, de comunicaciones e informáticas.

El estudio de la vulcanología ha evolucionado principalmente en los últimos dos siglos. Sin embargo, la primera erupción registrada en la historia fue la del volcán Vesubio en el año 79 D.C. y fue descrita por Plinio el Joven. En 1841 Giuseppe Mercalli fundó el primer observatorio vulcanológico del mundo en el reino de las dos sicilias, llamado el observatorio Vesubio. En 1879 se creó el Servicio Geológico de los Estados Unidos para consolidar las ciencias de la tierra desde una sola organización. En 1922 se fundó el *Bulletin Volcanologique*, un boletín que publica constantemente artículos científicos relacionados a la vulcanología. Este boletín fue creado por la *International Association of Volcanology and Chemistry of the Earth's Interior* (IAVCEI) que fue creada en 1919 como un esfuerzo internacional por unificar el conocimiento relacionado a la vulcanología.

La vulcanología se ayuda de diversas herramientas para sus estudios. Se usan sismógrafos cerca de los volcanes para registrar la sismicidad durante eventos volcánicos. También se monitoriza la deformación de la superficie y se observa los cambios de temperatura, así como las emisiones de gases. Se usan satélites para monitorizar grandes áreas sin necesidad de tener instrumentos localmente, principalmente esto se hace en zonas de difícil acceso en los que usar otros instrumentos sería demasiado caro.

Anteriormente, todos los datos de los instrumentos tenían que ser capturados manualmente en la computadora para luego procesarlos o analizarlos por medio de programas. El desarrollo de la tecnología ha llegado también a estas ciencias y actualmente los equipos pueden almacenar la información electrónicamente, la cual es

transferida a memorias externas o por medio de la red. Estos equipos son extremadamente caros, debido a que deben tener una gran precisión.

Muchos de los datos recogidos y el análisis que se hace sirve para determinar el nivel de actividad de un volcán, lo que puede ayudar a prevenir a la población en caso de una erupción inminente. Esto es muy difícil de hacer, pues los volcanes son muy impredecibles, sin embargo, a través de los años el conocimiento sobre ellos ha aumentado considerablemente. Sin embargo, es también del interés de los vulcanólogos el análisis de erupciones que ya ocurrieron, pues en gran medida ayudan a entender cómo se puede desarrollar en caso de que vuelva a ocurrir. Para esto se utilizan modelos matemáticos que pueden describir estas erupciones.

Constantemente los científicos crean modelos para describir fenómenos vulcanológicos, sin embargo muchos de esos modelos se quedan sólo en la teoría. Tal es el caso del modelo que describe una columna eruptiva pliniana, propuesto por Woods en 1988 y sobre el que se hablará más adelante. Descrito a grandes rasgos, este modelo permite obtener las coordenadas de la gráfica de una columna eruptiva para una erupción pliniana. Sin embargo, para aplicarlo, los científicos tienen que programar métodos numéricos para resolver las ecuaciones que lo componen, lo que en muchas ocasiones los lleva a buscar otros métodos más rápidos aunque no tan exactos. Aún cuando los investigadores implementan modelos matemáticos como el de Woods, generalmente lo hacen para uso personal, por lo que sus programas no hacen uso de una interfaz gráfica amigable y es complicado para otra persona usarlos si es que lo requiriera.

C. Objetivos

Con este proyecto se pretende dar un enfoque práctico a un modelo matemático de vulcanología creando un programa con una interfaz gráfica amigable al usuario y que pueda ser usado por un número extenso de usuarios. Con este incremento en el número de usuarios el programa además podría mejorar, pues se les daría la posibilidad a los usuarios de reportar errores o transmitir observaciones.

Lo que se propone hacer en este trabajo es demostrar como los ingenieros en computación podemos aportar a la ciencia, específicamente a la vulcanología, para hacer programas que tengan un alcance más grande y que sean amigables para los usuarios.

Objetivos específicos:

- Identificar problemas o áreas de mejora en un programa de Fortran para vulcanología
- Describir una propuesta de desarrollo para realizar programas en vulcanología.
- Describir el desarrollo de una aplicación completa y funcional para vulcanología que aproveche los conocimientos de un ingeniero en computación

D. Limitaciones

El problema se limitó a vulcanología, aunque se podría aplicar a otras ciencias. Sin embargo, es en ésta que se identificó dicho problema por lo que sólo se trabajará en esta área de la física. Además se usará un ejemplo de un modelo para erupciones volcánicas plinianas, para demostrar la manera en la que un Ingeniero en Computación puede aportar a la vulcanología aplicando principalmente los conocimientos de programación orientada a objetos e ingeniería de software, aunque también se aplicaron otros conocimientos.

Se implementó el método numérico para este modelo a partir de un código ya probado en Fortran, por lo que no se hizo desde cero, pues implicaría más tiempo el analizar el modelo y entenderlo, siendo que los investigadores ya tienen ese conocimiento. Por lo tanto, en el programa desarrollado en Java se tuvo como prioridad la interfaz de usuario, facilidad de uso, portabilidad, etcétera; temas importantes dentro de la Ingeniería en Computación. Además, se asume que el programa de Fortran arroja resultados correctos, por lo que sólo se verifica que el programa en Java arroje los mismos resultados para comprobar que los resultados son correctos.

A pesar de esto, se incluye la teoría básica sobre la vulcanología y las erupciones volcánicas para entender el problema que se está tratando. La teoría se muestra de la forma más simple posible, pues esta tesis es más de carácter ingenieril que científico. Por ello no se hace mayor énfasis en la deducción del modelo que describe las erupciones plinianas. En cambio, se presenta el modelo con una breve introducción y después se presentan las fórmulas del modelo.

II. Marco teórico

A. Descripción de la problemática

El propósito de esta tesis es mostrar la metodología de desarrollo para crear aplicaciones que sirvan a la vulcanología. Por ello se usará un programa como base para mostrar dicha metodología. En este caso se usará un programa escrito en Fortran que analiza el comportamiento de una erupción pliniana y se analizará cómo dotar a este programa de nuevas características que pueden ser aportadas por un ingeniero en computación. Este programa lleva por nombre **Columni** y fue creado por el Dr. Juan Manuel Espíndola Castro.

El Dr. Espíndola es un investigador del Departamento de Vulcanología del Instituto de Geofísica de la UNAM. Se graduó como Físico de la Facultad de Ciencias de la UNAM; su Maestría en Ciencias, así como su Doctorado en Filosofía (Geociencias) los realizó en la Universidad Purdue, West Lafayette, en Indiana, EEUU. Entre sus líneas de investigación se encuentran la estratigrafía de productos volcánicos, así como la aplicación de métodos numéricos a problemas de vulcanología física. Es precisamente en esta última área en la que entra el programa Columni, pues usa un modelo matemático que requiere del método numérico de Runge-Kutta para su solución.

El modelo matemático que describe las erupciones plinianas, usado en el programa Columni, lo tomó el Dr. Espíndola del artículo *The fluid dynamics and thermodynamics of eruption columns* escrito por A. W. Woods en 1988. En este artículo Woods propone un modelo para describir las erupciones plinianas. En el mismo artículo se compara este modelo con modelos anteriores que tienen inconsistencias y simplificaciones que este nuevo modelo intenta corregir.

Las erupciones volcánicas se pueden clasificar, entre otras maneras, en base a su nivel de explosividad. Para ello se usa el Índice de Explosividad Volcánica (IEV) que tiene una escala del 0 al 8, en base a varios factores mensurables como la altura alcanzada por la columna

eruptiva, duración de la erupción, entre otros. La tabla 1 muestra una clasificación de las erupciones volcánicas con un IEV asociado a cada una de ellas:

IEV	Clasificación	Descripción	Altura columna eruptiva	Volumen material arrojado	Periodicidad	Ejemplo	Total erupciones históricas
0	Erupción hawaiana	no-explosiva	< 100 m	> 1000 m³	diaria	Villarrica	-
1	Erupción stromboliana	ligera	100-1000 m	> 10,000 m³	diaria	Stromboli	-
2	Erupción vulcaniana/ stromboliana	explosiva	1-5 km	> 1.000.000 m³	semanal	Galeras, 1993	3477
3	Erupción Vulcaniana (sub-pliniana)	violenta	5-15 km	> 10.000.000 m³	anual	Nevado del Ruiz, 1985	868
4	Vulcaniana (sub-pliniana)/ pliniana	cataclísmica	10-25 km	> 0,1 km³	cada 10 años	Galunggung, 1982	278
5	Pliniana	paroxística	> 25 km	> 1 km³	cada 100 años	St. Helens, 1980	84
6	Pliniana/ Ultra-Pliniana (krakatoana)	colosal	> 25 km	> 10 km³	cada 100 años	Krakatoa, 1883 Santa María,1902	39
7	Ultra-Pliniana (krakatoana)	super-colosal	> 25 km	> 100 km³	cada 1.000 años	Tambora, 1815 Maipo, 500.000 a. C.	4
8	Ultra-Pliniana (krakatoana)	mega-colosal	> 25 km	> 1000 km³	cada 10.000 años	Toba, 69.000 a. C.	1

Tabla 1: Clasificación de erupciones volcánicas con base en su IEV.
*http://es.wikipedia.org/wiki/%C3%8Dndice_de_explosividad_*

Como el título del artículo lo menciona, el modelo analiza la dinámica de fluidos y la termodinámica de las columnas eruptivas, específicamente de las erupciones plinianas. Este tipo de erupciones se caracteriza por ser las erupciones de mayor violencia (vea la tabla 1) y toman su nombre debido a la erupción descrita por Plinio el Joven del volcán Vesubio. Las erupciones plinianas poseen las siguientes características:

- Columnas eruptivas de alturas superiores a los 30 km, que llegan a la estratósfera.
- Alto grado de explosividad con expulsión de grandes volúmenes de gas volcánico, fragmentos y cenizas.
- Grandes emisiones de pumita (piedra pómez) y continuas e intensas expulsiones de ráfagas de gas tóxico.
- La lava es generalmente riolita (tipo de roca ígnea, ver ilustración 1) , aunque se han dado casos de lava basáltica (tipo de roca ígnea, ver ilustración 2).

Ilustración 2: Roca Riolita.
http://es.wikipedia.org/wiki/Riolita

Ilustración 1: Roca Basáltica.
http://es.wikipedia.org/wiki/Basalto

En este tipo de erupción, el material es expulsado a velocidades de cientos de metros por segundo que hace que el material alcance unas decenas de kilómetros. La parte más baja de la columna eruptiva, en la que se encuentra el material más denso es llamada **gas-thrust region**. El aire entra en la columna y se expande debido al calor lo que provoca que el material en la columna se vuelva flotante (**buoyant**) . Si hay suficiente energía térmica se genera una sección en la columna llamada **buoyancy-driven region** o **convective region**, lo que ocasiona que la columna se eleve unas decenas de kilómetros. Si el momento inicial o algún otro factor como la temperatura es insuficiente para que el material llegue al punto de la **convective region**, la columna colapsa a una altura de unos cuantos kilómetros solamente.

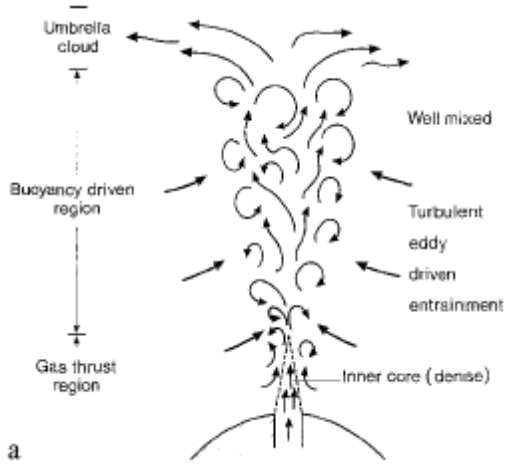


Ilustración 3: Diagrama esquemático. Woods, A. W., 1988

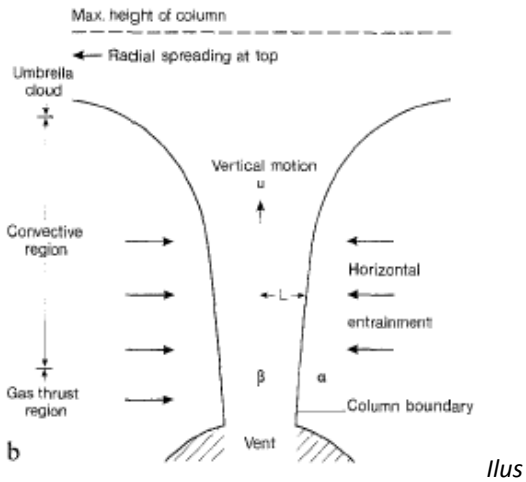


Ilustración 4: Diagrama del modelo. Woods, A. W., 1988

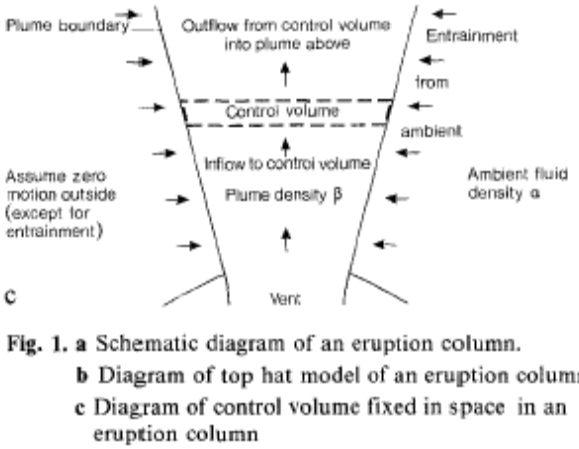


Fig. 1. a Schematic diagram of an eruption column.
b Diagram of top hat model of an eruption column.
c Diagram of control volume fixed in space in an eruption column

Ilustración 5: Diagrama del volumen de control. Woods, A. W., 1988

En la ilustración 3 y la ilustración 4 se aprecian las secciones de la columna eruptiva: la **gas-thrust region**, la **convective region** y la **umbrella cloud**. En ambos se observa como el aire entra en la columna lo que hace que tenga el efecto descrito anteriormente, aunque en el diagrama b se especifican ya algunas de las variables del modelo. En la ilustración 5 se observa el análisis a partir de un volumen de control que sirve a Woods para desarrollar el modelo, cuyas ecuaciones serán descritas en la siguiente sección.

1. Ecuaciones que modelan una erupción pliniana

A lo largo del artículo se describe la deducción de las fórmulas que forman el modelo propuesto por Woods. Como se mencionó en la sección , aquí se omite la deducción de las fórmulas, pues sólo es de nuestro interés la aplicación de ellas en el programa. A continuación se muestran las ecuaciones que forman el modelo, aclarando que la numeración que tienen es la misma que aparece en el artículo de Woods:

$$\frac{d}{dz}\left(\beta U^2 L^2\right) = g(\alpha - \beta)L^2 \quad \text{(2)}$$

$$\frac{1}{\beta} = (1-n)\frac{1}{\sigma} + \frac{nR_g\theta}{P} \quad \text{(3)}$$

$$n = 1 + (n_0 - 1)\frac{L_0^2 U_0 \beta_0}{L^2 U \beta} \quad \text{(4)}$$

$$R_g = R_a + \left(R_{g0} - R_a\right)\left(\frac{1-n}{n}\right)\left(\frac{n_0}{1-n}\right) \quad \text{(5)}$$

$$\frac{d}{dz}\left(C_p \theta \beta U L^2\right) = (C_a T)\frac{d}{dz}\left(\beta U L^2\right) + \frac{U^2}{2}\frac{d}{dz}\left(\beta U L^2\right) - \alpha U L^2 g \quad \text{(8)}$$

$$C_p = C_a + \left(C_{p0} - C_a\right)\frac{(1-n)}{(1-n_0)} \quad \text{(10)}$$

$$U\frac{dU}{dz} = \frac{-U^2}{8L}\sqrt{\frac{\alpha}{\beta}} + \frac{g(\alpha - \beta)}{\beta} \quad \text{(16)}$$

$$\frac{d}{dz}\left(\beta U L^2\right) = 2kUL\alpha \quad \text{(18)}$$

$$\alpha \le \beta \quad \text{(19)}$$

De acuerdo al artículo para la **gas-thrust region** se usan las ecuaciones 16,2,8,3,4,5 y 10 a las que se nombra como el modelo A. Para la **convective region** se usan las ecuaciones 18,2,8,3,4,5 y 10, a la cual se nombra modelo B. Para determinar dónde termina la **gas-thrust region** y donde empieza la **convective region** se usa la ecuación 19, es decir, la región cambia cuando alfa es mayor o igual a beta. Los modelos A y B y este último criterio

forman el modelo AB que describe la columna pliniana. Las ilustraciones 6 y 7 muestran una tabla con las variables y constantes definidas por Woods en su artículo.

Tabla. Lista de variables y constantes

Nota: el subíndice *o* denota una cantidad evaluada en el cráter, el subíndice ε denota una cantidad de arrastre (*entrained*)

C_a calor específico a presión constante del aire, 998 $J / (K\ kg)$

C_m calor específico a presión constante del gas volcánico emitido en el cráter, 1617 $J / (K\ kg)$

C_p calor específico aparente a presión constante del material en la columna

C_s calor específico de los piroclastos sólidos, 1617 $J / (K\ kg)$

C_v calor específico en volumen constante del aire

e la entalpía local específica en la columna

E entalpía específica aparente en la columna

f fracción de la altura de la columna en la cual todo el aire puede ser considerado como si hubiera sido arrastrado (*entrained*)

F fracción de sólidos en contacto termal con los gases

F_o el flujo de la flotación en el cráter por densidad unitaria

g la aceleración gravitacional, 9.81 m/s

G la frecuencia de Brunt-Vasaiala

H la altura de la escala atmosférica

H_1 la altura de la tropósfera, 11 km

H_2 la altura de la estratósfera, 20 km

H_t la altura del punto más alto de la columna

h la altura en la columna

k la constante de arrastre, 0.09

L el radio de la columna

$\dot{m}$ un flujo de masa en la columna

m la masa que pasa a través del borde del volumen de control

m_a la masa de aire arrastrado en un desplazamiento del grupo en una distancia δz

m_b la masa total de aire arrastrado dentro de la columna hasta esa altura

Ilustración 6: Variables usadas en el artículo de Woods. Traducción realizada por Mario Eduardo Aburto Gutiérrez. Woods, A. W., 1988

m_g la masa de gas en la columna
m_m la masa de gas volcánico en la columna
m_s la masa de sólidos en la columna
n el porcentaje de masa de gas en la columna
P la presión atmosférica, definida en la sección 2.5
Q el flujo de entalpía relativo (sección 7.5)
R_a la constante de gas para el aire, 285
R_g la constante de gas aparente para la columna
R_m la constante de gas para el gas volcánico, 462
R_o constante de normalización para L en la Fig. 4
T temperatura atmosférica ambiental
$T_o = 273$
u la velocidad vertical local en la columna
U la velocidad vertical aparente en la columna
U_1 la velocidad máxima en la columna
U_ε la velocidad de arrastre horizontal
WD el trabajo realizado al expandir el gas mientras pasa a través del volumen de control
y la distancia radial desde el eje
z la coordenada del eje de la columna.
α la densidad atmosférica ambiental
$\alpha_o = 1.3$
β la densidad aparente en la columna
γ la razón C_p / C_v
κ el gradiente del radio en la columna, $L = \kappa h$
ρ la densidad local en la columna
θ la temperatura aparente en la columna
λ la razón del radio con el largo de la mezcla
μ el gradiente de la temperatura en la tropósfera, 6.5 K / km
ω el gradiente de la temperatura en la estratósfera, 2.0 K / km
σ la densidad de los piroclastos sólidos
η el cambio en el lapso adiabático / el cambio en el lapso del entorno

Ilustración 7: Variables usadas en el artículo de Woods. Traducción realizada por Mario Eduardo Aburto Gutiérrez. Woods, A. W., 1988

En esta lista de variables se incluyen también algunas que no aparecen en las ecuaciones del modelo, pero que en el artículo se usaban para las deducciones de ecuaciones intermedias.

También es importante mostrar las ecuaciones usadas para determinar la presión (P) y la temperatura (T). Woods no las menciona explícitamente como parte del modelo, pero sí aparecen en el artículo y se usan en el programa. Las ilustraciones 8 y 9 muestran las fórmulas para calcular la presión y la temperatura.

$$T = \begin{cases} T_o - \mu z & \text{for } z \le H_1; \\ T_o - \mu H_1 & \text{for } H_1 \le z \le H_2; \\ T_o - \mu H_1 + \omega(z - H_2) & \text{for } z \ge H_2. \end{cases}$$

Ilustración 8: Ecuación para determinar la temperatura. Woods, A. W., 1988

$$P = \begin{cases} P_o(T_o - \mu z)^{\frac{g}{R_a \mu}} & \text{for } z \le H_1; \\ P_o(T_o - \mu H_1)^{\frac{g}{R_a \mu}} \exp\left(\dfrac{-g(z - H_1)}{R_a(T_o - \mu H_1)}\right) & \text{for } H_1 \le z \le H_2; \\ P_o(T_o - \mu H_1)^{\frac{g}{R_a \mu}} \exp\left(\dfrac{-g(H_2 - H_1)}{R_a(T_o - \mu H_1)}\right)(T_o - \mu H_1 + \omega(z - H_2))^{\frac{-g}{R_a \omega}} & \text{for } z \ge H_2. \end{cases}$$

Ilustración 9: Ecuación para determinar la presión. Woods, A. W., 1988

H1 es la altura de la tropósfera y H2 es la altura de la estratósfera. Como se ve, los valores de la presión y de la temperatura en la columna eruptiva dependen de la altura a la que se encuentre la sección de la columna que se desea calcular y si esta sección de la columna está a una altura inferior a la de la tropósfera, superior a la de la estratósfera o está entre ellas.

2. Código en Fortran que analiza una erupción pliniana

El programa en Fortran usa las ecuaciones descritas anteriormente para obtener las coordenadas para graficar la columna eruptiva, así como algunos otros resultados que se pueden obtener sobre la columna eruptiva. Para poder hacer los cálculos se usa el método de Runge-Kutta de cuarto orden para resolver un sistema de ecuaciones diferenciales ordinarias compuesto por las ecuaciones 2, 8 y 16 ó 18 según sea la sección **gas-thrust-region** o la **convective region**. Las ecuaciones 3,4,5 y 10 no son ecuaciones diferenciales, sin embargo, son parte del sistema y se requieren para resolver todas las variables requeridas.

La ilustración 10 muestra la sección de código que implementa el método de Runge-Kutta de cuarto orden con las ecuaciones 2, 8 y 16:

```
      GAB=(ALFA-BETA)*9.81
      BSA=BETA/ALFA
      SQBA=SQRT(BSA)
      EPS=EPS1
      IF(BSA .LT. 1.0)EPS=EPS2

         DO 1 J=1,4
         FRST=1.
         IF(J.EQ.1)FRST=0.0
         COEF=0.5
         IF(J.EQ.4)COEF=1.
         Y1=Y(1)+(K(1,J)*COEF*FRST)
         Y2=Y(2)+(K(2,J)*COEF*FRST)
         Y3=Y(3)+(K(3,J)*COEF*FRST)
C        FORMAR K1J
         T1=GAB/(BETA*Y1)
         K(1,J)=T1-(2.*Y1*ALFA*EPS/(BETA*B))
C        FORMAR K2J
         K(2,J)=EPS*Y1*B*ALFA*2.
C        FORMAR K3J
         X1=(CA*T)-Y3-(Y1*Y1/2.)
         X2=(Y1*K(1,J))+9.81
         K(3,J)=((2.*EPS*ALFA*X1)/(BETA*B))-X2
1        CONTINUE

      DO 2 I=1,3
      Y(I)=Y(I)+(DH*(K(I,1)+K(I,2)+K(I,2)+K(I,3)+K(I,3)+K(I,4))/6.)
2     CONTINUE
```

Ilustración 10: Implementación de Runge-Kutta en programa de Fortran

Como se puede apreciar hay un ciclo en el que se calculan valores para un arreglo llamado K y otro ciclo en el que se calculan valores para un arreglo llamado Y. El arreglo Y tiene un

tamaño de 3, pues son 3 las ecuaciones diferenciales que se van a resolver. El arreglo K es multidimensional, con 3 filas pues son tres ecuaciones diferenciales y 4 columnas porque se requieren cuatro valores de K para calcular a Y, como se explicará más adelante en el capítulo .

La ilustración 11 muestra otra sección del código en donde se utilizan las ecuaciones 3,4,5 y 10:

```
C     SOLUCION DE ECUACIONES ALGEBRAICAS ACOPLADAS
      UA=U
      U=Y(1)
      BETA2=BETA

      DO 78 JK=1,5
      UBB=U*B*B*BETA
      U0B0B0=U0*B0*B0*BETA0
      N=(ubb-u0b0b0+(n0*u0b0b0))/ubb
      RN1=(1.-N)/N
      RG=RA+((RG0-RA)*RN1*RN2)
      BETA=1./(((1.-N)/DP)+(N*RG*TH/P))
      CP=CA+((CP0-CA)*((1.-N)/(1.-N0)))
      B=SQRT(Y(2)/(Y(1)*BETA))
      TH=Y(3)/CP
78    CONTINUE
```

Ilustración 11: Ecuaciones acopladas en el programa de Fortran

Como se puede observar, aquí se hace uso de los valores de Y calculados anteriormente para obtener variables como U(velocidad), B (radio), n(fracción de masa) y TH (temperatura). Aquí se puede identificar el uso de las ecuaciones 3,4,5 y 10 para obtener las variables BETA (**bulk density**), RG (**bulk gas constant**), N (**gas mass fraction**) y CP (**bulk specific heat**).

3. Ejecución del programa en Fortran y resultados

El programa se corrió usando el IDE Microsoft Developer Studio, que permite la compilación para Fortran. La ilustración 12 muestra la ejecución del programa:

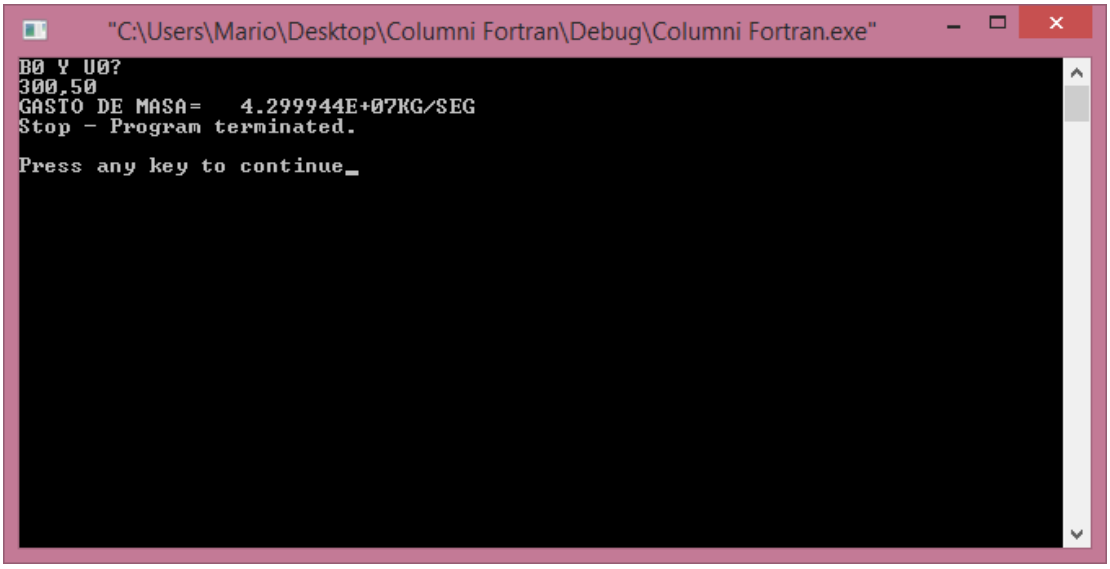


Ilustración 12: Ejecución del programa de Fortran

En la ejecución se pide B0 (radio inicial – radio del cráter del volcán) y U0 (velocidad inicial). El programa lee las demás variables de entrada del archivo *INPUT.txt* cuyo contenido se muestra en la ilustración 13.

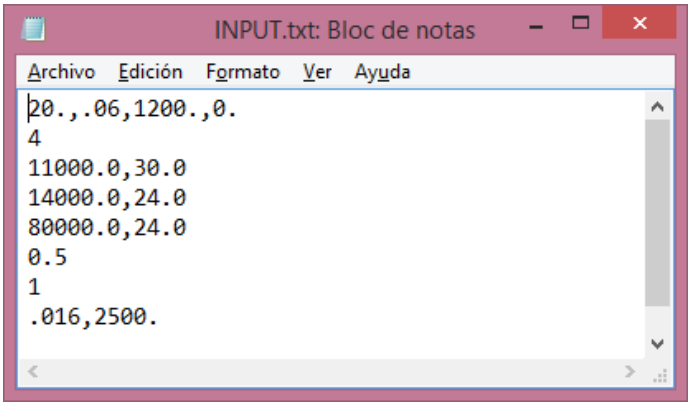


Ilustración 14: Archivo con los valores de las variables de entrada

```
OPEN (5,FILE ='INPUT.txt')
OPEN (6,FILE ='output1.txt')
OPEN (7,FILE ='output2.txt')
```

Ilustración 13: Abrir archivos en Fortran

Como se puede observar no es posible determinar a qué variable se refieren estos valores a simple vista. Para lograrlo, es necesario ver la sección del código en Fortran en donde se leen estos valores. En la ilustración 15 se muestra cómo se lee del archivo *INPUT.txt*. El número 5 fue asignado al archivo *INPUT.txt* en el programa en Fortran para hacer referencia a él, como se puede ver en la ilustración 14.

```
      WRITE(6,*) 'EPS1=',EPS1,'    EPS2=',EPS2

      READ (5,*) DH, N0, TH0, Z0

      WRITE(6,54) DH, N0, TH0, B0, U0, Z0, VELMIN
54    FORMAT(1X,'DH=',F4.1,3X,'N0=',F4.2,3X,'TH0=',F5.0,3X,'B0=',F4.0,
     *3X,'U0=',F5.0,3X,'Z0=',F6.1,3x,'VELMIN=',F3.1)

      READ(5,*) NPINV
C     NPINV NUMERO DE PUNTOS DONDE LA VELOCIDAD CAMBIA DE VALOR+1

      DO 11 I=2,NPINV
      READ(5,*) ZW(I),VW(I)
      WRITE(6,*) 'Z Y VW', ZW(I),VW(I)
11    CONTINUE
```

Ilustración 15: Sección del programa en Fortran donde se lee del archivo INPUT.txt

Los resultados se guardan en los archivos *output1.txt* y *output2.txt*. En el archivo *output1.txt* se guardan los valores de las variables de salida, así como de algunas variables intermedias. La ilustración 16 muestra el contenido del archivo *output1.txt* tras la ejecución del programa.

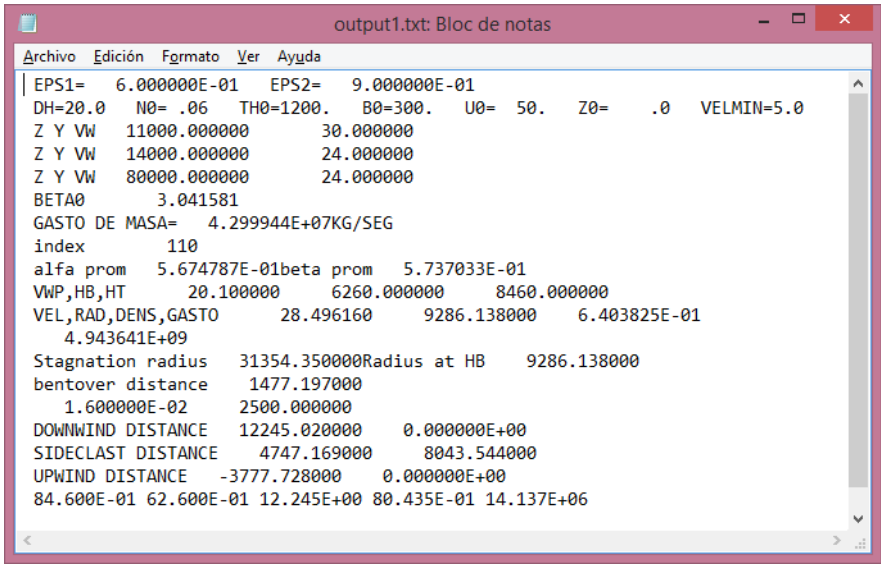


Ilustración 16: Archivo con los resultados del programa de Fortran

El archivo *output2.txt* guarda las coordenadas de la gráfica de la columna eruptiva. La ilustración 17 muestra una porción del archivo *output2.txt* tras la ejecución del programa.

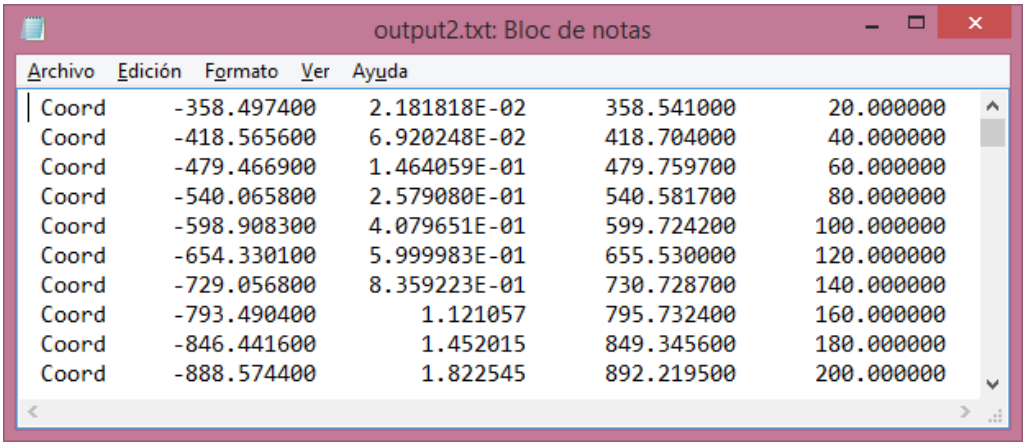


Ilustración 17: Archivo con las coordenadas de la columna eruptiva

El valor que está al final de la línea es la altura (Z), por lo que es la componente *y* de la coordenada. Los otros tres valores son componentes en *x* para una misma altura *y*. El primer valor es el extremo izquierdo de la columna, el segundo valor es el centro de la columna y el tercer valor es el extremo derecho de la columna.

B. Métodos de Runge-Kutta

Como se mencionó, el programa usa el método de Runge-Kutta de cuarto orden. Por ello es necesario conocer en qué consiste este método y cómo se usa. En esta sección se explicarán los métodos de Runge-Kutta en general, para después profundizar en el método específicamente de cuarto orden.

Los métodos de Runge-Kutta fueron diseñados para tener resultados similares a los métodos de las series de Taylor, pero con la ventaja de no requerir la evaluación explícita de las derivadas de *f (t, y)*. Recordemos lo que es una serie de Taylor. La serie de Taylor para una función *f (x)* que es infinitamente diferenciable en *a* (existen derivadas de cualquier orden en *a*) es:

$$f(a)+\frac{f'(a)}{1!}(x-a)+\frac{f''(a)}{2!}(x-a)^2+\frac{f^{(3)}(a)}{3!}(x-a)^3+\cdots$$

La idea es usar una combinación lineal de valores de *f (t, y)* para aproximarse al valor de *y(t)*. La combinación lineal se trata de acercar lo más posible a una serie de Taylor para *y(t)* para obtener métodos del mayor orden posible.

Una EDO del tipo:

$$\frac{dy}{dt}=f(t,y)$$

se resuelve usando el siguiente método:

$$y_{i+1}=y_i+\phi h$$

donde ϕ es llamada una función incremental, que puede interpretarse como la pendiente del intervalo. La función incremental se escribe en forma general de la siguiente manera:

$$\phi = a_1 k_1 + a_2 k_2 + \cdots + a_n k_n$$

donde $a_1, a_2, \cdots, a_n$ son constantes y $k_1, k_2, \cdots, k_n$ son:

$$\begin{aligned}
k_1 &= f(t_i, y_i) \\
k_2 &= f(t_i + p_1 h, y_i + q_{11} k_1 h) \\
k_3 &= f(t_i + p_2 h, y_i + q_{21} k_1 h + q_{22} k_2 h) \\
&\vdots \\
k_n &= f(t_i + p_{n-1} h, y_i + q_{n-1,1} k_1 h + q_{n-1,2} k_2 h + \cdots + q_{n-1,n-1} k_{n-1} h)
\end{aligned}$$

donde $p_1, p_2, \cdots, p_n$ y $q_1, q_2, \cdots, q_n$ son constantes. Las k's son relaciones de concurrencia, es decir, k_1 aparece en la ecuación para k_2, k_2 aparece en la ecuación para k_3 y así sucesivamente. Como cada k es una evaluación funcional, esta recurrencia hace a los métodos de Runge-Kutta eficientes como métodos numéricos para computadoras.

Diferentes métodos de Runge-Kutta se pueden usar empleando diferente número de términos en la función incremental, el cual es descrito por n. El caso más simple es para n=1, con el cual tenemos el método de Euler.

El método de Runge-Kutta de primer orden es el método de Euler. Cabe aclarar que aunque se llama de primer orden, por usar n=1, este método, así como los otros métodos de Runge-Kutta son capaces de resolver ecuaciones diferenciales de cualquier orden. Esto se logra al expresar la ecuación diferencial en un sistema de ecuaciones diferenciales de primer orden. Para el proyecto de tesis sólo hay ecuaciones diferenciales de primer orden, por lo que no se describirá el procedimiento para convertir EDO de orden superior a 1 a sistemas de EDO de primer orden.

Los métodos de segundo son del tipo:

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2) h$$

donde:

$$k_1 = f(t_i, y_i)$$
$$k_2 = f(t_i + p_1 h, y_i + q_{11} k_1 h)$$

Los valores de a_1, a_2, p_1 y q_{11} se obtienen al iguala la ecuación de y_{i+1} con una serie de Taylor de segundo orden. Al hacerlo queda el siguiente sistema de ecuaciones:

$$a_1 + a_2 = 1$$
$$a_2 p_1 = \frac{1}{2}$$
$$a_2 q_{11} = \frac{1}{2}$$

Como se aprecia, este es un sistema de 3 ecuaciones con 4 incógnitas por lo que hay un número infinito de soluciones para él, por lo que se deduce que hay un número infinito de métodos de Runge-Kutta de segundo orden. Para obtener uno, basta con asignar un valor arbitrario a alguna de las incógnitas y obtener los valores del resto. Aunque hay un número infinito de métodos de segundo orden, los más populares son:

- El método de Heun
- El método del punto medio.
- El método de Ralston

1. Método de Runge-Kutta de cuarto orden

El método de Runge-Kutta más popular es el de cuarto orden, debido a que es más preciso que los anteriores. En este caso también hay un número infinito de versiones de métodos de RK de cuarto orden. El siguiente es el más usado por lo que es llamado el método clásico de RK de cuarto orden:

$$y_{i+1}=y_i+\frac{1}{6}(k_1+2k_2+2k_3+k_4)h$$

donde

$$k_1=f(t_i,y_i)$$
$$k_2=f(t_i+\frac{1}{2}h,y_i+\frac{1}{2}k_1h)$$
$$k_3=f(t_i+\frac{1}{2}h,y_i+\frac{1}{2}k_2h)$$
$$k_4=f(t_i+h,y_i+k_3h)$$

Como se puede observar en la gráfica (ilustración 18), el método usa múltiples estimados de la pendiente para llegar a una pendiente promedio más acertada para el intervalo.

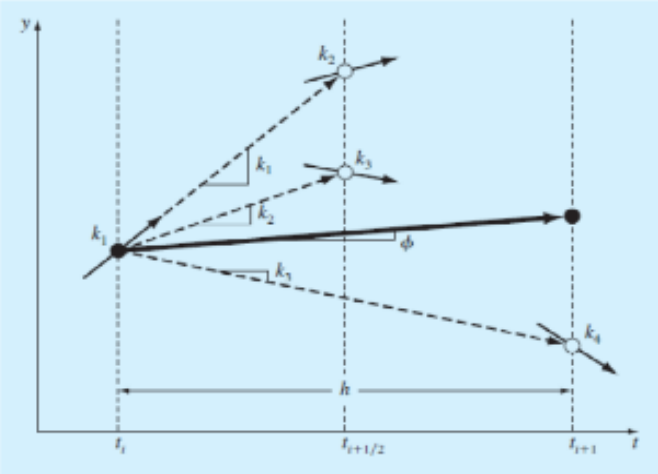


Ilustración 18: Representación gráfica de las pendientes estimadas empleadas en el método RK de cuarto orden.
Chapra, S., 2010.

Se pueden usar métodos de orden superior para obtener más precisión, sin embargo, como se observa, entre mayor es la n, más complejo se vuelve el método. Esto implica que el método tiene que hacer más operaciones para resolver la ecuación diferencial. Es por esto que los métodos de cuarto orden son populares; aunque un método de quinto orden sería más preciso, sería menos eficiente, pues tendría que hacer mucho más operaciones.

Muchos problemas prácticos de ingeniería y ciencias requieren la solución de sistemas de EDO del tipo:

$$\begin{aligned}
\frac{dy_1}{dt} &= f_1(t, y_{1,} y_{2,} \cdots, y_n) \\
\frac{dy_2}{dt} &= f_2(t, y_{1,} y_{2,} \cdots, y_n) \\
&\vdots \\
\frac{dy_n}{dt} &= f_n(t, y_{1,} y_{2,} \cdots, y_n)
\end{aligned}$$

Para resolver este tipo de sistemas se requiere que n condiciones iniciales se conozcan en el valor de inicio de t. Para el proyecto de esta tesis se aplica el método de Runge-Kutta para resolver un sistema de 3 ecuaciones diferenciales, por lo que existirían k's para cada una de ellas. Es por eso que en la sección se vio que en el programa de Fortran había un arreglo multidimensional de K con 3 filas y 4 columnas.

III. Propuesta de desarrollo

En este capítulo se analizarán todos los elementos necesarios para el desarrollo de este proyecto. Es importante tener claro desde un principio el lenguaje, herramientas y metodologías que se usarán antes de iniciar el desarrollo, para poder desarrollar un software de calidad en un tiempo razonable.

A. Elección de lenguaje de programación

En general, se puede clasificar a los lenguajes de programación en tres categorías:

1) Lenguajes compilados

2) Lenguajes interpretados

3) Híbridos entre los dos anteriores

En la primera categoría, tenemos a lenguajes como Fortran o C/C++. Estos lenguajes se distinguen porque usan un compilador que traduce el código de alto nivel a código máquina. Este código máquina generado está en el ejecutable (en Windows se distingue por la extensión *.exe*) y es dependiente de la máquina para la que se compiló. Es decir, si queremos correr un programa en un lenguaje compilado en varios sistemas, debemos compilar para cada uno de ellos. La principal ventaja de estos lenguajes es que una vez que se tiene el ejecutable este trabaja directamente con lenguaje máquina, por lo que la ejecución en general es más eficiente.

En la segunda categoría tenemos a los lenguajes interpretados, los cuales son compilados en tiempo de ejecución línea por línea. Son lenguajes en los que al ejecutar un programa cada línea es compilada y luego ejecutada. Ejemplos de lenguajes interpretados son Smalltalk, JavaScript y PHP. Estos lenguajes usan un intérprete que va ejecutando los programas línea por línea. La ventaja de esto es que el programa se vuelve independiente de la plataforma, por lo que puede correr en cualquier sistema que tenga un intérprete para el lenguaje en el que está escrito el programa. Sin embargo, estos lenguajes son en

general menos eficientes, pues cada instrucción tiene que ser interpretada antes de ejecutarse.

La tercera categoría es cuando se utilizan técnicas para hacer más eficientes los lenguajes interpretados. La mayoría de los lenguajes catalogados como interpretados usan ahora estas técnicas para solucionar el problema de la eficiencia. Generalmente lo que hacen es combinar la compilación con la interpretación. Por ejemplo, Java traduce el código a un código intermedio (*bytecode*) que interpreta la Máquina Virtual de Java (JVM), pero también usa la Compilación Justo a Tiempo (Just-in-time Compilation o JIT Compilation) para pasar a código máquina secciones de código que se usan regularmente. Los lenguajes de .NET son otro ejemplo; en dichos lenguajes se compila a código intermedio (Common Intermediate Language o CIL) que el intérprete CLR (Common Language Runtime) compila en tiempo de ejecución.

La tabla 2 muestra los lenguajes de programación considerados para hacer este proyecto.

Lenguaje de programación	**Descripción breve**	**Características**
Fortran	Fue desarrollado por IBM para usarse en aplicaciones científicas y de ingeniería con cálculos matemáticos complejos.	Optimizado para realizar operaciones sobre arreglos Popular en el ámbito científico Permite cómputo en paralelo Múltiples bibliotecas para uso científico
C	Popular por ser el lenguaje de desarrollo del sistema operativo UNIX. Se usa generalmente para programar los sistemas operativos de propósito general.	Lenguaje estructurado Eficiencia en cálculos al ser un lenguaje compilado Permite manipulación directa de la memoria con el uso de apuntadores Existen muchas bibliotecas de funciones en este lenguaje

Java	Lenguaje de programación orientado a objetos y multiplataforma	Programación Orientada a Objetos Multiplataforma Seguro y Robusto
Visual C#	Es uno de los lenguajes desarrollados por Microsoft para facilitar el desarrollo de aplicaciones para Windows	• Programación Orientada a Objetos • Usa Visual Studio como su IDE

Tabla 2: Lenguajes de programación considerados

La principal deficiencia del programa en Fortran es la falta de una interfaz gráfica amigable para el usuario. Es por ello que para la elección del lenguaje de programación se tomará en cuenta principalmente este punto. La carencia de interfaces gráficas en los programas de uso científico es muy común. Los científicos en su mayoría programan en lenguajes como C/C++ o Fortran para realizar programas que los apoyen en sus investigaciones. Estos lenguajes son los preferidos en el ámbito científico debido a que son lenguajes compilados, por lo que son más eficientes para programas que realizan muchas operaciones matemáticas.

Existen bibliotecas gráficas en lenguajes como C y Fortran que podrían permitir a los científicos dar mayor vistosidad a los programas con el uso de interfaces gráficas. Sin embargo, hay poca información sobre la implementación de interfaces con estos lenguajes, sus interfaces no son tan atractivas gráficamente y son más complicadas de implementar. Es por eso que estos lenguajes se descartan para la implementación de la interfaz gráfica.

En cuanto a los dos restantes, ambos son lenguajes híbridos, por lo que no son tan eficientes como los lenguajes compilados. Sin embargo, estos lenguajes tienen la ventaja de estar basados en el paradigma de Programación Orientada a Objetos (POO). Ésta es una gran ventaja pues la POO ayuda al programa a ser fácil de mantener y de hacer modificaciones. Además debido a la popularidad de estos lenguajes, se cuenta con

entornos de desarrollo muy potentes que pueden ser usados para acelerar aún más el proceso de la creación de interfaces gráficas.

Visual C# es un lenguaje creado por Microsoft. Esto le da ventajas y desventajas. Debido a que Windows es el sistema operativo más usado, las aplicaciones desarrolladas para este sistema operativo tienen un gran alcance de usuarios. Pero el problema es que no se pueden correr programas propios de Windows, por ejemplo los creados usando Visual C#, en otros sistemas operativos, como MacOS o Linux. Debido a que se pretende que el programa tenga un número grande de usuarios, se debe tomar en cuenta el hecho de que la aplicación debe correr en el mayor número de máquinas posible. Visual C# nos limitaría a máquinas con Windows, por lo que se descarta como opción.

El único que falta analizar y que será el elegido para la aplicación de vulcanología, es Java. Este lenguaje, que como se mencionó tiene el paradigma de programación orientada a objetos, además tiene la ventaja de ser multiplataforma. Un programa escrito en Java puede ser ejecutado en cualquier computadora, siempre y cuando ésta tenga instalada la Máquina Virtual de Java. Esto le da a Java un gran alcance, pues al menos para los sistemas operativos más populares (Windows, MacOS y las variantes de Linux) existe una implementación de la Máquina Virtual de Java. Es por esto que se decide usar este lenguaje de programación, así como por otras características de las que se hablará más adelante.

B. Lenguaje de programación Java

El lenguaje de programación Java fue liberado por Sun en 1995. Es un lenguaje de programación concurrente, basado en clases y orientado a objetos. Fue creado con el lema "write once, run anywhere" (WORA), es decir, está pensado para que los programas corran en cualquier máquina sin la necesidad de compilar para cada una de ellas. Para esto se usa la Java Virtual Machine (JVM) que tiene que estar instalada en la máquina en donde se desee correr un programa de Java.

A continuación se muestran algunas estadísticas que publica Oracle en la página oficial de Java (http://www.java.com/es/about)

- El 97% de los escritorios empresariales ejecutan Java
- El 89% de los escritorios (o computadoras) en Estados Unidos ejecutan Java
- 9 millones de desarrolladores de Java en todo el mundo
- La primera opción para los desarrolladores
- La primera plataforma de desarrollo
- 3 mil millones de teléfonos móviles ejecutan Java
- El 100% de los reproductores de Blu-ray incluyen Java
- 5 mil millones de Java Cards en uso
- 125 millones de dispositivos de televisión ejecutan Java
- 5 de los 5 principales fabricantes de equipos originales utilizan Java ME

El proyecto del lenguaje Java fue iniciado en 1991 por James Gosling. Mike Sheridan y Patrick Naughton. Su nombre inicial fue Oak, pero al final fue llamado Java, nombre tomado del café al que muy a menudo iban los creadores del lenguaje. La primera versión liberada fue Java 1.0 en 1995. En 1998 se liberó la versión de Java 2, a partir de la cual ya había versiones diferentes para diferentes tipos de plataformas: J2SE (Standard Edition), J2EE (Enterprise Edition) y J2ME (Micro Edition). Entre 2006 y 2007 Sun liberó la mayor parte del código de Java usando la licencia GNU GLP (General Public License) con lo que lo

hizo Free and Open Source Software (FOSS). Entre el 2009 y el 2010 Oracle adquirió la compañía de Sun Microsystems con lo que adquirió entre otras tecnologías a Java.

1. Versiones de Java

La tabla 3 muestra las diferentes versiones de Java con una breve descripción de sus características más importantes.

FECHA DE LIBERACIÓN	VERSIÓN	DESCRIPCIÓN
1995	JDK Alfa y Beta	Fueron las primeras versiones públicas de Java que se usaron para depurar, por lo que eran aún muy inestables
23 de enero de 1996	JDK 1.0	Originalmente llamado Oak, fue la primera versión estable de Java.
19 de febrero de 1997	JDK 1.1	Se agregó lo siguiente: • mejora del modelo de eventos de AWT, • permite uso de clases internas • Se agrega paquete para usar JavaBeans (componentes reutilizables de Java) • Se agrega JDBC (acceso a base de datos) • Se agrega RMI (Remote Method Invocation, para programar arquitecturas cliente-servidor) • Se agrega compilador JIT (just-in-time) en plataformas de Windows, lo que mejora el rendimiento de los programas.
8 de diciembre de 1998	J2SE 1.2	A partir de esta versión se le llama a la versión como Java 2 Standard Edition (J2SE) y se continua con este estándar hasta la versión J2SE 5.0. Las mejoras más importantes fueron: La API de Swing se integró al lenguaje El compilador JIT se integró a la JVM por primera vez. Se agrega Java IDL (Interface Definition Language) que es una implementación de CORBA en Java. Se agrega JCF (Java Collections Framework) que es un conjunto de clases e interfaces que implementan colecciones reutilizables de estructuras de datos (Listas, Pilas, Colas, etcétera)

8 de mayo de 2000	J2SE 1.3	Las mejoras más importantes fueron: Hotspot incluido en JVM RMI modificado para soportar compatibilidad con CORBA Se incluye JNDI (Java Naming and Directory Interface) Se incluye JPDA (Java Platform Debugger Architecture) Se incluye JavaSound
6 de febrero de 2002	J2SE 1.4	Fue la primera versión desarrollada a cargo del Java Community Process. Las mejoras que incluyó fueron: mejora en expresiones regulares permite encadenamiento de excepciones soporte para IPv6 permite entrada y salida con non-blocking se agrega API para E/S de imágenes en JPEG y PNG se integra parser de XML Se incluyen extensión de seguridad y criptografía (JCE, JSSE, JAAS) Se incluye Java Web Start
30 de septiembre de 2004	J2SE 5.0	Se cambia el formato de numeración de las versiones. Se agregó lo siguiente: Generics: eliminan necesidad de la mayorías de casts Metadata: se pueden agregar etiquetas con datos adicionales para ser procesados como metadata Autoboxing: conversión automática entre tipos primitivos y sus clases respectivas (por ejemplo convertir int en Integer) Enumeraciones: con la palabra clave enum se puede definir listas de valores sin necesidad de usar constantes para mayor claridad en código Varargs: se ponen con tres puntos seguidos (...) y permiten mandar a una función una cantidad variable de parámetros For each: es una sentencia for mejorada para iterar sobre elementos de un arreglo (o de un objeto Iterable) que no van a ser modificados.

11 de diciembre de 2006	Java SE 6	Se quita el .0 de el número de versión. Se agregó lo siguiente: Soporte para versiones viejas de Windows Soporte para Scripting Languages Mejoras de rendimiento en Swing Se mejora la funcionalidad de Web Service
2007- 2014	Java 6 updates	A esta versión se le dio soporte durante varios años sin hacer cambios en la API. Estas modificaciones se enfocaron en: Actualizaciones de seguridad Mejorar la usabilidad para el usuario final
28 de julio de 2011	Java SE 7	Las mejoras de esta versión son: Se pueden usar Strings en switch Manejo de recursos en sentencias try Catch de excepciones múltiples Utilidades para la concurrencia Biblioteca de E/S para mejorar la independencia de plataforma. Se mejora Java 2D para aprovechar las capacidades de las nuevas GPUs
2011-2014	Java 7 updates	Las modificaciones se enfocaron en: Actualizaciones de seguridad Mejorar la usabilidad para el usuario final Corrección de bugs
18 de marzo de 2014	Java SE 8	Se permite a los programadores agregar código de JavaScript en las aplicaciones de Java Lambda expressions: permiten usar el código como datos. Mejoras de rendimiento en HashMaps Mejoras de seguridad

Tabla 3: Versiones de Java

2. La API de Java

Una API (Application Programming Interface) define cómo van a interactuar los componentes de software entre ellos y cuenta con un conjunto de funciones y procedimientos que el usuario del lenguaje puede usar. En Java, la API viene organizada en paquetes, dentro de los cuales se encuentran las clases que pertenecen a ese paquete. En otros lenguajes, generalmente se usan bibliotecas de funciones para definir la API por lo que si se quiere usar una biblioteca, se incluye en el programa (en C por ejemplo se usa la sentencia *include*). Pero como Java es un lenguaje orientado a objetos, la API se define a través de clases, que si se quieren usar deben ser importadas al programa usando la sentencia *import*.

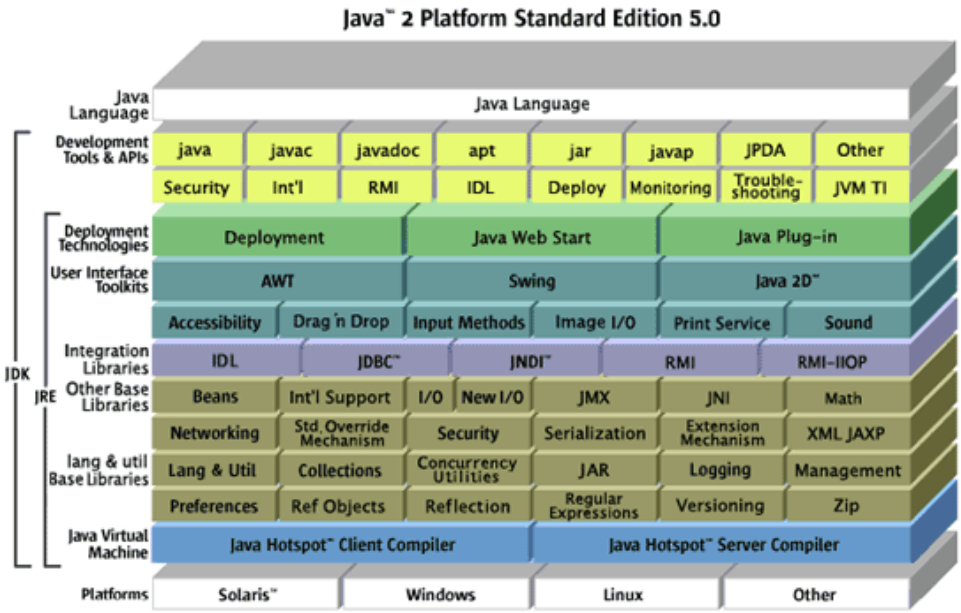


Ilustración 19: API de Java (J2SE 5.0). http://www.cs.mun.ca/java-api-1.5/

La ilustración 19 muestra la multitud de paquetes que contiene la API de Java. Como se puede observar existen paquetes para propósitos muy específicos. Para este proyecto son especialmente importantes los que se muestran en la categoría User Interface Toolkits (herramientas para interfaz de usuario).

3. Máquina Virtual de Java

La máquina virtual de java es la tecnología responsable de la independencia que tiene Java con respecto al hardware y al sistema operativo en el que corre, el tamaño pequeño de su código compilado y la habilidad para ofrecer seguridad a los usuarios contra programas maliciosos. Se trata de una computadora virtual, que al igual que una computadora física tiene un set de instrucciones y manipula la memoria para correr programas.

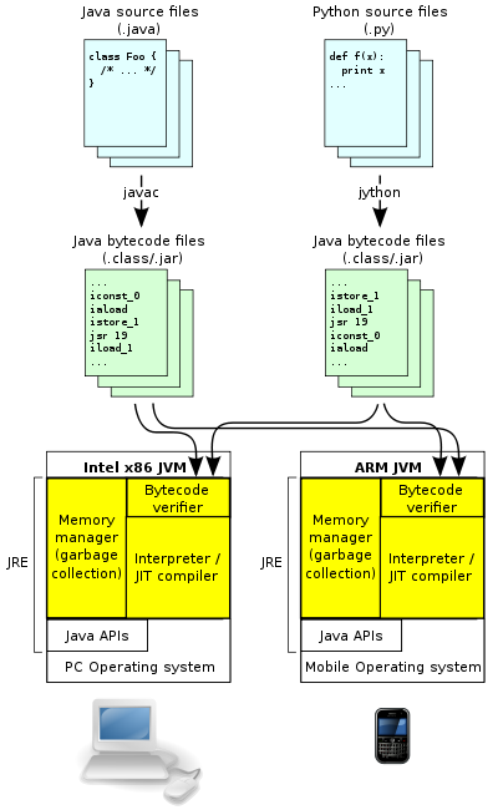


Ilustración 20: Arquitectura JVM. http://en.wikipedia.org/wiki/Java_virtual_machine

La Máquina Virtual de Java no entiende el lenguaje de Java; lo que entiende es el formato binario del tipo de archivo con extensión *.class,* el cual contiene el *bytecode* del programa. El *bytecode* es un código intermedio que la JVM compila justo antes de ejecutarse con el compilador JIT (ilustración 20).

El set de instrucciones de la JVM tiene instrucciones que entran en las siguientes categorías:

- Load and Store
- Type conversion
- Object creation and manipulation
- Operand stack management (*push*/*pop*)
- Control transfer (branching)
- Method invocation and return
- Throwing exceptions
- Monitor-based concurrency

La JVM verifica el *bytecode* antes de que éste se ejecute. Esto ayuda a prevenir errores graves del sistema, pues el código que podría causar errores nunca es ejecutado por la máquina física.

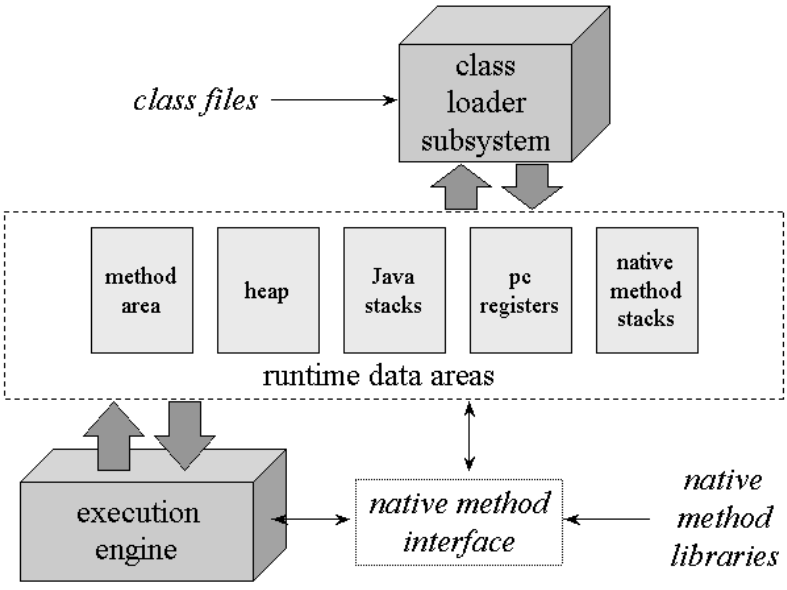


Ilustración 21: Arquitectura interna de la JVM.
http://www.artima.com/insidejvm/ed2/jvm2.html

La ilustración 21 muestra la arquitectura interna de la JVM. Sus componentes son:

- Class Loader Subsystem.- es el encargado de cargar las clases de Java en la máquina virtual
- Method area.- Es la sección de memoria en donde se almacena información de las clases
- Heap area.- Es la sección de memoria en donde se almacena información de los objetos
- Java stacks.- Para cada hilo, en el Java stack se guarda el estado de las invocaciones de métodos.
- PC registers.- Para cada hilo que se ejecuta, hay un PC register que tiene la dirección de memoria de la siguiente instrucción que se va a ejecutar.
- Native Java stacks.- Es lo mismo que el Java stack pero para métodos nativos que son dependientes de la implementación

Aunque al programar en Java nunca nos tendremos que preocupar por todos estos detalles sobre la JVM, es importante conocerlos para entender cómo trabaja. Es principalmente gracias a la JVM que Java tiene tantas ventajas sobre otros lenguajes, como la seguridad, la robustez y ser un lenguaje multiplataforma. Pero también hay otras características que Java tiene, como la facilidad de mantenimiento y la reutilización de código, que no son producto de la JVM, sino del paradigma sobre el que se diseñó Java.

C. Programación orientada a objetos

La programación orientada a objetos, abreviada POO, es un paradigma de programación que usa objetos para representar los datos (atributos en POO) y procedimientos (métodos en POO) de un programa. Se trata de un paradigma que tiene como objetivo permitir el diseño de sistemas de software modulares y reutilizables. El paradigma orientado a objetos está basado en conceptos como la programación estructurada y los tipos abstractos de datos. Un objeto es un tipo abstracto de dato que tiene como agregados el polimorfismo y la herencia. En la programación estructura se hace una separación entre código y datos, cuando en el programación orientada a objetos ambos conceptos se unen para forma un objeto, el cual tiene un estado (atributos) y un comportamiento (métodos).

Algunas de las ventajas que ofrece la programación orientada a objetos son:

- Permite definir el programa de una manera más natural con el uso de objetos que describen el problema
- Facilitar el mantenimiento del software al hacer programas más claros.
- Facilitar la evolución, extensión o mejoramiento del software por su modularidad.
- Fomenta la reutilización del código al programarse en módulos que pueden ser usados por otros programas.
- Ahorra tiempo de desarrollo al permitir desarrollar en paralelo módulos independientes del software

Los conceptos en los que se basa este paradigma se empezaron a desarrollar desde la década de 1950, pero fue hasta la década de 1960 cuando el concepto de objeto en programación tomó forma. Simula 67, un lenguaje de programación desarrollado en el Centro de Cómputo Noruego en Oslo, fue el primer lenguaje de programación en aplicar el concepto de objeto. Smalltalk fue desarrollado durante la década de 1970 por Xerox PARC y fue el primero en introducir el término Programación Orientada a Objetos para

representar el uso de objetos y mensajes en un programa de computadora. Después de Smalltalk, muchos lenguajes existentes adoptaron la POO como paradigma de programación y se crearon otros lenguajes partiendo de la POO.

Ejemplos de lenguajes que usan la POO son:

- C++
- Objective-C
- Smalltalk
- Java
- C#
- Perl
- Python
- Ruby
- PHP

Algunos conceptos relacionados a la programación orientada a objetos son:

Clase.- Definición de los atributos y el comportamiento de un tipo de objeto.

Instancia.- Se trata de la creación de un objeto a partir de las reglas definidas en la clase.

Atributos.- Son las características del objeto (datos) que se definen en la clase.

Métodos.- Son subrutinas asociadas a un objeto y que se definen en la clase para describir el comportamiento del objeto.

Abstracción.- Es la definición de un objeto en base a sus características esenciales (atributos y comportamiento). La abstracción es una de las características más importantes en el diseño orientado a objetos, pues permite modelar un problema en base a un conjunto de clases.

Modularidad.- Al crear clases, el software se separa naturalmente en módulos, que pueden ser utilizados como parte de varios programas.

Mensajes.- Es la comunicación hacia un objeto que le indica que ejecute un método con los parámetros indicados en el mensaje.

Las características básicas que un lenguaje de programación debe tener para entrar en el paradigma orientado a objetos son: encapsulamiento, herencia y polimorfismo.

1. Encapsulamiento

El encapsulamiento se refiere a agregar dentro de un objeto todos los recursos necesarios para que el objeto funcione, esto es, agregar los métodos y los atributos que requiere. La idea del encapsulamiento es que el código no influencie en otro código, ni sea influenciado por otro código. Las clases permiten que los datos y los métodos estén estrechamente ligados, formando una "cápsula" congruente con el objeto que se está definiendo.

El encapsulamiento es usado para esconder los valores de los atributos dentro de una clase, de modo que sólo el objeto pueda tener acceso a ellas y no lo tengan otros objetos externos. Para esto los atributos deben ser definidos como privados, es decir, sólo los métodos de la clase pueden acceder a ellos o modificarlos. Para acceder a los atributos desde otras clases se proporcionan en la clase métodos específicos para esto:

- Para consultar el valor de un atributo desde una clase externa se tiene que definir un método *get* para este atributo.
- Para modificar el valor de un atributo desde una clase externa se tiene que definir un método s*et* para este atributo.

Un ejemplo puede ayudar mucho para entender cómo funciona esto. Un ejemplo sencillo que puede ser la clase Vehículo. Para la clase Vehículo tenemos que definir sus atributos y sus métodos. Debido a que una clase es una abstracción no es necesario incluir todos los

atributos y métodos (comportamientos) que se pueden encontrar en un vehículo real. Tomaremos los que consideremos importantes. Para este caso se elegirán los atributos: posición y color. Para sus métodos escogeremos sólo el método mover. En este caso, para nuestros propósitos estos elementos son suficientes para describir un vehículo. Es importante que cuando usemos la clase vehículo, la usemos sólo a través de sus métodos, para no cambiar directamente sus atributos, pues esto puede generar errores. Por ejemplo, se tiene el método mover, el cual modificará la posición de acuerdo a la implementación que se haya hecho del método. En la clase vehículo consideraremos cómo es que se tendrá que mover o con qué reglas para que la posición se actualice correctamente. Tanto para posición como para color, se pueden agregar métodos *get* y *set* para poder modificarlos. La ventaja de usar estos métodos, es que podemos implementar en ellos medidas de prevención para que el usuario de la clase no ponga valores no válidos.

2. Herencia

La herencia es una de las características más poderosas de la programación orientada a objetos, pues fomenta la reutilización del código. La herencia se refiere a que se pueden crear clases que son *hijas* de otra clase. A la clase *padre* se le llama *superclase*, la cual contiene atributos y métodos comunes para todas las clases *hijas*. Las clases *hijas* se llaman *subclases* y son las que heredan estos elementos comunes de las *superclases*, pero contienen también atributos y métodos propios. De esta manera, se puede ahorrar mucho código por ejemplo al implementar métodos una sola vez en la *superclase*, además de que el código es más fácil de mantener, pues sólo se modifica en una parte y esta modificación pasa automáticamente a las *subclases*. Es importante mencionar que los métodos heredados por las *subclases* pueden ser reimplementados por ellas para que se adecuen a la *subclase* en específico.

La *superclase* directa es aquella que está explícitamente declarada como *superclase* de la *subclase*. En Java se utiliza la palabra reservada *extends* para indicar esto. Sin embargo,

también existen *superclases* indirectas, que son aquellas que no aparecen explícitamente en la clase que hereda, pero que aparecerá en alguna clase de la jerarquía de clases. La jerarquía en Java siempre empieza con la clase *Object*, es decir, todas las clases en Java son *subclases* de la clase *Object*, ya sea directa o indirectamente. En Java sólo se puede heredar directamente de una clase.

En el ejemplo con la clase Vehículo podríamos crearle dos *subclases*: la clase Carro y la clase Avión. En la clase Vehículo teníamos los atributos posición y color, y el método mover. Un carro y un avión comparten estas características, por ello cuando creemos las clases indicaremos que heredan de Vehículo. Pero además, cada uno de ellos puede tener atributos y métodos propios y para el método mover, cada uno podría tener su propia implementación, pues un carro y un avión en la vida real, no se mueven de la misma manera, por lo que en el programa se tendría que tomar esto en consideración. Este último punto sobre la implementación de métodos propios en la *subclase*, tiene mucho que ver con el concepto de polimorfismo, que se verá a continuación.

3. Polimorfismo

El polimorfismo está muy ligado a la herencia. El polimorfismo se refiere a que dentro del código se puede tratar a todas las *subclases* como si se tratara de su *superclase*. Esto simplifica mucho la programación y la hace más clara. El enfoque que se da es el de tratar los problemas de manera general y no de manera específica.

Para el ejemplo de la clase Vehículo y las *subclases* Carro y Avión, esto se podría explicar de la siguiente manera. Por ejemplo, digamos que tenemos un método en nuestro programa principal se recibe un objeto de tipo Vehículo. Aunque el método se define como que recibe objetos del tipo Vehículo, si le mandamos un objeto de tipo Carro o Avión, no fallará, pues al ser *subclases* de Vehículo, los considera vehículos. Supongamos que dentro del método se manda a llamar al método mover del objeto. En caso de que se trate de un objeto de tipo Carro, se mandará a llamar al método mover de la clase Carro, y

si es un objeto Avión, se mandará a llamar al método mover de la clase Avión. Es decir, con la misma llamada del método, se tienen diferentes respuestas. Eso es el polimorfismo.

Otra de las formas donde se implementa el polimorfismo es dentro de una misma clase. Para este caso, lo que hacemos es implementar varios métodos con un mismo nombre. Este concepto es llamado sobrecarga de métodos. Cuando un método de este tipo se ejecuta en un programa, Java identifica de cuál se trata por el número de parámetros que cada uno recibe. Entonces por ejemplo, para el método mover de nuestro programa, podríamos tener un método que no reciba nada y otro que reciba la velocidad a la que se va a mover.

Ahora queda claro que Java debe en gran medida su éxito a la Programación Orientada a Objetos. Como se comentó en esta sección, las mayores ventajas de este paradigma de programación son brindar claridad al código, de manera que sea más fácil desarrollar y mantener aplicaciones. Éste es un punto muy importante que se tomó en cuenta para la decisión del lenguaje de programación, pues un lenguaje que trabaje con este paradigma logrará que el programa pueda servir por más tiempo, al ser más fácil adaptarlo en caso de requerir cambios. Ahora el punto importante es decidir qué herramientas usar para crear la interfaz gráfica. Para este punto se tendrá que decidir el entorno de desarrollo con el que se va a trabajar, tema que se tratará en la siguiente sección.

D. Entorno de desarrollo

Los Entornos Integrados de Desarrollo, abreviado IDE por sus siglas en inglés (Integrated Development Environment) son aplicaciones de software que ayudan al programador en el desarrollo de software. Generalmente todos los IDEs contienen al menos un editor de código fuente, herramientas de compilación y un debugger (depurador). Actualmente, los IDEs cuentan con más herramientas, como por ejemplo función de autocompletar el código mientras se va escribiendo.

Los entornos de desarrollo han sido pensados para aumentar la productividad del programador. Esto se puede lograr al integrar en un sólo programa de desarrollo las herramientas más comunes que usa el programador. Esto reduce el tiempo de configuración antes del desarrollo de un proyecto.

Algunos IDEs son específicos para un lenguaje en particular, aunque la mayoría soportan múltiples lenguajes. Los IDEs facilitan generación de interfaces gráficas de usuario. Muchos IDEs permiten al programador crear interfaces gráficas arrastrando elementos de la interfaz y definiendo sus características visualmente, sin la necesidad de escribir código. Debido a que este proyecto se hará usando Java, hay que decidir qué entorno de desarrollo para Java utilizar. Para este proyecto se usará Netbeans, pues facilita en gran medida la creación de interfaces gráficas, además de tener otras características que se mencionarán a continuación.

Netbeans es un entorno de desarrollo escrito en Java que permite programar en múltiples lenguajes de programación. Cuenta con diferentes Plugins que se pueden descargar dependiendo de las herramientas que necesitemos para nuestro desarrollo. Dichos Plugins son desarrollados por la comunidad de Netbeans, e incluyen varias categorías, por ejemplo:

- soporte para diversos lenguajes de programación

- herramientas de edición
- soporte para bases de datos
- depuración
- herramientas de control de versiones
- desarrollo web
- utilidades varias

Estos Plugins facilitan al programador varias tareas. En el caso de este proyecto, se utilizó un Plugin para Mercurial, sistema de control de versiones, de modo que el control de versiones se pueda llevar a cabo desde el mismo entorno de desarrollo. Sobre este sistema de control de versiones se hablará en la sección .

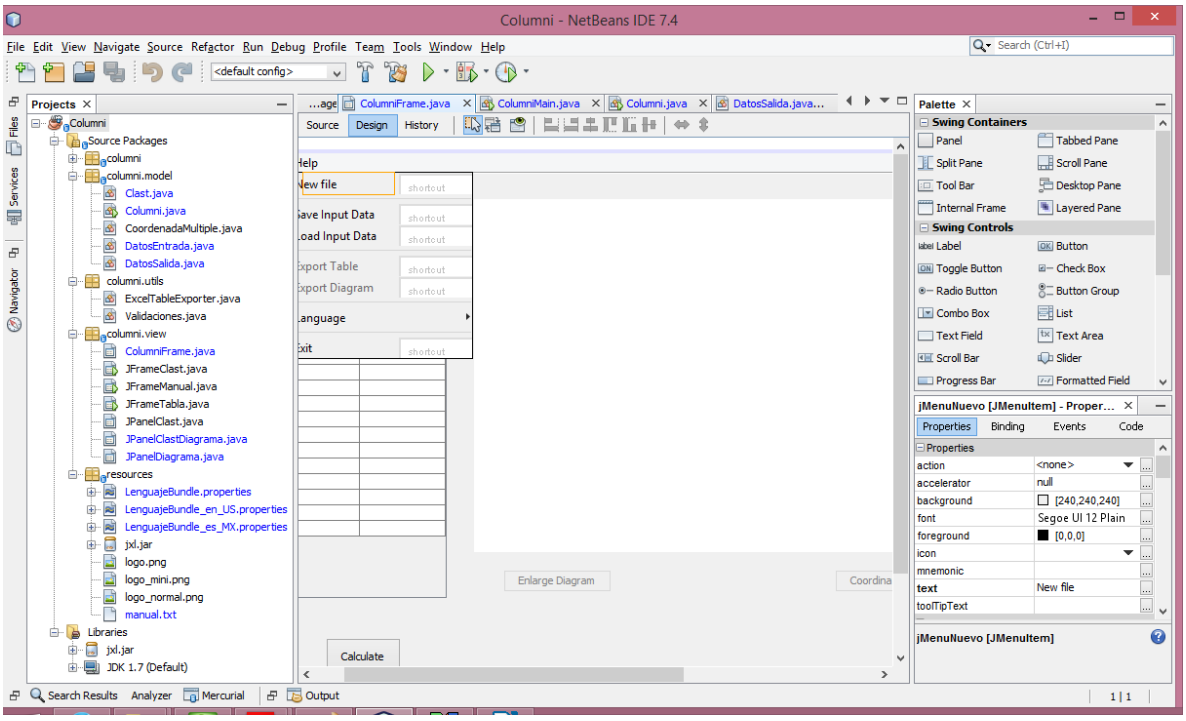


Ilustración 22: Entorno de desarrollo Netbeans

En la ilustración 22 se observa una interfaz gráfica creada con Netbeans. Como se puede observar, del lado derecho se encuentran todos los elementos disponibles para implementar una interfaz gráfica. Basta con arrastrar estos elementos hacia la pantalla del centro para ir acomodándolos dentro de la interfaz. Hay que resaltar que como se mencionó, el IDE es especialmente útil para manejar elementos de la interfaz gráfica, pues se pueden agregar y editar de manera visual. Esto nos ahorra tiempo en el desarrollo de la Vista, un elemento de la arquitectura MVC, de la que se hablará en la siguiente sección.

E. Arquitectura propuesta – Modelo Vista Controlador

Para la arquitectura del sistema, se eligió la arquitectura Modelo Vista Controlador (MVC). Esta arquitectura está pensada para el desarrollo de interfaces gráficas de usuario. Con esta arquitectura, se puede llevar un mejor control de los elementos que componen al sistema, de modo que sea más fácil el desarrollo y mantenimiento del mismo. Este tipo de arquitectura o patrón de diseño está muy ligado a la programación orientada a objetos, pues se basa en la abstracción y también en la reutilización de código.

El MVC tiene tres componentes que interactúan entre sí: el modelo, la vista y el controlador (ver ilustración 23).

Modelo

Representa los datos de la aplicación y las reglas, la lógica y las funciones. El modelo notifica a la vista y al controlador cuando hay un cambio en el estado de la aplicación. que gobiernan el acceso a y la actualización de los datos.

Vista

Especifica cómo deben ser presentados los datos del modelo. Si los datos del modelo cambian, la vista debe actualizar la presentación de los datos en el momento. Para esto se puede usar un *push model*, donde el modelo manda los cambios en todo momento, o un *pull model*, donde la vista es responsable de llamar al modelo cuando necesite actualizar a los datos más recientes.

Controlador

Se encarga de interpretar las acciones que el modelo debe efectuar, cuando el usuario interactúa con la Vista. Estas interacciones son los clics del mouse, una selección de menú o un GET en un mensaje de HTTP por ejemplo, dependiendo de la aplicación de la que se trate.

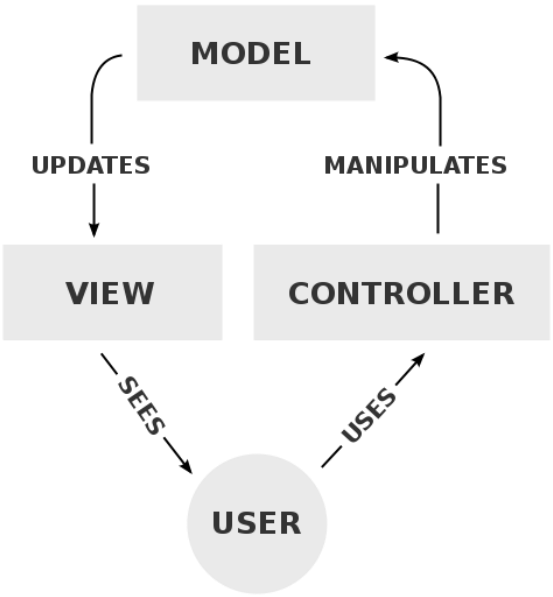


Ilustración 23: Modelo Vista Controlador.
http://en.wikipedia.org/wiki/Model%E2%80%93view
%E2%80%93controller

Un ejemplo de la aplicación del MVC en Java está en sus mismas bibliotecas de gráficos. Tal es el caso de Swing, una biblioteca gráfica de Java, en la que los componentes se comportan de acuerdo al MVC como se muestra a continuación

Modelo.- Cada componente de Swing tiene modelos asociados, que pueden ser usados por el programador tal cual están, o implementar sus propios modelos usando las *interfaces* de Java. Por ejemplo, para un botón en Java, el modelo definiría los eventos (presionar, liberar, etcétera) y las propiedades del botón (color, texto, etcétera).

Vista.- En un componente de Swing, la vista sería lo que ve el usuario en pantalla y está implementado en su mayoría en los métodos paint() de los componentes.

Controlador.- El controlador se encargará de detectar los eventos que surjan tras la interacción del usuario con la vista. Esto eventos son detectados al implementar *Listeners* de Java para los eventos específicos.

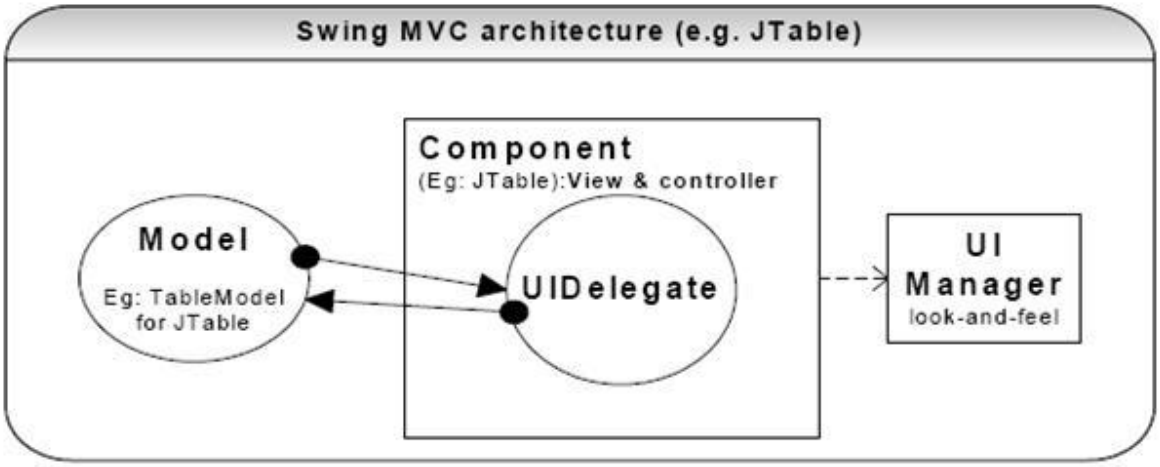


Ilustración 24: Arquitectura MVC de Swing. http://www.java-forums.org/blogs/java-swing/775-jtable-mvc.html

Como se puede observar, en el caso de Swing (ilustración 24) la vista y el controlador están muy ligados, por lo que se encuentran incluso en la misma clase.

Debido a que se trata de un patrón de diseño abstracto, es posible implementarlo de diferentes maneras. En Java, una de las implementaciones que se puede hacer es creando directamente las clases para el Modelo, la Vista y el Controlador. Esto aplicaría para un proyecto pequeño, en el que no hay muchos elementos que manejar y en el que fueran suficientes estas tres clases para todo el sistema. Pero para un sistema más grande, este enfoque no sería conveniente.

La alternativa es usar paquetes para organizar las clases dentro de las categorías de Modelo, Vista y Controlador. Éste es el enfoque que se usará para este proyecto, pues como se verá más adelante en el capítulo 4, , el proyecto tiene un gran número de clases. Si no se llevará este orden, sería muy difícil manejar tantas clases. Sin embargo, cabe mencionar que el paquete para la Vista contendrá el código tanto para la Vista como para el Controlador, pues como vimos anteriormente, ambos elementos están muy ligados.

F. Modelo de desarrollo

Para hacer un desarrollo efectivo de un programa, no sólo es necesario saber programar, también es importante tener conocimiento de las técnicas de la ingeniería de software. La ingeniería de software es la aplicación de la ingeniería al diseño, desarrollo y mantenimiento del software. Se podría decir que es el conjunto de conocimientos que se han ido adquiriendo en las recientes décadas sobre las mejores prácticas para el desarrollo de software. Desde el punto de vista de la ingeniería de software, es importante elegir un modelo de desarrollo antes de empezar a escribir el código.

Un modelo de desarrollo es una serie de actividades relacionadas que van a llevar a la producción de un producto de software. Dentro de los diferentes modelos que existen, todos deben incluir las siguientes cuatro actividades:

- Especificación del software: también llamada la etapa de análisis, es donde se identifican los problemas que han de resolverse y se hace una especificación de requerimientos.

- Diseño e implementación del software: en esta etapa se crea el producto de software. En ocasiones es más claro separar esta etapa en dos, para identificar lo que se debe hacer en cada etapa.

- Validación del software: se llama validación a las pruebas que hace el usuario final (cliente) al software para asegurarse de que cumple con los requerimientos.

- Evolución del software: esta etapa es llamada también de operación y mantenimiento. Conforme el software se use puede haber modificaciones para cumplir requerimientos que no se habían especificado en un principio.

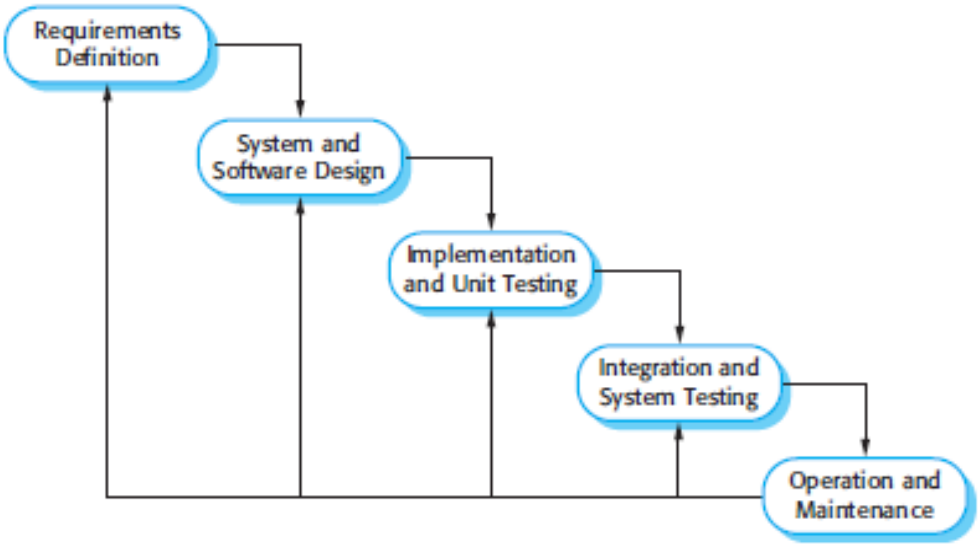


Ilustración 25: Modelo de cascada. Sommerville, I., 2011.

El modelo que se utilizará en este proyecto es el modelo de cascada cuyas etapas se muestran en la ilustración 25. En este modelo se planean las actividades desde un principio y una vez definido todo el proceso, se desarrollan las actividades. En este modelo las actividades de desarrollo fundamentales son:

- Análisis y definición de requerimientos: Se debe hacer un análisis a detalle de los requerimientos del sistema. En esta etapa es importante consultar al usuario final del sistema para identificar sus necesidades y poder hacer una especificación de requerimientos detallada.

- Diseño del sistema: usando la especificación de requerimientos se diseña el sistema tomando en cuenta los requerimientos de hardware y software. En esta etapa se diseña la arquitectura del sistema, de modo que al iniciar con el desarrollo (implementación) se tenga un esquema sobre el cual trabajar.

- Implementación y pruebas unitarias: se realiza el desarrollo del sistema a través de la creación de los diferentes elementos que lo formarán. En esta etapa se prueban los componentes por separado con las pruebas unitarias.

- Integración y pruebas del sistema: en esta etapa ya todos los elementos se han probado individualmente y se unen para formar el sistema. En esta etapa se prueba el sistema completo y se corrigen errores que se pudieran producir durante la integración. Después de las pruebas se libera el software al cliente o usuario final.

- Operación y mantenimiento: en esta etapa el software es usado por el usuario final, que es a lo que se refiere la operación. El mantenimiento implica la corrección de errores o identificación de requerimientos que no fueron descubiertos durante las otras etapas.

En la sección , usaremos este modelo para desarrollar la aplicación en Java que analiza una columna eruptiva.

G. Control de versiones

Durante todas las etapas del proceso de desarrollo es importante documentar los cambios que se van haciendo conforme se va avanzando en el proyecto. De esta manera se puede cuantificar los avances conforme transcurre el tiempo y saber si se tendrá el software en el tiempo estimado. Esto se puede realizar simplemente llevando una bitácora de los cambios que se van haciendo, aunque hay herramientas que permiten hacer esto y agregan otras características interesantes. A este tipo de herramientas se les llama sistemas de control de versiones o System Version Control (SVC) en inglés. Con estos sistemas podemos ir documentando el avance del proyecto, pero además nos permite hacer un regreso a una versión anterior en caso de que se requiera. Entre los SVCs más populares tenemos:

- CVS (Control Version System): muy popular para desarrollo de software libre. Usa la licencia GPL.
- Subversion: esta herramienta es de código abierto y muy popular para el desarrollo de software de código abierto
- Team Foundation Version Control: es una herramienta que viene integrada en la suite de desarrollo de Visual Studio de Microsoft.
- Bazaar: esta herramienta es patrocinada por canonical ltd, la empresa creadora de Ubuntu.
- Git: este SVC fue diseñado por Linus Torvalds, el creador del núcleo de Linux, pensado para proyectos de código abierto en donde el número de archivos es muy extenso y también para el desarrollo colaborativo.
- Mercurial: originalmente creado para Linux, pero ahora funciona en MacOS y Windows también. Está pensado para hacer un desarrollo distribuido, que generalmente se da en proyectos de software libre o de código abierto.

1. Mercurial

Para este proyecto se usó mercurial, pues este proyecto podría ser un buen candidato para ser de código abierto. Un proyecto de código abierto es aquel en el que la licencia permite al usuario modificar el código y compilarlo para crear su propia versión de él. Los proyectos de código abierto se benefician de la interacción de múltiples programadores que pueden aportar ideas y colaborar a través de Internet. Una ventaja de mercurial es que está integrado con Netbeans, lo que lo hace fácil de usar si ya se conoce la plataforma de Netbeans.

Para llevar el control de versiones simplemente hay que ir haciendo *commits*, conforme se va avanzando con el proyecto. Los *commits* indican al Mercurial que se deben considerar los cambios que se hicieron al programa como definitivos, por lo que el *commit* creará una versión del programa. Para hacer un *commit* desde Netbeans se da clic derecho sobre el nombre del proyecto y en *Mercurial->Commit*.

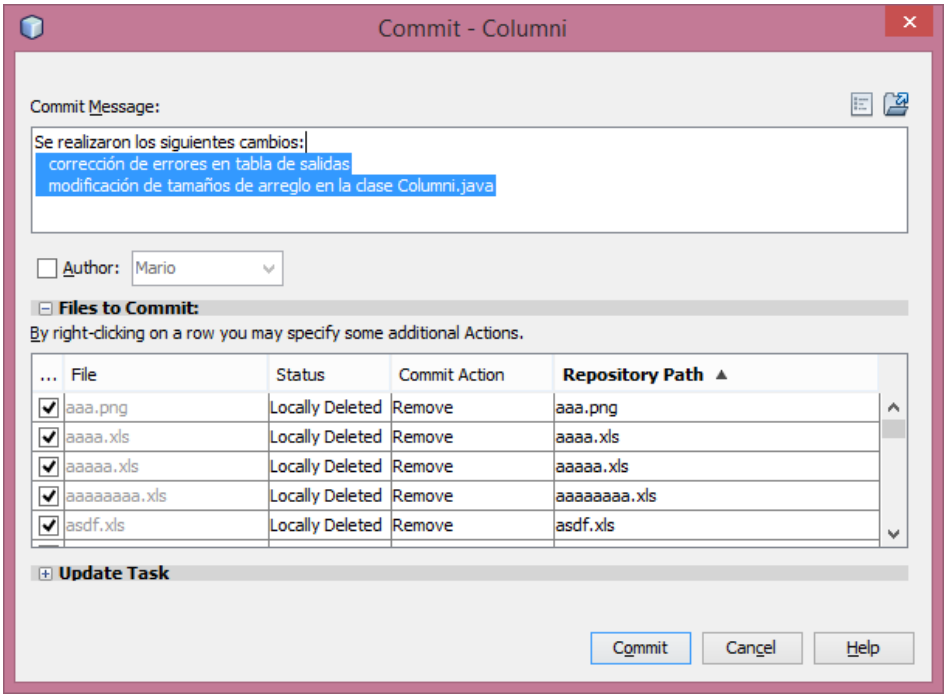


Ilustración 26: Realización de un commit

Al hacer un *commit* se debe agregar una descripción de los cambios que se hicieron (ver ilustración 26) para que quede registrado en la bitácora. Esta descripción debe ser breve y concisa para que en caso de que el programador tenga que consultar alguna de las versiones pueda ver rápidamente que cambios se realizaron en ella.

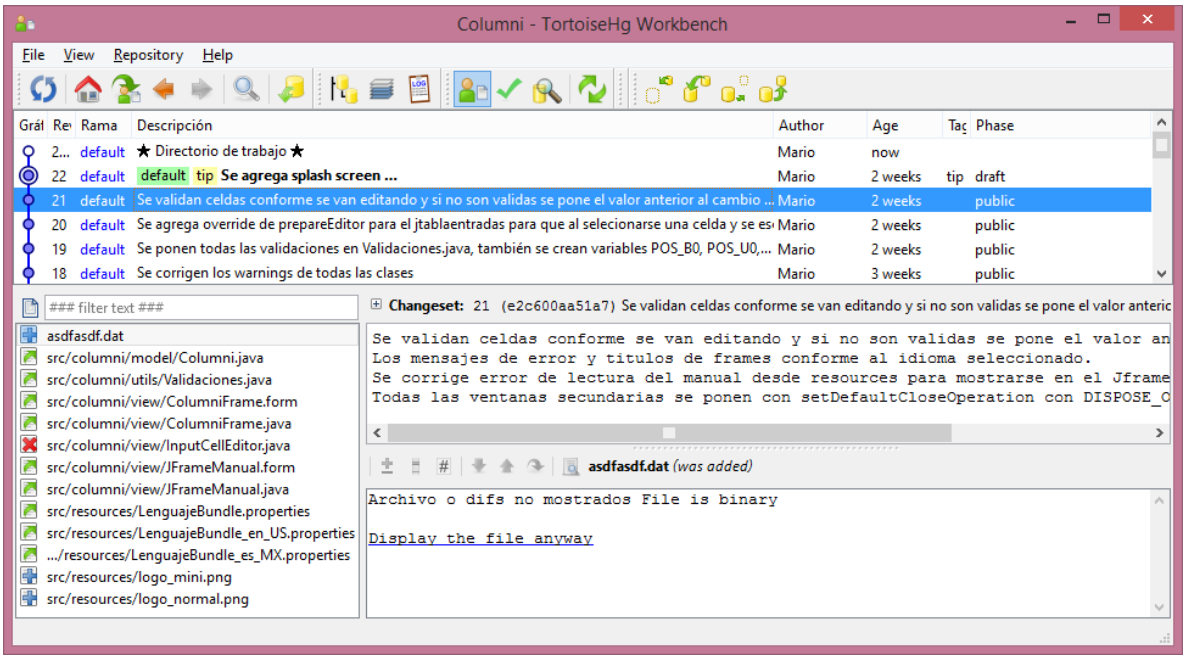


Ilustración 27: Histórico de versiones

Al consultar el histórico de las versiones (ilustración 27), se puede ver la descripción que se puso y las diferencias entre los archivos de esa versión y la versión anterior. Si se usa un repositorio, todos estos datos se podrán consultar también desde ahí.

H. Repositorio

Además del control de versiones, es importante tener un método de respaldo para el proyecto. Esto a veces no se tiene en cuenta y por algún motivo externo se puede perder todo el avance de un proyecto. Hay veces que por alguna situación ajena a nosotros el equipo en donde se está trabajando el proyecto se descompone. Ya sea por una falla eléctrica, deterioro de alguno de los componentes o por un descuido (derramar el café por ejemplo) el equipo puede quedar dañado y la información contenida perdida para siempre. Una manera de hacer respaldos es simplemente guardando el proyecto en CDs o memorias externas, de modo que si hay algún problema, podamos recuperar aunque sea una versión antigua del proyecto. Para asegurar que en caso de falla la pérdida sea mínima, es importante hacer respaldos constantes (diarios, semanales o mensuales).

Otra manera un poco más profesional de hacer respaldos, es a través del uso de repositorios. Un repositorio es una estructura de datos almacenada en un servidor que contiene un conjunto de archivos y directorios, y una bitácora de cambios en el repositorio. Generalmente en las empresas, donde el código es cerrado y es un bien que debe resguardarse, los repositorios se crean en servidores internos que no pueden ser accedidos desde fuera. Sin embargo, para proyectos de software libre, código abierto y hasta proyectos escolares, hay muchos sitios que ofrecen el servicio de repositorio desde Internet. De esta manera, lo único que se necesita para subir una versión del proyecto al repositorio es una conexión a Internet.

El repositorio para este proyecto se creó usando el servicio de Bitbucket. Bitbucket es un servicio de alojamiento de código fuente para proyectos que usan Mercurial o Git como sus sistemas de control de versiones. En sus cuentas gratuitas, se permite la creación de los repositorios que se deseen y permite elegir si el repositorio se quiere hacer privado o público, de modo que otras personas puedan descargar el proyecto y hacer modificaciones, pero sin la posibilidad de subir esas modificaciones. Para que otras

personas puedan subir cambios al repositorio se necesita tener una cuenta de pago, con lo que el repositorio podría ser accedido por un número determinado de usuarios.

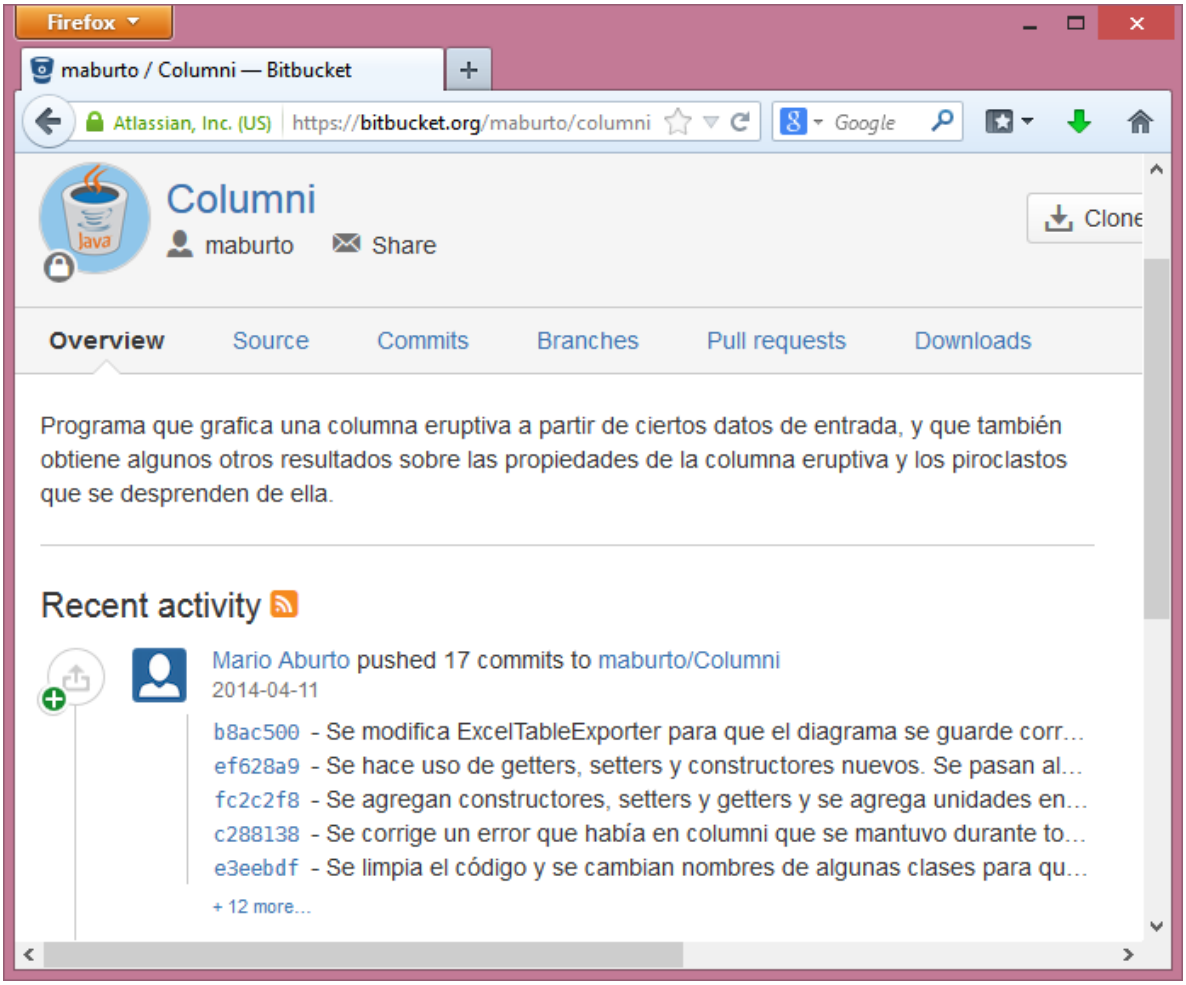


Ilustración 28: Repositorio del proyecto Columni

En la pantalla anterior (ilustración 28) se muestra el repositorio que se creó para el proyecto. Como se puede observar, aparece que el usuario *Mario Aburto* realizó un serie de *commits*. Estos *commits* son los que se hacen desde mercurial. Para subir el proyecto al repositorio se tiene que hacer un *push* desde Netbeans. Para esto se da clic derecho en el nombre del proyecto y luego en *Mercurial->Remote->Push.* El Plugin de Mercurial debe estar instalado dentro de Netbeans para que aparezca esta opción. Netbeans mostrará la siguiente ventana (ilustración 29):

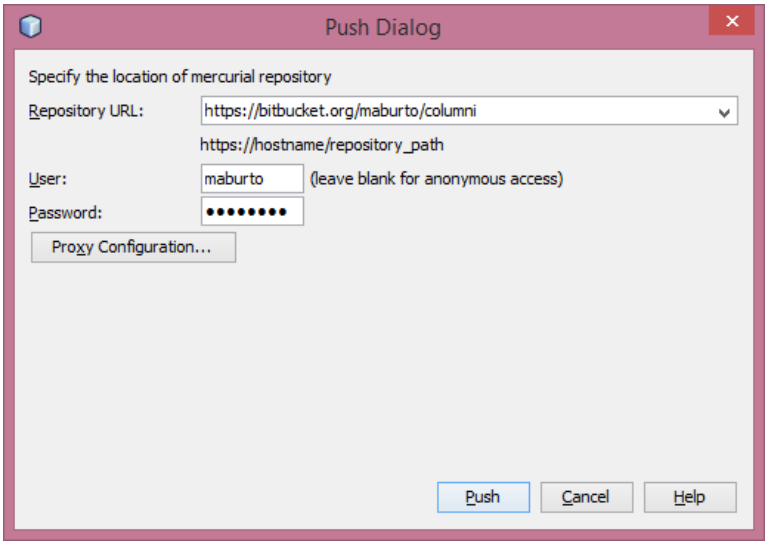


Ilustración 29: Realización de un push

En dicha ventana se debe escribir la dirección del repositorio del proyecto, así como el usuario y contraseña de un usuario del repositorio. Al presionar el botón *Push* el proyecto se subirá al repositorio. De esta manera tendremos respaldado el proyecto.

Para descargar el proyecto del repositorio simplemente se hace un *pull* desde el menú *Team->Remote->Pull*. Se pedirán los mismos datos que para el *push* (ilustración 30).

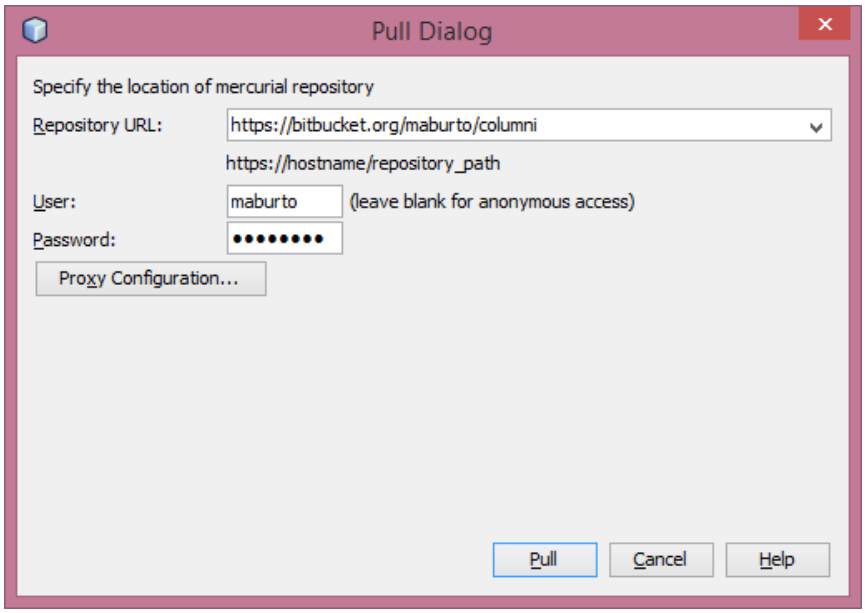


Ilustración 30: Realización de un pull

IV. Desarrollo de aplicación

En este capítulo se describirá cada etapa del modelo de cascada, que fue el que se eligió para el desarrollo de la aplicación.

A. Análisis y definición de requerimientos

Para la etapa de análisis primero se hizo el análisis del programa en Fortran. Este análisis viene detallado en la sección expuesta anteriormente. Tras analizar el código de Fortran, se clasificaron las variables en variables de entrada y de salida. La tabla 4 muestra las variables de entrada.

Nombre	**Descripción**	**Unidades**
B0	Radio del cráter (radio en Z0)	[m]
U0	Velocidad inicial (velocidad en Z0)	[m/s]
N0	Fracción de masa del gas inicial (en Z0) [porcentaje de gas en la columna eruptiva]	[-] (adimensional)
TH0	Temperatura inicial (temperatura en Z0)	[K]
Z0	Altura del cráter	[m]
DH	Incremento para Altura	[m]
DT	Incremento para Tiempo	[s]
NPINV	Número de alturas en las que se conoce la velocidad del viento	[-] (adimensional
ZW1,VW1	Altura1, Velocidad1	[m],[m/s]
NCLAST	Número de piroclastos de los que se tienen datos	[-] (adimensional)
CLAST,DENS	Diámetro1, Densidad1	[m],[kg/m3]

Tabla 4: Variables de entrada

La tabla 5 muestra las variables de salida.

Nombre	**Descripción**	**Unidades**
MassFluxZ0	Gasto másico en Z0: cantidad de material que atraviesa esa sección por segundo.	[kg/s]
HB	Es la altura en la que termina la sección **convective region** y comienza la sección **umbrella cloud.**	[m]
RadHB	Radio de la columna en la altura HB.	[m]
VelHB	Velocidad a la que viaja el material en HB.	[m/s]
DensHB	Densidad de la columna en la altura HB.	[kg/m3]
MassFluxHB	Gasto másico en HB: cantidad de material que atraviesa esa sección por segundo.	[kg/s]
Clast1	Diámetro y densidad del piroclasto 1.	[m],[kg/m3]
Downwind	Sirve para determinar zona en la que cayeron piroclastos de las mismas características. Ver ilustración 31.	[m]
Upwind	Sirve para determinar zona en la que cayeron piroclastos de las mismas características. Ver ilustración 31.	[m]
Sideclast	Sirve para determinar zona en la que cayeron piroclastos de las mismas características. Ver ilustración 31.	[m]

Tabla 5: Variables de salida

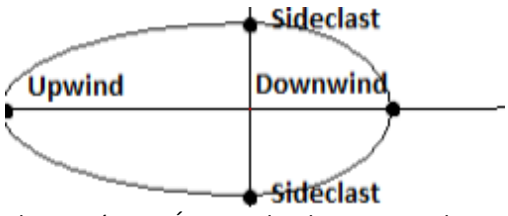


Ilustración 31: Área en donde caen piroclastos con las mismas características

Cabe aclarar que al inicio del desarrollo del sistema, algunas de las variables tenían otros nombres. Los nombres que se muestran en las tablas anteriores son los nombres finales que se pusieron a estas variables.

Después de analizar el código, se diseñó un primer prototipo usando una hoja de cálculo para representar de manera muy general la interfaz gráfica que se iba a generar (ver ilustración 32 en la sección). Este prototipo se mostró al Dr. Espíndola para que opinara sobre él y se modificó en base a sus observaciones. En esta fase se descartaron muchas variables consideradas en un principio como variables de salida, pero que el Dr. Espíndola indicó que eran sólo variables intermedias que no servían. También se observó que era importante agregar funcionalidades de guardado y cargado de datos, así como agregar una manera fácil de tomar los datos del programa para usarlos en otros sitios.

Los requerimientos específicos fueron los siguientes:

- Interfaz gráfica desde la cual se puedan introducir los datos y observar los resultados de los cálculos
- Diagrama a escala en el que se pueda observar la columna eruptiva usando las coordenadas calculadas por el programa.
- Diagramas individuales para los piroclastos de los que se tiene información. El programa mostrará un diagrama a escala del área alrededor del volcán donde se pueden encontrar piroclastos con las mismas características
- Posibilidad de guardar datos de entrada en un archivo para poder cargarlos en otro momento en que se ejecutara la aplicación.
- Funcionalidad para exportar el diagrama a un archivo de tipo PNG.
- Funcionalidad para exportar todos los datos de entrada y salida a un archivo de tipo XLS (hoja de cálculo). En este archivo se tiene que incluir los datos de entrada,

los datos de salida, las coordenadas de la columna eruptiva y el diagrama de la columna (imagen).

- Idioma inglés por defecto para todos los textos, pero agregando la posibilidad de cambiarlo a español desde la misma aplicación.
- Manual de usuario para entender lo que hace el sistema.

B. Diseño del sistema

El prototipo del sistema se hizo en un principio en una hoja de cálculo para indicar a grandes rasgos las secciones que el sistema debía tener. La ilustración 32 muestra el esquema que se hizo.

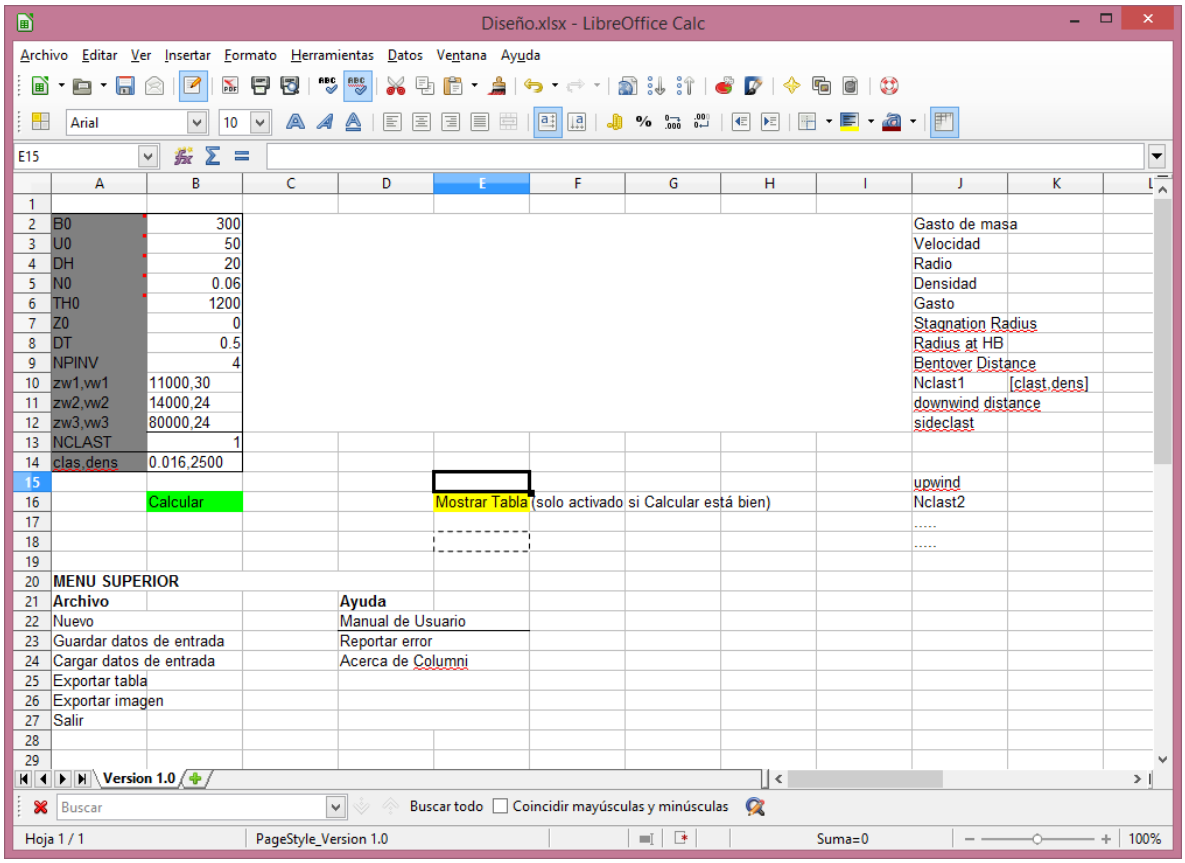


Ilustración 32: Prototipo 1 de la aplicación Columni

Existen herramientas más especializadas para hacer este tipo de prototipos. Sin embargo, para el primer prototipo con esta hoja de cálculo fue suficiente, pues usar otras herramientas hubiera implicado más tiempo para tener el prototipo. Este enfoque sencillo permitió agilizar las cosas, sin tener un impacto en la calidad del sistema. Este prototipo ayudó a identificar las variables de entrada y de salida principalmente.

El siguiente prototipo no tuvo un diseño muy bueno (ver ilustración 33), pero ayudó precisamente a entender mejor los requerimientos de la interfaz . Fue después de que se

hizo este prototipo que se hizo evidente la necesidad de implementar tablas para que la aplicación tuviera una apariencia más estética, más ordenada y más profesional.

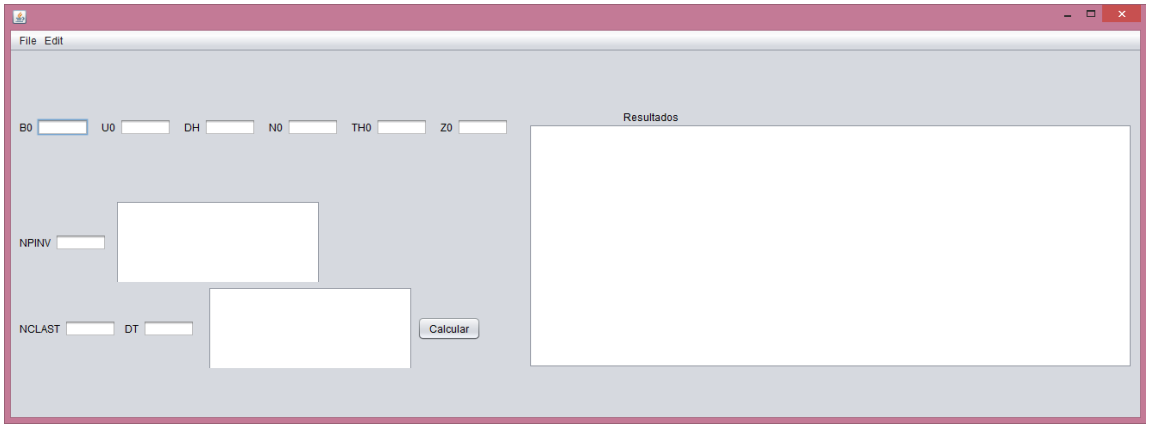


Ilustración 33: Prototipo 2 de la aplicación Columni

En el siguiente prototipo (ilustración 34) se hizo uso de tablas para los datos, con lo que se ve más profesional el diseño. Aún así, en este prototipo aún no se tienen todos los elementos que quedarían en la versión final del proyecto. Esto se debe a que conforme se fue avanzando se fueron identificando más elementos que debían agregarse al proyecto. Estos elementos serán descritos en las secciones siguientes.

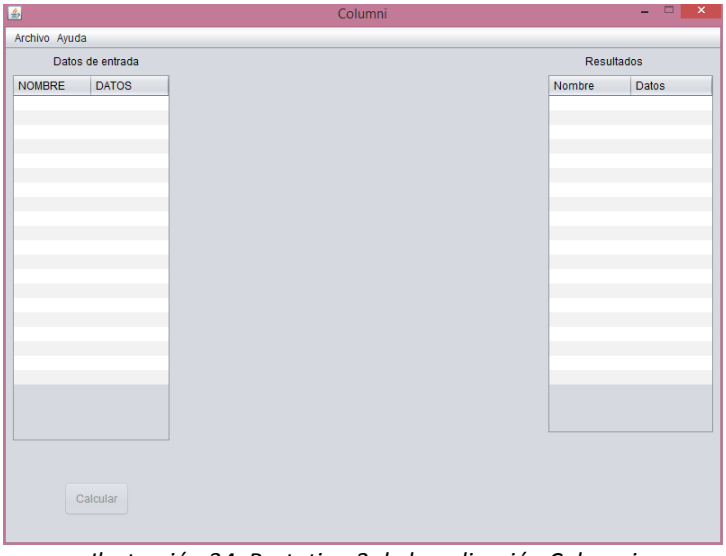


Ilustración 34: Prototipo 3 de la aplicación Columni

Una vez definido a grandes rasgos cómo se tenía que ver la aplicación, tenemos que determinar los elementos que conformarán el sistema. En este caso, tendremos una clase que se encargará de hacer todos los cálculos que hacía el programa de Fortran y la llamaremos *Columni.java*. El resto serán básicamente elementos de la interfaz gráfica. Como aún no está claro qué clases tendrá el proyecto, se analizará primero los paquetes que se utilizarán:

columni: contiene la clase principal del programa, desde la que se ejecuta el resto del código.

columni.model: contiene las clases más abstractas del programa, que forman el modelo de la aplicación.

columni.utils: son clases que sirven de soporte al programa, pero que no forman parte del modelo, ni la vista, ni el controlador. Tal es el caso de las validaciones que se hacen para los datos de la tabla de entradas y la clase que realiza la exportación del archivo de hoja de cálculo.

columni.view: en este paquete se encuentran las clases que forman la vista y el controlador. No se hizo un paquete especial para el controlador, pues en esta aplicación vista y controlador están muy unidos, por lo que sería poco práctico separarlos.

resources: en este paquete se encuentran los archivos necesarios para aplicar diferentes lenguajes a la aplicación, la biblioteca de *JExcelApi* que se usa para la exportación a una hoja de cálculo, los logos que se usan dentro de la aplicación, así como el manual de usuario.

En la siguiente sección se empezarán a identificar las clases que formarán parte de los paquetes listados anteriormente.

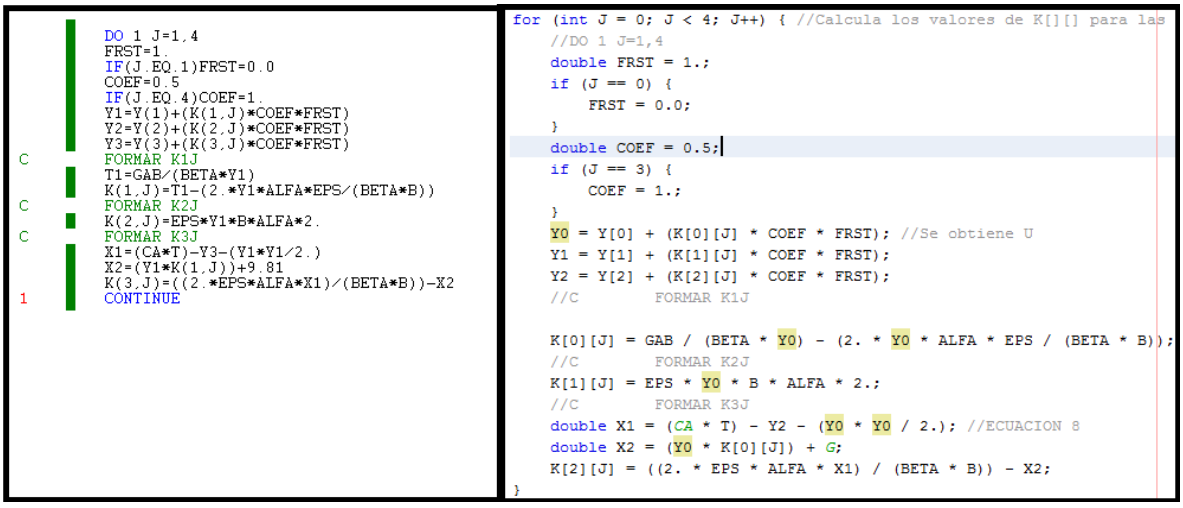
C. Implementación y pruebas unitarias

En un principio, se dio prioridad al elemento que realiza todos los cálculos. Es por ello que en esta etapa se realizó el desarrollo de la clase *Columni.java* que fue traducida línea por línea desde el código de Fortran y es la que realiza todos los cálculos. Para crear esta clase a partir del programa en Fortran fue necesario hacer una serie de adaptaciones por las diferencias entre estos lenguajes. La ilustración 35 muestra algunas de estas adaptaciones. Las consideraciones más relevantes para estas adaptaciones fueron las siguientes:

- En Fortran los arreglos van de 1 a n, y no de 0 a n-1 como en Java, donde n es el tamaño del arreglo. Por lo tanto, se tuvieron que hacer ajustes en el código para tomar en cuenta este detalle.

- Varios de los ciclos en el código de Fortran se hacían empleando la sentencia *goto*, que no existe en Java, por lo que se adaptaron para ser ciclos *while*.

- Se usaba la sentencia *goto* para salir de los ciclos bajo determinadas situaciones. En Java se usaron banderas, así como las sentencias *break* y *continue* para lograr el mismo efecto



```
      DO 1 J=1,4
      FRST=1.
      IF(J.EQ.1)FRST=0.0
      COEF=0.5
      IF(J.EQ.4)COEF=1.
      Y1=Y(1)+(K(1,J)*COEF*FRST)
      Y2=Y(2)+(K(2,J)*COEF*FRST)
      Y3=Y(3)+(K(3,J)*COEF*FRST)
C     FORMAR K1J
      T1=GAB/(BETA*Y1)
      K(1,J)=T1-(2.*Y1*ALFA*EPS/(BETA*B))
C     FORMAR K2J
      K(2,J)=EPS*Y1*B*ALFA*2.
C     FORMAR K3J
      X1=(CA*T)-Y3-(Y1*Y1/2.)
      X2=(Y1*K(1,J))+9.81
      K(3,J)=((2.*EPS*ALFA*X1)/(BETA*B))-X2
1     CONTINUE
```

```
for (int J = 0; J < 4; J++) { //Calcula los valores de K[][] para las
    //DO 1 J=1,4
    double FRST = 1.;
    if (J == 0) {
        FRST = 0.0;
    }
    double COEF = 0.5;
    if (J == 3) {
        COEF = 1.;
    }
    Y0 = Y[0] + (K[0][J] * COEF * FRST); //Se obtiene U
    Y1 = Y[1] + (K[1][J] * COEF * FRST);
    Y2 = Y[2] + (K[2][J] * COEF * FRST);
    //C         FORMAR K1J

    K[0][J] = GAB / (BETA * Y0) - (2. * Y0 * ALFA * EPS / (BETA * B));
    //C         FORMAR K2J
    K[1][J] = EPS * Y0 * B * ALFA * 2.;
    //C         FORMAR K3J
    double X1 = (CA * T) - Y2 - (Y0 * Y0 / 2.); //ECUACION 8
    double X2 = (Y0 * K[0][J]) + G;
    K[2][J] = ((2. * EPS * ALFA * X1) / (BETA * B)) - X2;
}
```

Ilustración 35: Diferencias entre programa en Fortran (izquierda) y programa en Java (derecha)

Como se mencionó en la introducción, el objetivo de esta tesis no es directamente la realización del programa que implemente el modelo de la columna eruptiva. Por ello, no se hizo desde cero, si no que se tomó como base el programa de Fortran. Es importante mencionar que para este punto los expertos son los científicos, por lo que incluso si el desarrollo se hiciera desde cero, tendría que hacerse con asesoría continua de un científico. Aún así se vuelve importante que como ingenieros tengamos buenas bases de ciencias básicas, pues de otra manera sería muy complicado implementar este tipo de proyecto.

Lo que se hizo para comprobar que el programa Columni en Java funcionara correctamente fue hacer que los resultados se imprimieran en pantalla y se guardaran en archivos, como lo hace el programa original en Fortran. De esta manera se pudo comprobar que los resultados eran los mismos. Para esto se usó el mismo archivo *INPUT.txt* para las variables de entrada:

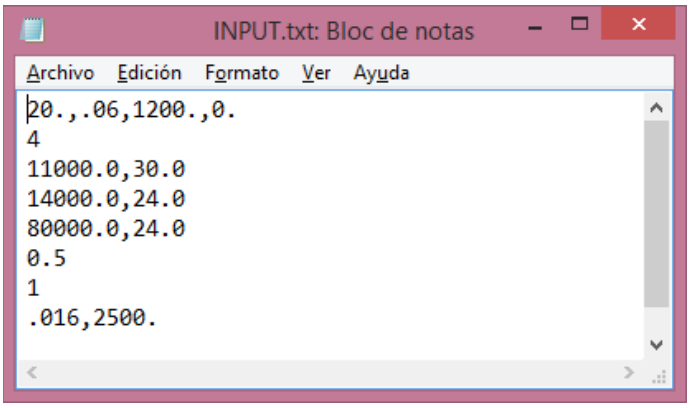


Ilustración 36: Archivo con los valores de las variables de entrada

En la ilustración 37 se muestra el archivo *output1.txt* generado tras la ejecución de la clase *Columni.java*

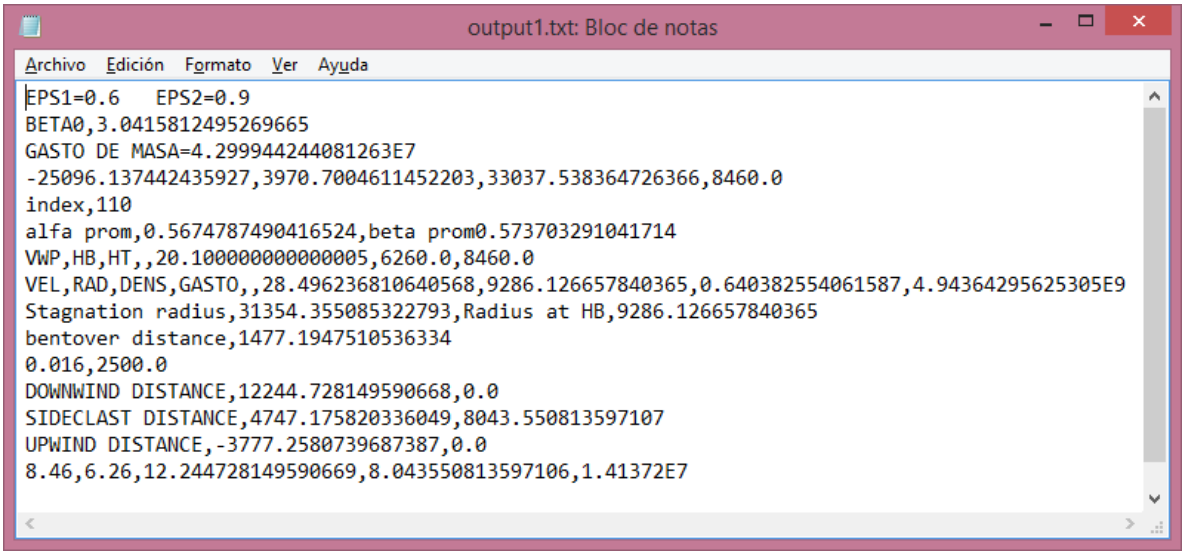


Ilustración 37: Archivo con los resultados del programa en Java

En la ilustración 38 se muestran los resultados de la ejecución del programa en Fortran, mismos que se mostraron en la sección .

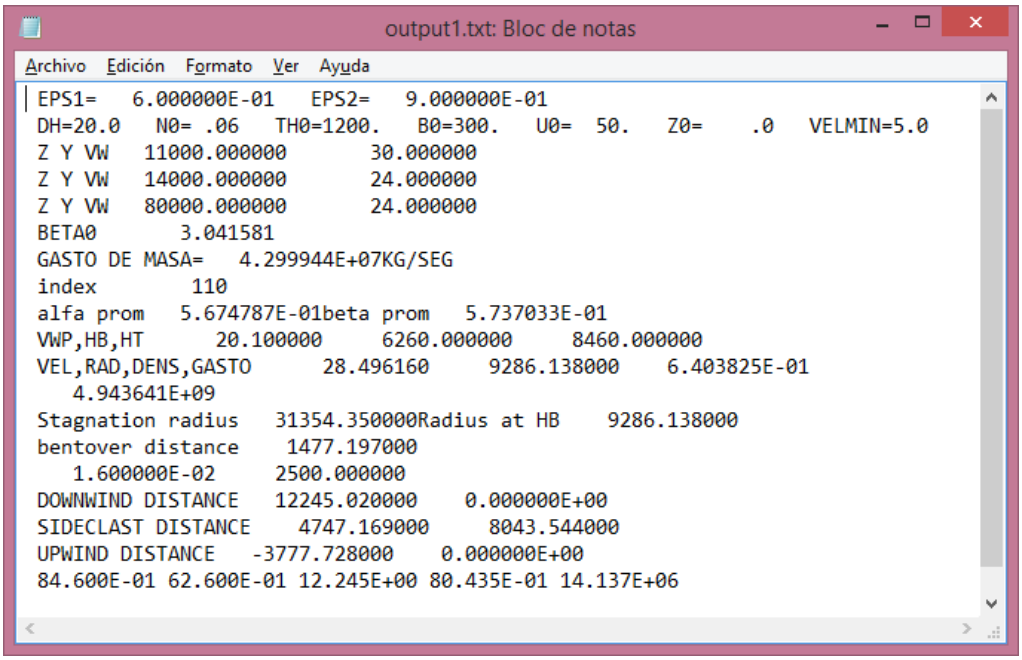


Ilustración 38: Archivo con los resultados del programa en Fortran

La tabla 6 muestra los resultados tanto para el programa en Fortran como para el programa en Java usando los mismos datos de entrada.

Variable de salida	**Resultado en Fortran**	**Resultado en Java**	**Porcentaje de error** $e = \frac{\lvert V_{real} - V_{estimado} \rvert}{V_{real}} x100$
MassFluxZ0(GASTO DE MASA)	4.299944E+07	4.299944244081263e7	0.000006%
HB	6260.0	6260.0	0%
RadHB (RAD)	9286.138	9286.126657840365	0.000122%
VelHB (VEL)	28.496160	28.496236810640568	0.000270%
DensHB (DENS)	6.403825E-01	0.640382554061587	0.000008%
MassFluxHB (GASTO)	4.943641E+09	4.943642956253O5E9	0.000040%
Clast1	0.016,2500	0.016,2500	----------
Downwind	12245.02	12244.728149590668	0.002383%
Sideclast	4747.169	4747.175820336049	0.000144%
Upwind	-3777.728	-3777.2580739687387	0.012439%

Tabla 6: Comparación de resultados para programa de Fortran y programa de Java

Se calculó el porcentaje de error para todas las variables, considerando que el valor real era el del programa en Fortran y el valor aproximado era el del programa en Java. El porcentaje de error es mínimo, pero no es 0. Se determinó que esta diferencia en los resultados se debe a que Fortran y Java usan diferente número de bytes para almacenar los valores de tipo real de doble precisión (*double*). Es por esto que al ir realizando los cálculos, el resultado se desvía un poco.

Una vez que se comprobó que la clase *Columni.java* se comportaba correctamente, se inició la creación de la interfaz gráfica directamente desde Netbeans. Poco a poco se

fueron agregando las clases necesarias para cumplir con los requerimientos, quedando así la organización de las clases en paquetes (ilustración 39):

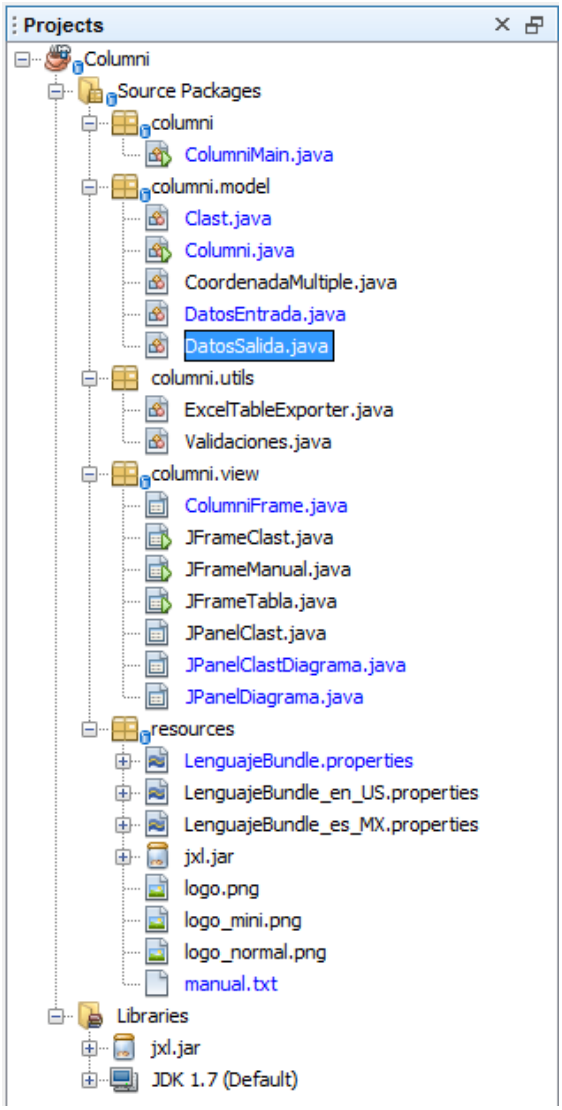


Ilustración 39: Organización en paquetes

Se generó un diagrama de clases para el proyecto, pero debido a su tamaño, no es posible ponerlo completo aquí. Es por eso que se muestran sólo miniaturas de los diagramas de clase para cada paquete del proyecto, así como la lista de clases que compone a cada paquete. De esa manera se explicará individualmente cada componente del sistema y qué función tiene.

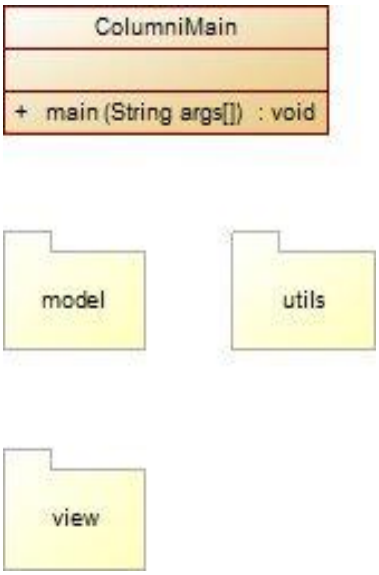


Ilustración 40: columni.main

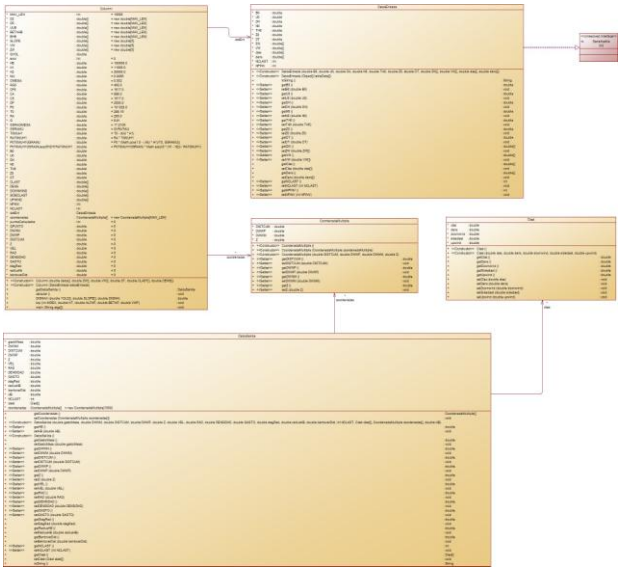


Ilustración 41: columni.model

columni.main (**ilustración** 40)

- *ColumniMain.java*: es la clase principal del proyecto (contiene el método main). Desde aquí se manda crea la ventana principal, definida por la clase *ColumniFrame.java*.

columni.model (ilustración 41)

- *Clast.java*: define a un piroclasto por sus atributos clas, dens, downwind, sideclast y upwind.

- *Columni.java*: es el programa que hace todos los cálculos para la columna eruptiva. Está basado en el programa de Fortran.

- *CoordenadaMultiple.java*: define las tres componentes en *x* para una componente en *y* para la gráfica de la columna eruptiva. Sus atributos son DISTCUM, DWWP, DWWM y Z.

- *DatosEntrada.java*: es un objeto que define los datos de entrada de la aplicación. El objeto de la clase *Columni.java* debe recibir un objeto de tipo *DatosEntrada.java* en su constructor para después poder hacer los cálculos. Sus atributos son

básicamente las variables de entrada que fueron definidas anteriormente en la tabla de variables de entrada.

- *DatosSalida.java*: es un objeto que define los datos de salida de la aplicación. Tras realizar los cálculos, el objeto de la clase *Columni.java* crea un objeto de la clase *DatosSalida.java* para devolverlo a *ColumniFrame.java*, que será la clase que desplegará los resultados. Sus atributos son básicamente las variables de salida que fueron definidas anteriormente en la tabla de variables de salida.

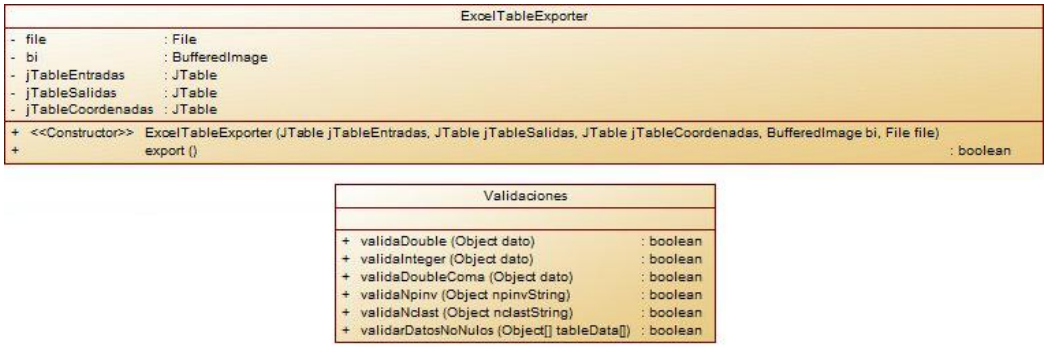


Ilustración 42: columni.utils

columni.utils (ilustración 42)

- *ExcelTableExporter.java*: clase auxiliar para realizar la exportación al archivo con extensión *.xls* (hoja de cálculo). Esta clase usa la biblioteca *JExcelApi* (ilustración 43), que se tuvo que agregar al proyecto con el JAR *jxl.jar* (ilustración 44).

```
package columni.utils;

import java.awt.image.BufferedImage;
import java.io.*;
import javax.imageio.ImageIO;
import javax.swing.*;
import jxl.write.*;
import jxl.*;
import jxl.write.Number;

/**
 *
 * @author Mario
 */
public class ExcelTableExporter {
```

Ilustración 43: Importación JExcelApi

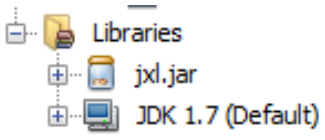


Ilustración 44: Biblioteca JExcelApi

- *Validaciones.java*: clase auxiliar para realizar las validaciones de la tabla de entradas en la ventana principal (ilustración 45). Estas validaciones son para comprobar si un dato es un número entero, un número real (*double)* o si es nulo.

```
public class Validaciones {

    public static boolean validaDouble(Object dato) {
        try {
            Double.parseDouble((String) dato);
            return true;
        } catch (NumberFormatException e) {
            return false;
        }
    }

    public static boolean validaInteger(Object dato) {
```

Ilustración 45: Sección de código de la clase Validaciones.java

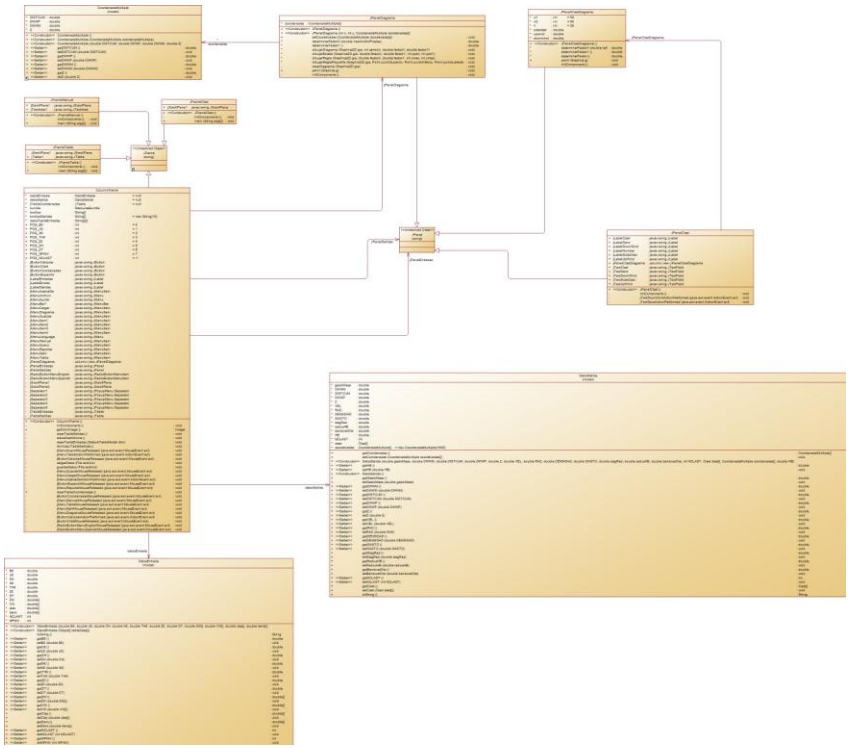


Ilustración 46: columni.view

columni.view (ilustración 46)

- *ColumniFrame.java*: ventana principal de la aplicación, aquí es donde se muestran todos los datos de entrada y de salida, y el diagrama de la columna eruptiva.

- *JFrameClast.java*: ventana que muestra los resultados y el diagrama para los piroclastos de los que se tiene información. Sólo se muestra una vez que se han hecho los cálculos y al dar clic en el botón *Clast*.

- *JFrameManual.java*: ventana que muestra el manual de usuario. Este manual se lee del archivo *manual.txt* que se encuentra en el archivo JAR del programa (*Columni.jar*)

- *JFrameTabla.java:* ventana para mostrar las coordenadas de la columna eruptiva en una tabla. Los campos de la tabla son DWWM,DISTCUM,DWWP y Z.

- *JPanelClast.java*: panel individual para cada piroclasto. En el objeto de tipo *JFrameClast* se agregan tantos objetos *JPanelClast* como piroclastos se hayan introducido en el programa. Estos objetos *JPanelClast* se encuentran en un *JScrollPane*, de modo que sea posible desplazarse sobre la ventana en caso que no quepan todos en el tamaño de la ventana.

- *JPanelClastDiagrama.java*: panel al que se le sobreescribe el método *paint()* para poder dibujar el diagrama del área donde es posible que caigan otros piroplastos con las mismas características.

- *JPanelDiagrama.java*: panel al que se le sobreescribe el método *paint()* para poder dibujar el diagrama de la columna eruptiva.

En la siguiente sección, , se muestra como todos los elementos descritos anteriormente trabajan en conjunto. Los elementos de la interfaz gráfica están muy relacionados entre sí. Es por eso que es en la fase de integración donde se verán trabajando en conjunto. Por lo tanto seguiremos hablando de estos elementos ahí, pero ya vistos como un todo.

D. Integración y pruebas del sistema

El archivo ejecutable que se genera es un archivo JAR (Java Archive) que son archivos que tiene formato ZIP pero con la extensión *.jar*. Dentro de este archivo JAR se encuentran las clases del proyecto y el paquete *resources*, necesarios para correr el programa de Java. Al ejecutar el archivo JAR aparece un SplashScreen que se muestra durante 0.75 segundos con el logo de la aplicación, tras lo cual se abre la ventana principal del programa. Esto se logró poniendo en el archivo MANIFEST.MF la línea *SplashScreen-Image: resources/logo_normal.png* y llamando a la función *Thread.sleep(750)* en el método *main* de la clase *ColumniMain.java.* Al ejecutar la aplicación este logo se observa también en el ícono de la barra de tareas y en la esquina superior izquierda de la ventana principal. La idea era darle mayor presencia a la aplicación *Columni*. La imagen se hizo a partir de una de las gráficas generadas por el programa y se le dieron los colores de la UNAM y los colores de la Facultad de Ingeniería (ilustración 47).

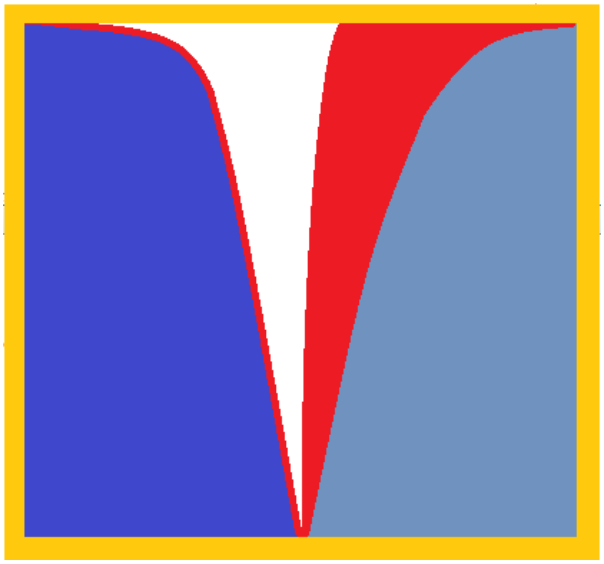


Ilustración 47: Splash Screen

La parte más importante de la interfaz gráfica es la ventana principal del programa (ilustración 48), la cual es una clase que hereda de *JFrame* y que fue personalizada para contener los siguientes elementos:

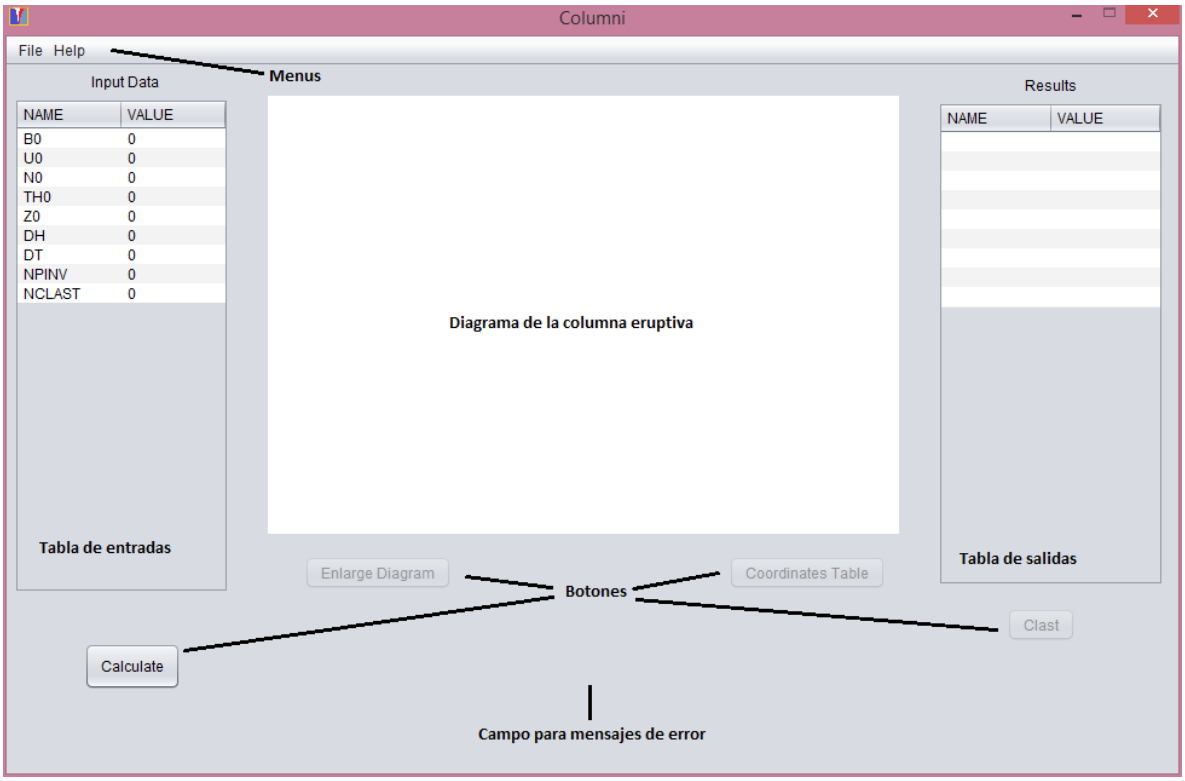


Ilustración 48: Elementos de la interfaz gráfica

En la siguiente imagen (ilustración 49) se muestra la misma ventana pero ya después de haber introducido valores de prueba y presionado el botón *Calculate*. Como se puede observar, en esta ventana ya aparecen los resultados de los cálculos, así como la gráfica de la columna eruptiva. Cabe mencionar que la línea interna representa el centro de la columna eruptiva y las líneas exteriores representan los bordes de la columna. También es importante resaltar que cuando se presiona el botón *Calculate*, se habilitan otros botones dentro de la interfaz de usuario como se observa en la imagen.

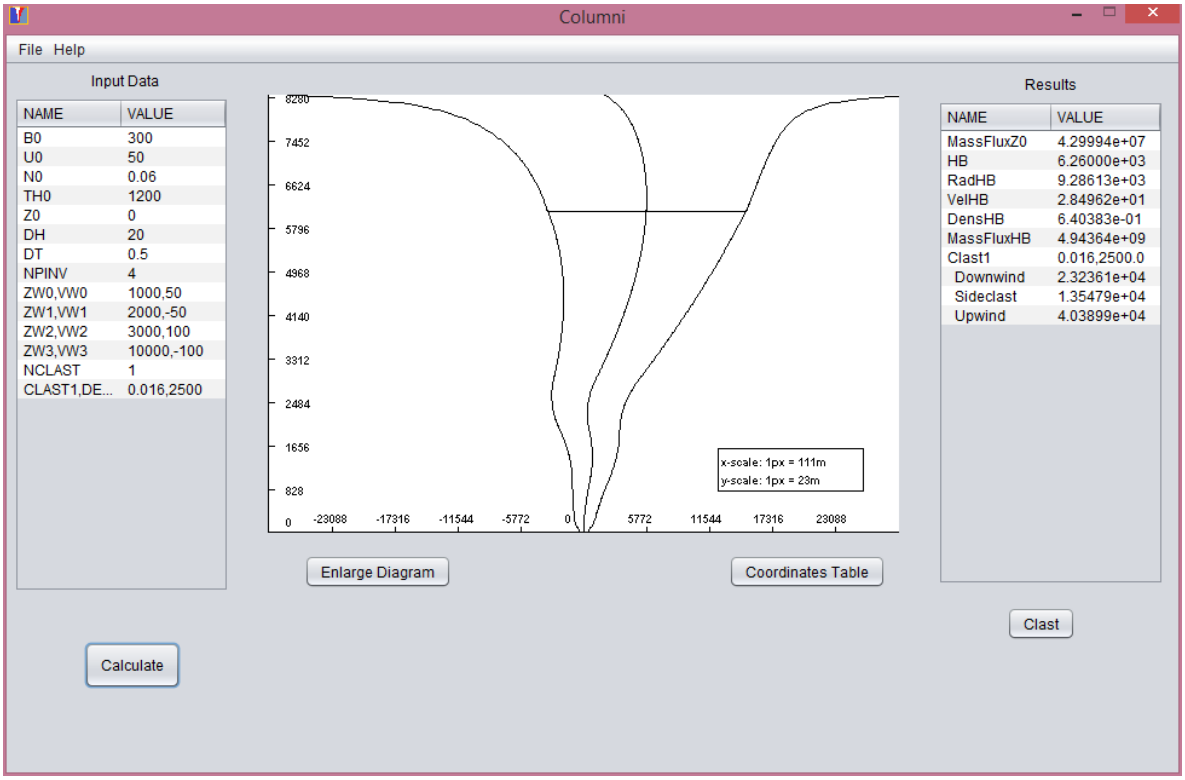


Ilustración 49: Aplicación Columni después de introducir valores de datos de entrada y presionar Calcular

A continuación se muestra una descripción de los componentes de la interfaz gráfica:

Tabla de entradas: en esta tabla se introducen las variables de entrada que se requieren para el cálculo de los resultados y de las coordenadas del diagrama. Es importante mencionar que esta tabla valida que los valores introducidos sean correctos cuando se introducen en la tabla. Todos los valores son de tipo real (*double*) a excepción de las variables NPINV y NCLAST, que son valores de tipo entero. En caso de que el usuario introduzca un valor no válido, la tabla no se actualiza con el nuevo valor y se muestra un mensaje de error.

Tabla de salidas: Esta tabla se genera al dar clic en el botón Calcular. Contiene los resultados de los cálculos para la columna eruptiva.

Diagrama de la columna eruptiva: el diagrama se usa para graficar las coordenadas de la columna eruptiva.

Campo de mensajes de error: en este campo se muestran mensajes de error cuando se detecta que el usuario introdujo un valor no válido en la tabla de entradas.

Botones:

- **Calculate:** Una vez que se han introducido los valores en la tabla de entradas, al presionar este botón se realizan los cálculos usando estos valores. Tras hacer esto se generarán los valores de la Tabla de salidas, así como el diagrama de la columna eruptiva

- **Enlarge Diagram:** con este botón se puede ver el diagrama en una ventana individual (ver ilustración 50). Esta ventana se puede redimensionar para ver el diagrama con más detalle. La escala del diagrama se ajusta para las nuevas dimensiones de la ventana.

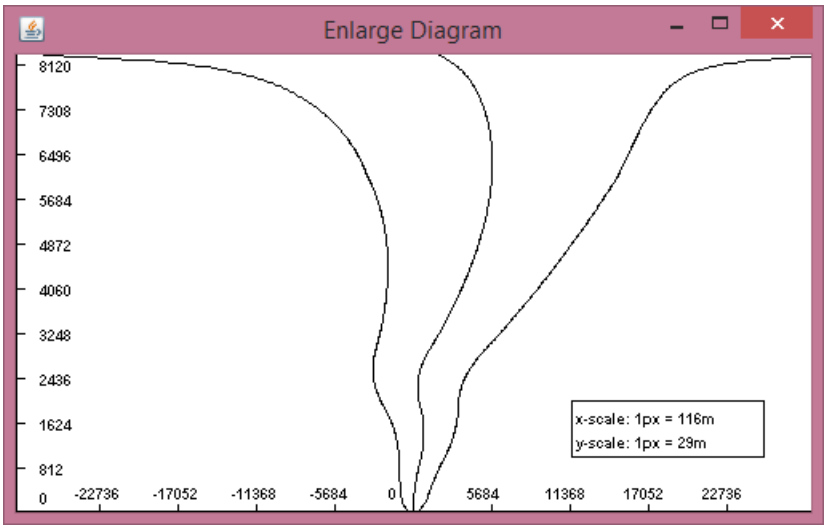
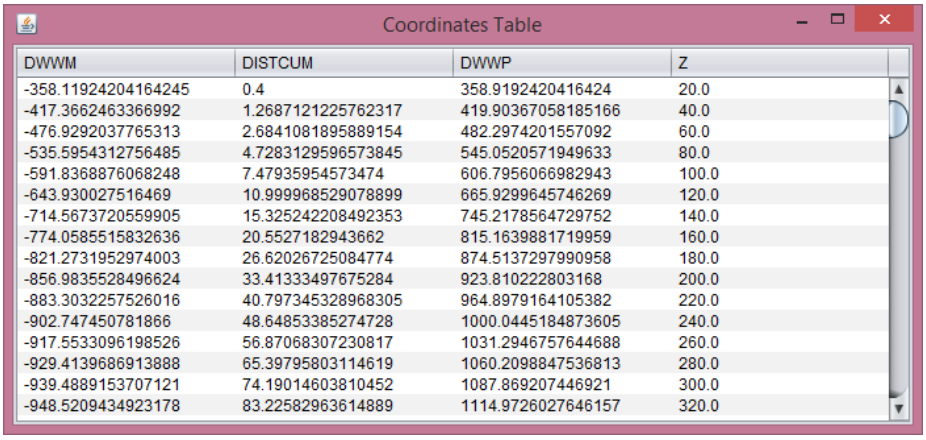


Ilustración 50: Ventana individual con diagrama de la columna eruptiva

- **Coordinate Table:** este botón abre una ventana en la que se muestra la tabla de coordenadas de la gráfica de la columna eruptiva (ilustración 51)



DWWM	DISTCUM	DWWP	Z
-358.11924204164245	0.4	358.9192420416424	20.0
-417.3662463366992	1.2687121225762317	419.90367058185166	40.0
-476.9292037765313	2.6841081895889154	482.2974201557092	60.0
-535.5954312756485	4.7283129596573845	545.0520571949633	80.0
-591.8368876068248	7.47935954573474	606.7956066982943	100.0
-643.930027516469	10.999968529078899	665.9299645746269	120.0
-714.5673720559905	15.325242208492353	745.2178564729752	140.0
-774.0585515832636	20.5527182943662	815.1639881719959	160.0
-821.2731952974003	26.62026725084774	874.5137297990958	180.0
-856.9835528496624	33.41333497675284	923.810222803168	200.0
-883.3032257526016	40.797345328968305	964.8979164105382	220.0
-902.747450781866	48.64853385274728	1000.0445184873605	240.0
-917.5533096198526	56.87068307230817	1031.2946757644688	260.0
-929.4139686913888	65.39795803114619	1060.2098847536813	280.0
-939.4889153707121	74.19014603810452	1087.869207446921	300.0
-948.5209434923178	83.22582963614889	1114.9726027646157	320.0

Ilustración 51: Tabla con las coordenadas de la columna eruptiva

- **Clast:** al presionar este botón se muestran los diagramas para los piroclastos de los que se hicieron cálculos (ilustración 52). Estos diagramas representan el área alrededor del volcán en la cual se pueden encontrar piroclastos con las mismas características.

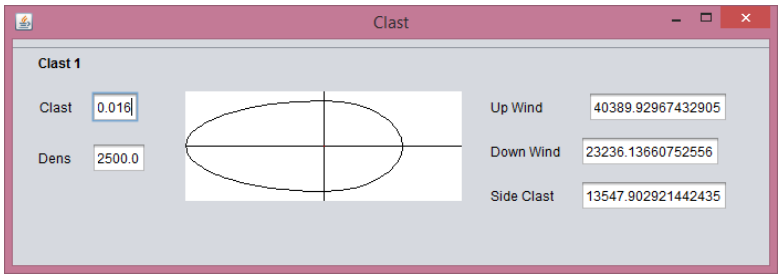


Ilustración 52: Diagrama de la zona donde caen piroclastos con las mismas características

Menús: (ilustraciones 53 y 54) en estos menús se muestran los siguientes elementos:

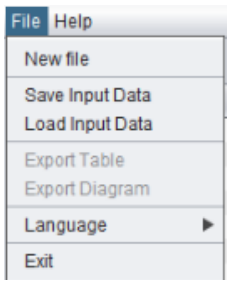


Ilustración 53: Menú File

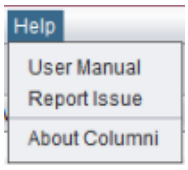


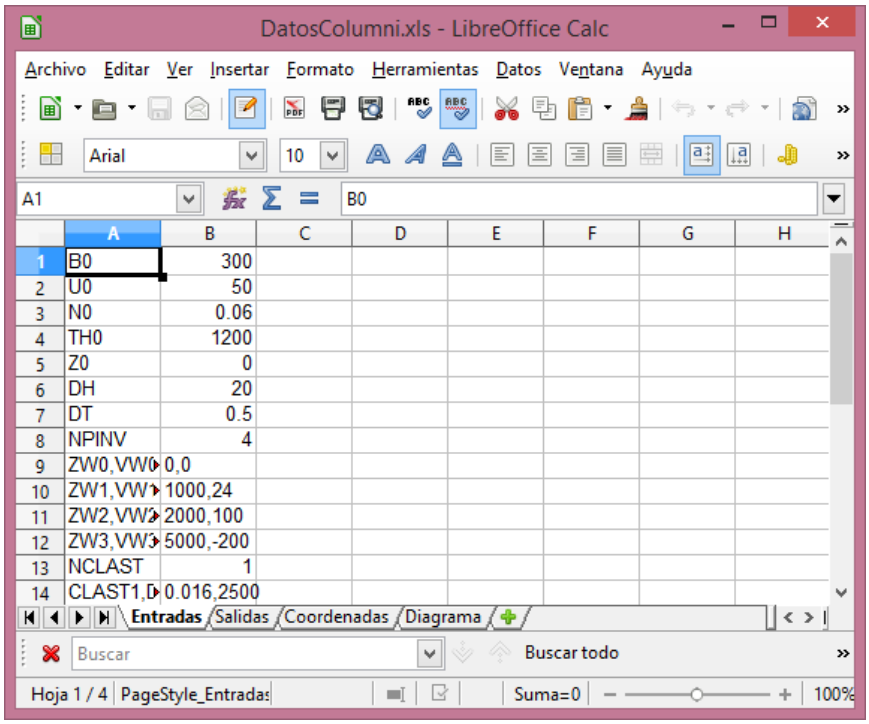
Ilustración 54: Menú Help

- **New file**: borra los datos de tabla de entradas, tabla de salidas y el diagrama, para que el usuario pueda introducir nuevos valores al programa.

- **Save Input Data**: permite guardar los datos que se encuentran en la tabla de entradas, para que puedan ser utilizados posteriormente en otra ejecución del programa

- **Load Input Data:** carga los datos de entrada desde un archivo con extensión *.dat*. Al cargar los datos desde el archivo, la tabla de entradas se llenará con los valores que habían sido guardados en una ejecución anterior del programa.

- **Export Table:** exporta todos los datos a un archivo de tipo hoja de cálculo con extensión *.xls* (ilustración 55). Se crea dentro del archivo cuatro secciones: datos de entrada, datos de salida, coordenadas y diagrama.



	A	B
1	B0	300
2	U0	50
3	N0	0.06
4	TH0	1200
5	Z0	0
6	DH	20
7	DT	0.5
8	NPINV	4
9	ZW0,VW0	0,0
10	ZW1,VW1	1000,24
11	ZW2,VW2	2000,100
12	ZW3,VW3	5000,-200
13	NCLAST	1
14	CLAST1,D	0.016,2500

Ilustración 55: Archivo exportado con extensión .xls

- **Export Diagram:** exporta el diagrama a una imagen con extensión *.png*.

- **Language**: permite elegir entre inglés y español. Cuando el idioma cambia, todos los elementos de la interfaz gráfica cambian el texto al idioma elegido. Para esto se usó la clase *java.util.ResourceBundle* (ilustración 57) y se crearon los archivos *LenguajeBundle.properties, LenguajeBundle_en_US.properties y LenguajeBundle_es_MX.properties* en el paquete *resources* (ilustración 56) .

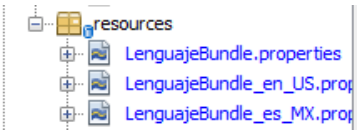


Ilustración 56: Archivos LenguajeBundle

```
Locale defaultLocale = new Locale("en", "US");
bundle = java.util.ResourceBundle.getBundle("resources/LenguajeBundle", defaultLocale);
JOptionPane.setDefaultLocale(Locale.ENGLISH);
```

Ilustración 57: Uso de ResourceBundle

- **Exit:** sirve para cerrar la aplicación.

- **User Manual:** muestra en una nueva ventana el manual de usuario. Este manual se encuentra dentro del ejecutable tipo JAR y es un archivo con extensión *.txt*. (ver ilustración 64 en)

- **Report Issue:** Aquí se muestra una dirección de correo a la que se pueden enviar reportes de problemas del programa Columni (ilustración 58).

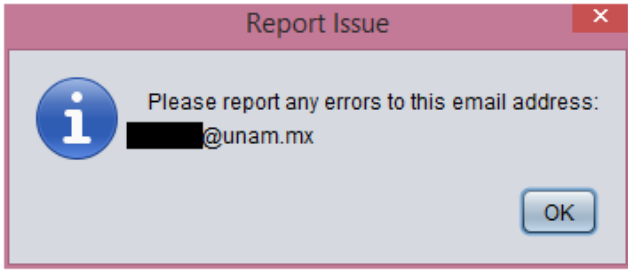


Ilustración 58: Ventana Report Issue

- **About Columni**: Muestra el nombre y versión del programa, así como los autores del mismo (ilustración 59).

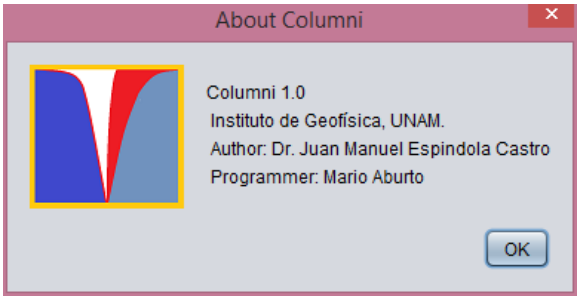


Ilustración 59: Ventana About Columni

Durante el desarrollo del proyecto, se fueron haciendo pruebas conforme se iban agregando elementos de la interfaz gráfica. Al final del desarrollo de la aplicación completa, también se hicieron pruebas. Las pruebas caen en las siguientes categorías:

- Comprobar que los botones y menús se habiliten y deshabiliten correctamente de acuerdo al estado del programa (ver ilustración 60).
- Comprobar que el programa valide correctamente cada uno de los valores de la tabla de datos de entradas
- Verificar la generación correcta de archivos: guardar y cargar datos de entrada, generación de tabla de hoja de cálculo y generación de imagen PNG con diagrama.

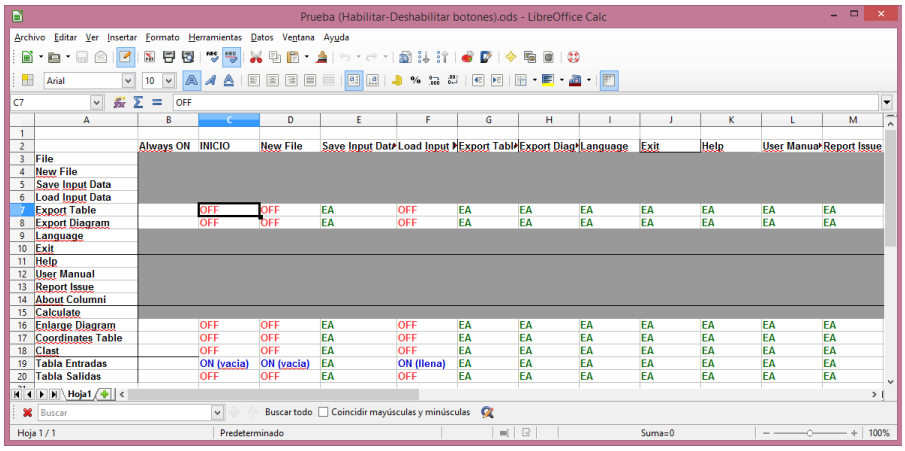


Ilustración 60: Prueba (habilitar-deshabilitar botones)

E. Operación y mantenimiento

En esta etapa se libera el programa al usuario final. Durante esta etapa se pueden encontrar errores importantes que deben ser corregidos. Para dar posibilidad a los usuarios de enviar estos errores se incluyó un botón en el menú de la aplicación que se llama *Report Issue* o *Reportar error* en el que se indica un correo al que pueden enviar los reportes de errores.

Esta etapa del ciclo de vida del software es la más larga de todas. Es por esto que las consideraciones hechas en el capítulo tuvieron en cuenta en gran medida a esta etapa. A continuación se muestran las consideraciones más destacadas para esta etapa del ciclo de vida del software y por qué le ayudan:

- Lenguaje de programación Java: al ser un lenguaje multiplataforma, no es necesario adaptar el código para cada tipo de sistema en el que se va a usar. La máquina virtual se encarga de que el código se ejecute correctamente independientemente del sistema a usar. Por ello, no habrá mayores complicaciones si durante esta etapa se requiere correr en algún otro sistema.

- Paradigma orientado a objetos: se uso este paradigma pues facilita el mantenimiento debido principalmente a sus características de encapsulamiento, herencia y polimorfismo.

- Netbeans como entorno de desarrollo: Netbeans es el IDE más usado para Java, por lo que al trabajar el proyecto en este entorno, será fácil de mantener para cualquier programador.

- Control de versiones: al llevar un control de versiones, se tiene un registro de los cambios que se van haciendo, por lo que durante la etapa de Operación y mantenimiento se llevará un registro de las correcciones que se vayan haciendo.

- Repositorio: la elección del repositorio también tiene que ver con la posibilidad de que este proyecto se vuelva un proyecto de código abierto. Este repositorio, junto al sistema de control de versiones elegido, permitirían que el código pudiera ser trabajado colaborativamente como en los proyectos de código abierto.

Este software podrá ser continuado en un futuro por cualquier otro programador sin muchas dificultades, pues se implemento usando buenas prácticas de programación. El programa viene documentado con la ayuda de Javadoc, lo que facilita aún más su mantenimiento.

Los métodos del programa fueron documentados usando el formato para Javadoc, de manera que se pudiera generar la documentación con esta herramienta. Javadoc genera un documento de tipo HTML en donde se describen las clases, sus atributos y métodos, así como a qué paquetes pertenecen. La documentación generada tiene el mismo formato que la documentación de la API de Java, por lo que es sencilla de entender si se ha consultado ya la API de Java.

```
18      /**
19       *
20       * @param B0 Vent Radius
21       * @param U0 Initial Velocity
22       * @param DH Distance Step
23       * @param N0 Water fraction
24       * @param TH0 Initial Temperature
25       * @param Z0 Vent Elevation
26       * @param DT Time Step
27       * @param ZW Height
28       * @param VW Velocity
29       * @param clas Diameter
30       * @param dens Density
31       */
32      public DatosEntrada(double B0, double U0, double DH, double N0,
33          this.B0 = B0;
```

Ilustración 61: Documentación del código con etiquetas de Javadoc

Para agregar documentación de Javadoc en el proyecto se deben poner comentarios que empiezan con "/**" y terminan con "*/". Dentro, se deben usar etiquetas que empiezan con el carácter @, para indicar que es un campo de Javadoc (ver ilustración 61). Ejemplos

de este tipo de etiquetas son @param, @return, @author. Generar el archivo HTML se puede lograr muy fácilmente desde Netbeans. Simplemente dar clic derecho en el nombre del proyecto y seleccionar la opción *Generate Javadoc* (ilustración 62).

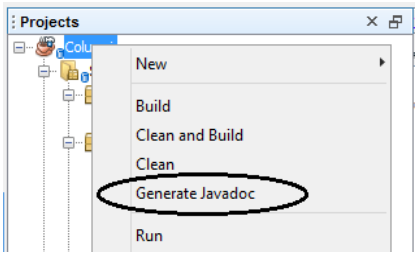


Ilustración 62: Generar Javadoc

Con esto Netbeans ejecutará el programa de Javadoc y se generará el archivo HTML que contendrá la documentación de las clases del programa Columni (ilustración 63):

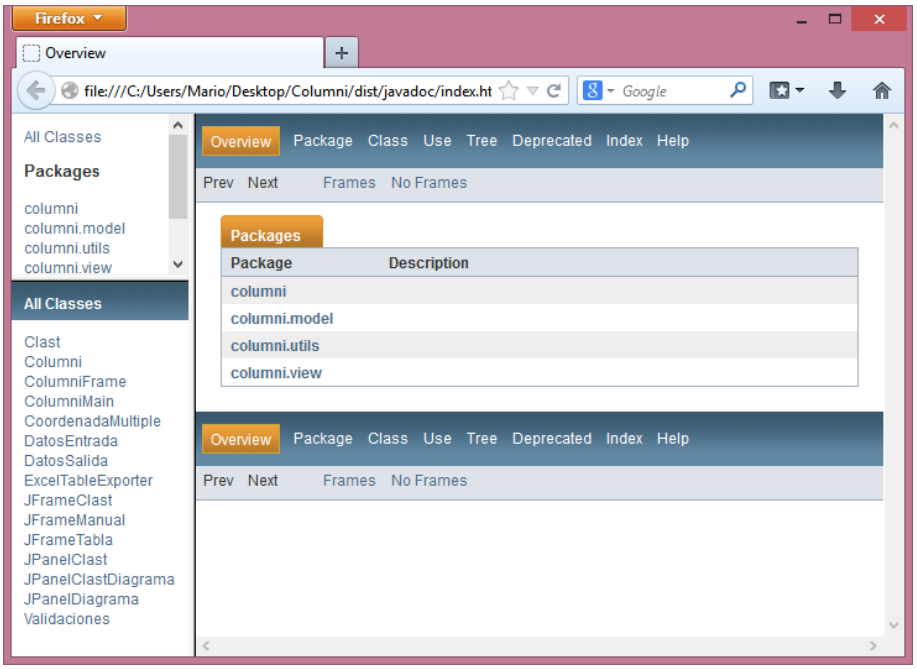


Ilustración 63: Archivo HTML generado por Javadoc

Este archivo puede ser consultado posteriormente por algún programador que quiera retomar el proyecto y lo ayudará a entender, sin necesidad de ver el código, para qué sirve cada elemento del programa.

V. Conclusiones

Los problemas en vulcanología son resueltos casi exclusivamente por científicos. Un ingeniero en computación parece no tener lugar en esta área de la ciencia. Pero como se demostró, la ingeniería en computación puede aportar mucho a la vulcanología. Aunque es cierto que es complicado entender varios conceptos físicos que vienen implícitos en los modelos matemáticos, las bases que tenemos como ingenieros en el área de las ciencias ayudan mucho.

Al analizar un programa en Fortran que describe las columnas eruptivas, se pudieron ver los problemas o las áreas de mejora que tenía. Entre las áreas de mejora detectadas, las más importantes son las siguientes:

- Mejorar la documentación dentro del código para facilitar a un nuevo programador modificar el programa
- Eliminar la necesidad de ser programador para poder obtener los resultados del programa
- Uso de una interfaz gráfica fácil de usar

Tomando en cuenta los objetivos planteados en esta tesis, los resultados más importantes son los siguientes:

- Se establecieron los elementos necesarios para desarrollar una aplicación para vulcanología con éxito, explicados en el capítulo . Para este punto se aplicaron las técnicas más novedosas que beneficiarían al proyecto, entre las que se pueden destacar el paradigma orientado a objetos, el modelo vista controlador, el sistema de control de versiones y el repositorio.

- Se mostró el desarrollo de una aplicación para vulcanología en el capítulo explicando cada paso que se siguió durante las etapas del modelo de desarrollo elegido (modelo de cascada).
- Se generó una aplicación completa en Java que analiza el comportamiento de una columna eruptiva.

VI. Apéndice A. Manual de usuario de la aplicación

Para acceder al manual de usuario se puso un botón en el menú superior llamado *User Manual* o *Manual de Usuario* dependiendo del lenguaje con el que se está corriendo. El manual se muestra en un *JFrame* independiente, con una barra de deslizamiento para ver su contenido.

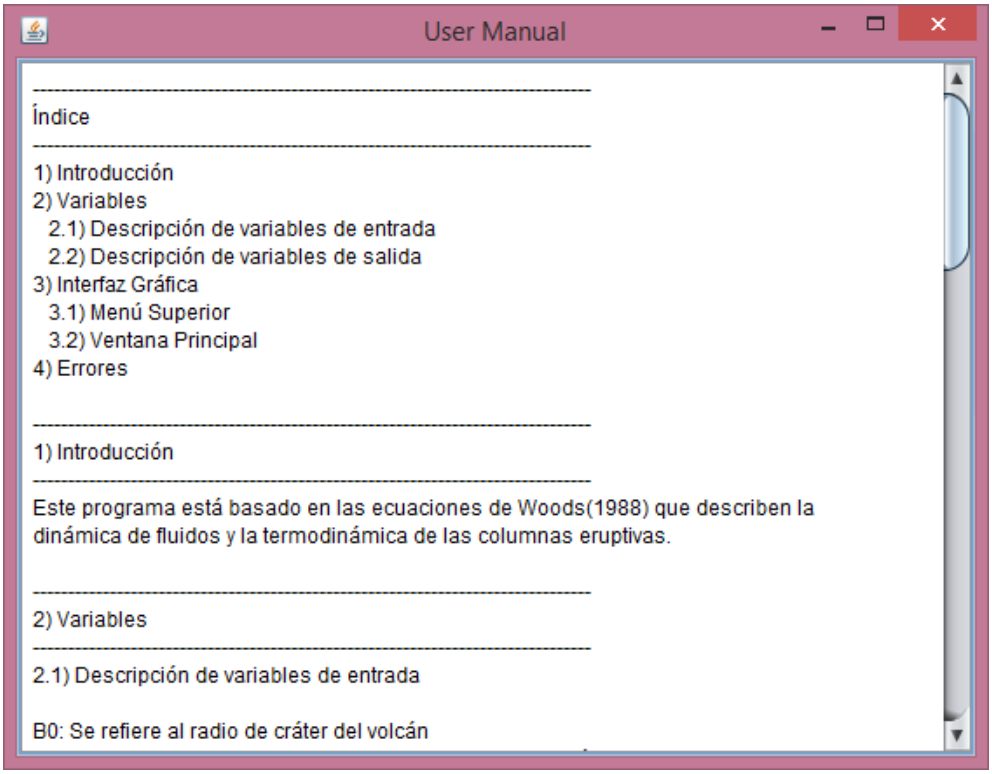


Ilustración 64: Ventana de Manual de Usuario

Este manual en realidad es un archivo *txt* que se encuentra dentro del archivo *.jar.* Desde Netbeans se puede editar, pues se encuentra en el paquete *resources* con el nombre *manual.txt.* En esta versión del programa el manual es muy sencillo y simplemente describe un poco las variables, menús y botones del programa. Se decidió que el manual de usuario estuviera dentro de la misma aplicación para evitar tener múltiples archivos en la máquina del usuario.

VII. Apéndice B. Definiciones

Basalto. es una roca ígnea volcánica de color oscuro, de composición máfica —rica en silicatos de magnesio y hierro y bajo contenido en sílice—, que constituye una de las rocas más abundantes en la corteza terrestre. http://es.wikipedia.org/wiki/Basalto

Estratigrafía. (Del lat. *strātus*, lecho, y *-grafía*). **1.** f. Estudio de los estratos arqueológicos, históricos, lingüísticos, sociales, etc. **2.** f. *Geol.* Parte de la geología que estudia la disposición y caracteres de las rocas sedimentarias estratificadas. **3.** f. *Geol.* Disposición seriada de las rocas sedimentarias de un terreno o formación. http://lema.rae.es/drae/?val=estratigrafia

Estratosfera. Es una de las capas más importantes de la atmósfera, situada entre la tropósfera y la mesósfera, y se extiende en una capa que va desde los 10 hasta los 50 km de altura aproximadamente. http://es.wikipedia.org/wiki/Estratosfera

Geodesia. (Del gr. γεωδαισία, división de la tierra). **1.** f. Ciencia matemática que tiene por objeto determinar la figura y magnitud del globo terrestre o de gran parte de él, y construir los mapas correspondientes. http://lema.rae.es/drae/?val=geodesia

Geofísico, ca. (De *geo-* y *físico*). **1.** adj. Perteneciente o relativo a la **geofísica. 2.** f. Parte de la geología que estudia la física terrestre. http://lema.rae.es/drae/?val=geofisica

Geología. (De *geo-* y *-logía*). **1.** f. Ciencia que trata de la forma exterior e interior del globo terrestre, de la naturaleza de las materias que lo componen y de su formación, de los cambios o alteraciones que estas han experimentado desde su origen, y de la colocación que tienen en su actual estado. http://lema.rae.es/drae/?val=geologia

Geoquímico, ca. (De *geo-* y *químico*). **1.** adj. Perteneciente o relativo a la **geoquímica. 2.** m. y f. Persona versada en **geoquímica. 3.** f. Estudio de la distribución, proporción y asociación de los elementos químicos de la corteza terrestre, y de las leyes que las condicionan. http://lema.rae.es/drae/?val=geoquimica

Lava. Magma que durante su ascenso a través de la corteza terrestre alcanza la superficie. http://es.wikipedia.org/wiki/Lava

Magma. (del latín *magma* y éste del griego μάγμα, «pasta») es el nombre que reciben las masas de rocas fundidas del interior de la Tierra u otros planetas. Suelen estar compuestos por una mezcla de líquidos, volátiles y sólidos. http://es.wikipedia.org/wiki/Magma

Piroclasto. (del griego πῦρ "fuego" y κλαστός "roto") también conocido como tefra (del griego "ceniza"), es cualquier fragmento sólido de material volcánico expulsado a través de una columna eruptiva arrojado al aire durante una erupción volcánica.

Pumita. (también llamada piedra pómez, jal o liparita) es una roca ígnea volcánica vítrea, con baja densidad (flota en el agua) y muy porosa, de color blanco o gris. http://es.wikipedia.org/wiki/Pumita

Riolita. es una roca ígnea de color gris a rojizo con una textura de granos finos o a veces también vidrio y una composición química muy parecida a la del granito. http://es.wikipedia.org/wiki/Riolita

Roca ígnea. (latín *ignius*, "fuego"). Roca que se forma cuando el magma (roca fundida) se enfría y se solidifica . http://es.wikipedia.org/wiki/Roca_%C3%ADgnea

Silicatos. Son el grupo de minerales de mayor abundancia, pues constituyen más del 95% de la corteza terrestre, además del grupo de más importancia geológica por ser petrogénicos, es decir, los minerales que forman las rocas. Todos los silicatos están compuestos por silicio y oxígeno. Estos elementos pueden estar acompañados de otros entre los que destacan aluminio, hierro, magnesio o calcio. http://es.wikipedia.org/wiki/Silicato

Tropósfera. es la capa de la atmósfera terrestre que está en contacto con la superficie de la Tierra. http://es.wikipedia.org/wiki/Troposfera

Volcán. (Del port. *Volcão*). **1.** m. Abertura en la tierra, y más comúnmente en una montaña, por donde salen de tiempo en tiempo humo, llamas y materias encendidas o derretidas. **2.** m. El mucho fuego, o la violencia del ardor. **3.** m. Pasión ardiente; p. ej., el amor o la ira. http://lema.rae.es/drae/?val=volcan

Vulcanología. (Del lat. *Vulcānus*, Vulcano, dios del fuego, y *-logía*). **1.** f. Estudio de los fenómenos volcánicos. http://lema.rae.es/drae/?val=vulcanologia

VIII. Referencias

Bibliografía:

Chapra, S., 2010, *Métodos numéricos para ingenieros*, Sexta Edición, Mc Graw Hill, México.

Deitel, P., Deitel, H., 2012, *Cómo programar en Java*, Novena Edición, Pearson Educación, México.

Sommerville, I., 2011, *Ingeniería de Software,* Novena Edición, Pearson Educación, México.

Tarbuck E. J., Lutgens F. K. Y Tasa, D, 2005, *Ciencias de la Tierra*, Octava Edición, Prentice Hall, México.

Woods, A. W., 1988, *The fluid dynamics and thermodynamics of eruption columns,* Bulletin of Volcanology vol. 50 , Springer-Verlag, EEUU, p. 169-193.

Recursos Electrónicos:

Bitbucket features, disponible en línea, https://bitbucket.org/features, consultado el 4 de mayo de 2014.

Chandle, J., The history of volcanology, disponible en línea, http://www.ehow.com/about_5397406_history-volcanology.html, consultado el 30 de diciembre del 2013.

Conceptos de la metodología orientada a objetos, disponible en línea, http://profesores.fi-b.unam.mx/carlos/aydoo/conceptos_oo.html, consultado el 3 de abril de 2014.

Conozca más sobre la tecnología Java, disponible en línea, http://www.java.com/es/about/, consultado el 30 de marzo de 2014.

Control de versiones, disponible en línea, http://es.wikipedia.org/wiki/Control_de_versiones, consultado el 4 de mayo de 2014.

Currículo Dr. Juan Manuel Espíndola Castro, disponible en línea, http://www.geofisica.unam.mx/vulcanologia/spanish/personal/doc_pdf/espindola.pdf, consultado el 1 de enero de 2014

Erupción pliniana, disponible en línea, http://es.wikipedia.org/wiki/Erupci%C3%B3n_pliniana, consultado el 5 de enero de 2014.

Erupción volcánica., disponible en línea, http://es.wikipedia.org/wiki/Erupci%C3%B3n_volc%C3%A1nica, consultado el 23 de enero de 2014

Erupciones plinianas, disponible en línea, http://www.uclm.es/profesorado/egcardenas/erup.htm, consultado el 5 de enero de 2014.

Fortran 90 Tutorial, Michigan Technological University, Department of Computer Science, disponible en línea, http://www.cs.mtu.edu/~shene/COURSES/cs201/NOTES/fortranj.html, consultado el 28 de diciembre del 2013.

Índice de explosividad volcánica, disponible en línea, http://es.wikipedia.org/wiki/%C3%8Dndice_de_explosividad_volc%C3%A1nica, consultado el 23 de enero de 2014.

International Association of Volcanology and Chemistry of the Earth, disponible en línea, http://en.wikipedia.org/wiki/International_Association_of_Volcanology_and_Chemistry_of_the_Earth%27s_Interior, consultado el 23 de enero de 2014.

Introduction to Object Oriented Programming Concepts (OOP) and More, disponible en línea, http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep#Conclusion, consultado el 2 de abril de 2014.

Java (programming language), disponible en línea, http://en.wikipedia.org/wiki/Java_%28programming_language%29, consultado el 30 de marzo de 2014.

Java SE Application Design With MVC, disponible en línea, http://www.oracle.com/technetwork/articles/javase/index-142890.html, consultado el 1 de abril de 2014.

Java version history, disponible en línea, http://en.wikipedia.org/wiki/Java_version_history, consultado el 03 de abril de 2014.

Java Virtual Machine, disponible en línea, http://en.wikipedia.org/wiki/Java_virtual_machine, consultado el 20 de abril de 2014

Lenguaje de programación interpretado, disponible en línea, http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_interpretado, consultado el 15 de enero de 2014

Mercurial, disponible en línea, http://es.wikipedia.org/wiki/Mercurial#Proyectos_que_usan_Mercurial, consultado el 4 de mayo de 2014.

Metodología para la vigilancia volcánica. Servicio Geológico colombiano, disponible en línea, http://www.sgc.gov.co/Observatorios-Vulcanologicos/Metodologia-para-la-vigilancia-volcanica.aspx, consultado el 30 de diciembre de 2013.

Microsoft Visual C Sharp, disponible en línea, http://en.wikipedia.org/wiki/Microsoft_Visual_C_Sharp, consultado el 1 de enero de 2013.

Object Oriented Software Development: Object-Oriented Programming, disponible en línea, http://cs.smu.ca/~porter/csc/465/notes/oop.html, consultado el 2 de abril de 2014

Object-oriented programming, disponible en línea, http://en.wikipedia.org/wiki/Object-oriented_programming, consultado el 2 de abril de 2014.

Programación orientada a objetos, disponible en línea, http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos, consultado el 2 de abril de 2014.

Repository, disponible en línea,

http://en.wikipedia.org/wiki/Repository_%28version_control%29, consultado el 4 de mayo de 2014.

Software Engineering, disponible en línea,

http://en.wikipedia.org/wiki/Software_engineering#Criticism, consultado el 4 de mayo de 2014.

The architecture of the Java Virtual Machine, disponible en línea,

http://www.artima.com/insidejvm/ed2/jvm2.html, consultado el 20 de abril de 2014

The Goals of Object Oriented Programming, disponible en línea, http://www.particle.kth.se/~fmi/kurs/PhysicsSimulation/Lectures/02B/oopGoals.html, consultado el 2 de abril de 2014.

The Java Virtual Machine, disponible en línea, http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-1.html#jvms-1.2, consultado el 20 de abril de 2014

Ventajas de la Programación Orientada a Objetos (POO), disponible en línea, http://android-linux.net/12-desarrollo/22-ventajas-de-la-programacion-orientada-a-objetos-poo, consultado el 2 de abril de 2014.

What's new in Java 8, disponible en línea, http://www.oracle.com/technetwork/java/javase/8-whats-new-2157071.html, consultado el 20 de abril de 2014