Universidad
Rey Juan Carlos

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

Curso Académico 2009/2010

Proyecto de Fin de Carrera

SIMULACIÓN MEDIANTE APPLETS DE JAVA DE SISTEMAS DINÁMICOS CON ESCAPE

Autor: Pablo Andrés Sánchez

Tutores: Jesús M. Seoane Sepúlveda
Inés Pérez Mariño

Agradecimientos

Como en cualquier aspecto que se me haya podido presentar en la vida, mis mayores agradecimientos a mi familia y a mis padres en concreto. A pesar de que ellos no me pueden ayudar técnicamente con el trabajo ni con nada relacionado con mi carrera, sí pueden, y de hecho lo hacen, ayudarme moralmente para que no sólo este proyecto sino toda la carrera sean finalizados con éxito. Por tanto mis mayores agradecimientos a mis padres.

También debo dedicar un sitio entre estos agradecimientos a mis tutores del proyecto los Profesores Jesús M. Seoane e Inés Pérez del Departamento de Física de la Universidad Rey Juan Carlos, quienes siempre me han ofrecido su ayuda y me han dado ánimos en los peores momentos.

Resumen

En este documento se va describir el comportamiento de tres sistemas físicos que pueden presentar comportamiento caótico y que están enmarcados dentro de los sistemas dinámicos con escape, que son modelos prototipo en Dinámica Galáctica. Los sistemas dinámicos con escape que se van a describir serán los Sistemas de Hénon-Heiles, Barbanis y Contopoulos. Los tres sistemas describen el comportamiento caótico o no de una partícula y se representan matemáticamente por un conjunto de ecuaciones diferenciales que conforman un sistema dinámico no lineal. Se van a ver diversos métodos para resolver estas ecuaciones de manera aproximada. Finalmente se optará por el método de integración Runge-Kutta de 4º orden por ser uno de los más efectivos en nuestro propósito.

Para poder representar gráficamente lo descrito matemáticamente, se van a dibujar las trayectorias de las partículas según el sistema elegido. Para ello se va a utilizar una aplicación gráfica llamada applet escrita en lenguaje Java orientado a objetos y dividida por tanto en diferentes clases. Este applet irá destinado a la utilización de cualquier usuario que lo requiera. Por ello la aplicación deberá estar pensada para una fácil comprensión y manejo por parte del usuario como uno de los requisitos iniciales y uno de los objetivos finales a cumplir. Para ello se va a describir una guía de uso bastante completa para saber utilizar el applet y para poder explotar al máximo sus posibilidades.

Para que cualquier usuario pueda acceder fácilmente al applet, éste estará integrado en una página web dentro de Internet. Ésta es la ventaja más importante que nos ha llevado a elegir un applet de Java frente a otras posibilidades que proporciona la Informática y la programación concretamente. Se van a ver las ventajas y los inconvenientes del applet frente a esas otras posibilidades.

Índice

1. Introducción

Los sistemas dinámicos se utilizan para explicar y describir fenómenos que ocurren en nuestro entorno, que presentan un cambio o evolución de su estado en un tiempo. Desde el punto de vista matemático los sistemas dinámicos se pueden clasificar en sistemas lineales o no lineales. La dificultad que presentan los sistemas lineales es relativamente baja pudiendo analizar las soluciones con bastante facilidad. En el caso de los sistemas no lineales, pueden dar lugar a dinámicas muy complejas y difíciles de analizar recurriendo así a métodos numéricos para encontrar soluciones aproximadas.

Nuestro applet trata de estos últimos y, en particular, nos centraremos en los sistemas caóticos. La principal característica de un sistema caótico es su dependencia de las condiciones iniciales ya que una mínima diferencia en esas condiciones hace que el sistema evolucione de manera totalmente distinta. En nuestro applet el usuario podrá apreciar cómo para un sistema caótico como es el caso de Hénon-Heiles una partícula puede presentar trayectorias de movimiento totalmente diferentes con un simple cambio en las condiciones iniciales.

El comportamiento caótico ha sido encontrado en una enorme variedad de contextos, desde la Física hasta la Biología, pasando por la Química y la Meteorología. El carácter impredecible de los sistemas caóticos puede ser problemático en ciertos contextos, de modo que en los últimos años se han propuesto distintos esquemas que permiten obtener una respuesta determinada de un sistema dinámico, aplicando para ello pequeñas perturbaciones. Esas técnicas se engloban en al rama de la Dinámica No Lineal denominada “Control del Caos”. Asimismo, se han propuesto distintas técnicas para hacer que ciertos tipos de dinámicas complejas (asociadas a fenómenos dinámicos universales como los transitorios caóticos, las cuencas de atracción fractales y la intermitencia, por citar algunos de los más relevantes) puedan ser controladas. Dada la ubicuidad de este tipo de comportamientos en la Naturaleza, estas técnicas de control tienen múltiples aplicaciones en una enorme variedad de campos.

Los sistemas dinámicos no lineales están compuestos de ecuaciones diferenciales. Las ecuaciones diferenciales aparecen naturalmente al modelar situaciones físicas en las ciencias naturales, ingeniería, y otras disciplinas, donde hay envueltas razones de cambio de una ó varias funciones con respecto a una ó varias variables. Estos modelos varían entre los más sencillos que envuelven una sola ecuación diferencial para una función, hasta otros más complejos que envuelven sistemas de ecuaciones diferenciales acopladas para varias funciones. Usualmente estas ecuaciones están acompañadas de una condición adicional que especifica el estado del sistema en un tiempo o posición inicial. Esto se conoce como la *condición inicial* y junto con la ecuación diferencial forman lo que se conoce como el *problema de valor inicial*. Por lo general, la solución exacta de un problema de valor inicial es imposible ó difícil de

obtener en forma analítica. Por tal razón los métodos numéricos se utilizan para aproximar dichas soluciones como se verá más adelante.

La herramienta para poder representarlos va ser una aplicación informática llamada applet. Dicho applet está programado en un lenguaje orientado a objeto llamado Java.

A pesar de que hubo casos que datan de fechas anteriores, una de las primeras aplicaciones de la Teoría del Caos data del año 1970 cuando el meteorólogo Edward Lorenz dio a conocer un curioso modelo climático que posteriormente fascinaría a muchos físicos por su extraño comportamiento. Antes de Lorenz cabe destacar a Hénon-Heiles que presentó un modelo en 1964 para estudiar las trayectorias estelares de una partícula cualquiera a través de una galaxia. El sistema de Hénon-Heiles es el modelo típico para el estudio de sistemas hamiltonianos conservativos (caóticos) motivados al observar el comportamiento de estrellas en galaxias. Los sistemas conservativos son los que dan origen a fuerzas conservativas ya que provienen de una energía potencial.

Los astrónomos M. Hénon y C. Heiles modelaron el movimiento de una estrella dentro de una galaxia como un sistema hamiltoniano de cuatro variables de estado:

$$H = \frac{1}{2}\left(X^2 + Y^2\right) + \frac{1}{2}\left(x^2 + y^2\right) + x^2 y - \frac{1}{3}y^3$$

Donde X, Y representan la velocidad de la partícula y (x, y) la posición en el espacio.

Este sistema caótico es conservativo ya que da origen a fuerzas conservativas que provienen de un potencial:

$$V(x,y) = \frac{1}{2}\left(x^2 + y^2\right) + x^2 y - \frac{1}{3}y^3$$

y representa el acoplamiento no lineal de dos osciladores armónicos. Las correspondientes ecuaciones de movimiento son:

$$\dot{x} = \frac{\partial H}{\partial p_x} = p_x$$

$$\dot{y} = \frac{\partial H}{\partial p_y} = p_y$$

$$\dot{p}_x = -\frac{\partial H}{\partial x} = -\left(x + 2xy\right)$$

$$\dot{p}_y = -\frac{\partial H}{\partial y} = -\left(y + x^2 - y^2\right)$$

Como el sistema es conservativo un elemento de volumen en el espacio de fases se conserva. El hecho de que el volumen no cambia en sistemas conservativos implica que no hay regiones en el espacio de fases que actúen como ciclos límite ni la existencia de atractores extraños. En sistemas conservativos es posible construir aproximadamente el espacio de fases identificando puntos de equilibrio y analizar la estabilidad de las trayectorias en la vecindad de éstos. Para el Hamiltoniano que estudiamos los puntos fijos son hiperbólicos y elípticos. Un punto hiperbólico o silla se asocia con trayectorias en el espacio de fases que se acercan o se alejan de ésta. Cerca de los puntos elípticos o centrales las trayectorias en el espacio de fases rotan alrededor de éstos. Alrededor de los puntos hiperbólicos los movimientos se hacen inestables a diferencia de los puntos elípticos.

Como ya se ha visto, el sistema de Hénon-Heiles al ser caótico dependiendo de las condiciones iniciales, la trayectoria o movimiento de la partícula en una galaxia puede variar mucho. Las siguientes figuras (figura 1 y figura 2) ilustran el comportamiento de este sistema bajo dos condiciones iniciales diferentes, de las cuales la primera conduce a comportamiento regular mientras que la segunda conduce a movimiento caótico:

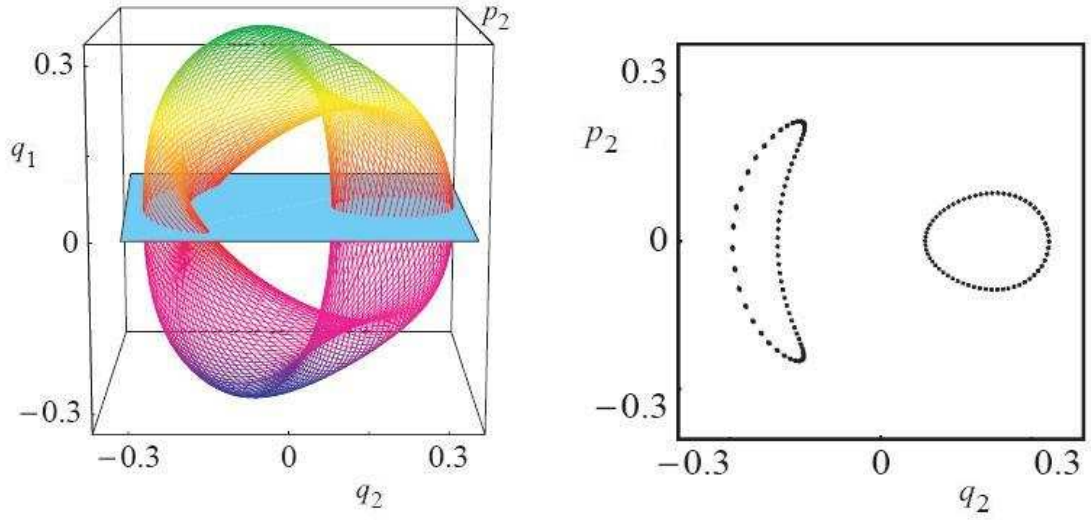

Figura 1. Dinámica del sistema de Hénon-Heiles bajo condiciones que conducen a comportamiento regular. Al lado derecho la superficie de Poincaré (cortes a energía constante)

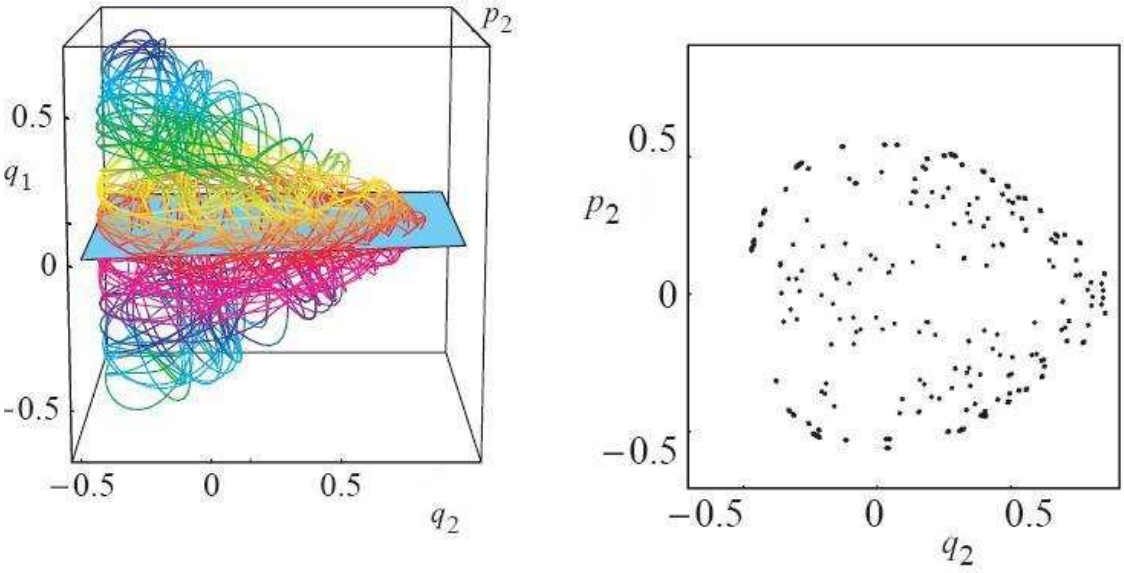

Figura 2. Dinámica del sistema de Hénon-Heiles bajo condiciones que conducen a comportamiento caótico. Al lado derecho la superficie de Poincaré.

Cuando los sistemas hamiltonianos son conservativos, como es este caso, el comportamiento caótico puede ser distinguido de un comportamiento cuasiperiódico analizando la apariencia del conjunto de puntos generado por las trayectorias en una superficie conocida como mapa de Poincaré.

El sistema de Hénon-Heiles es un hamiltoniano con 3 salidas de escape para la partícula como se puede apreciar en la figura 3:

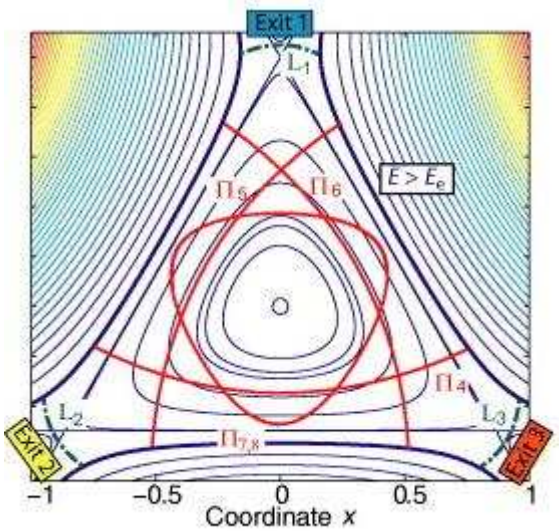

Figura 3. Representación del movimiento de una partícula en el sistema de Hénon-Heiles y las tres vías de escape que presenta el sistema

donde *coordinate x* es coordenada de las x y *exit 1, exit 2,* y *exit 3* son las salidas 1, 2 y 3 respectivamente.

Además del potencial de Hénon-Heiles como hamiltoniano abierto, existen otros potenciales representativos de los hamiltonianos abiertos como es el caso del potencial de Barbanis con dos salidas de escape o fugas y el potencial de Contopoulos que tiene 4 fugas como se puede observar en las figuras 4 y 5 respectivamente.

El potencial de Barbanis tiene el siguiente hamiltoniano:

$$H = \frac{1}{2}\left(X^2 + Y^2\right) + \frac{1}{2}\left(x^2 + y^2\right) - xy^2$$

Este hamiltoniano es simétrico con respecto al eje de las *y* con dos salidas para los valores de la energía E > 1/8 como se puede apreciar en la figura 4. Su potencial se denomina potencial Barbanis en la comunidad de la Química y es utilizado ampliamente en la dinámica cuántica.

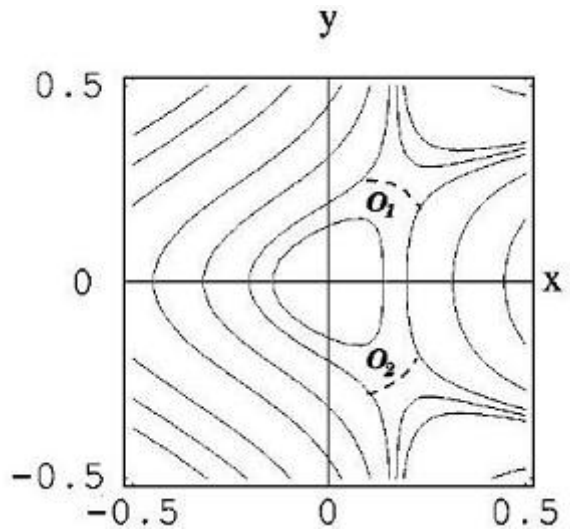

Figura 4. Representación de las curvas de nivel del sistema de Barbanis en el que pueden observarse sus dos vías de escape.

El potencial de Contopoulos tiene el siguiente hamiltoniano:

$$H = \frac{1}{2}\left(X^2 + Y^2\right) + \frac{1}{2}\left(x^2 + y^2\right) - x^2 y^2$$

Este hamiltoniano es invariable con respecto al eje de las *x* o de las *y* con cuatro escapes o salidas para los valores de la energía E > 1/4 como se muestra en la figura 5. Se utiliza en los movimientos galácticos típicos.

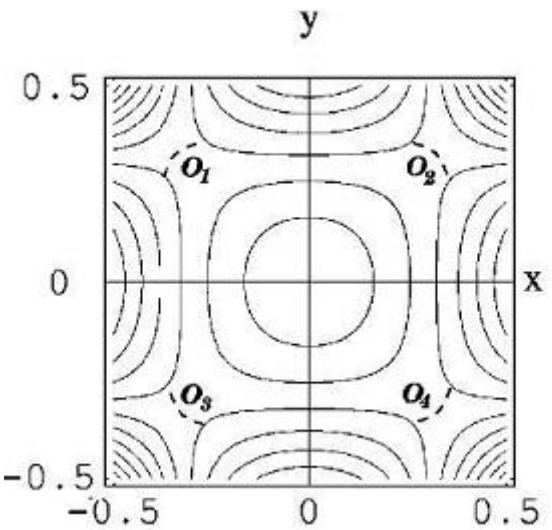

Figura 5. Representación de las curvas de nivel del sistema de Contopoulos en el que pueden observarse sus cuatro vías de escape.

2. Objetivos y metodología

El principal objetivo de este proyecto es diseñar un applet de Java que simule los movimientos galácticos que puedan presentar comportamientos caóticos. En concreto obtener trayectorias de una o dos partículas en un espacio ficticio simulando el movimiento de las estrellas en el espacio real. Los sistemas caóticos galácticos que hemos utilizado son los de Hénon-Heiles, Barbanis y Contopoulos que ya hemos visto antes.

Para ello se va a utilizar una aplicación informática que, siendo compleja en su interior, deba ser simple de cara al usuario. Mediante esta aplicación el usuario podrá estudiar las variaciones en el comportamiento del sistema, dependiendo de los parámetros que introduzca por pantalla. La interfaz mostrará, dependiendo de los parámetros introducidos por el usuario, la trayectoria de una partícula (o dos, en el caso

de que se seleccione la opción “comparación”) según sea el sistema dinámico galáctico seleccionado Hénon-Heiles, Barbanis o Contopoulos.

Por todo ello, se debe utilizar una aplicación informática que cumpla estos requisitos:

- que sea simple de cara al usuario
- que sea capaz de simular la trayectoria de una o dos partículas según los sistemas de Hénon-Heiles, Barbanis o Contopoulos
- que sea fácilmente accesible para todo el que quiera utilizar dicha aplicación.

Se ha elegido un applet en Java como medio para cumplir estos tres requisitos fundamentales que nos propusimos al principio. Los applets son aplicaciones informáticas insertadas en Internet mediante una página web, por lo que el tercer requisito lo cumple con creces ya que no hay nada más accesible hoy en día y universal que Internet. Además actualmente Java proporciona un paquete Swing que proporciona mucha potencia y complejidad de cara a las aplicaciones gráficas por lo que el primer requisito se cumple también. Y, por último, los applets proporcionan una gama amplia de posibilidades de cara a la interactuación de la aplicación con el usuario, por lo que el segundo requisito es posible cumplirlo.

En conclusión, se va a diseñar una aplicación que sea capaz de representar, de la manera más sencilla para el usuario, los sistemas dinámicos galácticos mencionados.

2.1. Descripción del problema

De las fórmulas de los hamiltonianos de Hénon-Heiles, Barbanis y Contopoulos podemos obtener los potenciales de los tres sistemas.

El potencial de Hénon-Heiles es:

$$V(x,y)=\frac{1}{2}\left(x\ +y^2\right)-x^2y-\frac{1}{3}y^3$$

Las ecuaciones diferenciales respecto de *x* e *y* son:

$$\frac{d^2x}{dt^2}=-x-2xy$$

$$\frac{d^2y}{dt^2}=-y-x^2+y^2$$

El potencial de Barbanis es:

$$V(x,y)=\frac{1}{2}\left(x^2+y^2\right)-xy^2$$

Las ecuaciones diferenciales respecto de *x* e *y* son:

$$\frac{d\ x}{dt}=-x+y^2$$
$$\frac{d^2y}{dt^2}=-y+2xy$$

El potencial de Contopoulos es:

$$V(x,y)=\frac{1}{2}\left(x^2+y^2\right)-x^2y^2$$

Las ecuaciones diferenciales respecto de *x* e *y* son:

$$\frac{d^2x}{dt^2}=-x+2xy^2$$
$$\frac{d^2y}{dt^2}=-y+2x^2y$$

Las variables *x* e *y* de las fórmulas de los tres modelos se refieren a la posición de la partícula en el plano. Si ésta está en la zona de influencia del potencial, la *x* y la *y* estarán enmarcadas dentro de la región del potencial. Las variables *X* e *Y* son los vectores o componentes de la velocidad de la partícula.

2.2. Estudio de alternativas

Los movimientos que presentan comportamientos caóticos se expresan mediante ecuaciones diferenciales. El problema de las ecuaciones diferenciales es que en algunas de las ocasiones no es posible encontrar una solución de forma analítica debido a la complejidad de éstas. Para poder conseguir una solución aunque aproximada se pueden utilizar varios métodos de integración numéricas:

Método de Euler

Este método es el método más sencillo que presenta una solución numérica a ecuaciones diferenciales ordinarias y, en concreto a las que hemos visto anteriormente. Se basa en aproximaciones numéricas para la resolución de dichas ecuaciones según la siguiente fórmula:

$$y_{n+1} = y_n + hf(x_n, y_n) + O(h^2)$$

La derivada en el punto inicial de cada intervalo se extrapola para encontrar el siguiente valor de la función como se muestra en la figura 6:

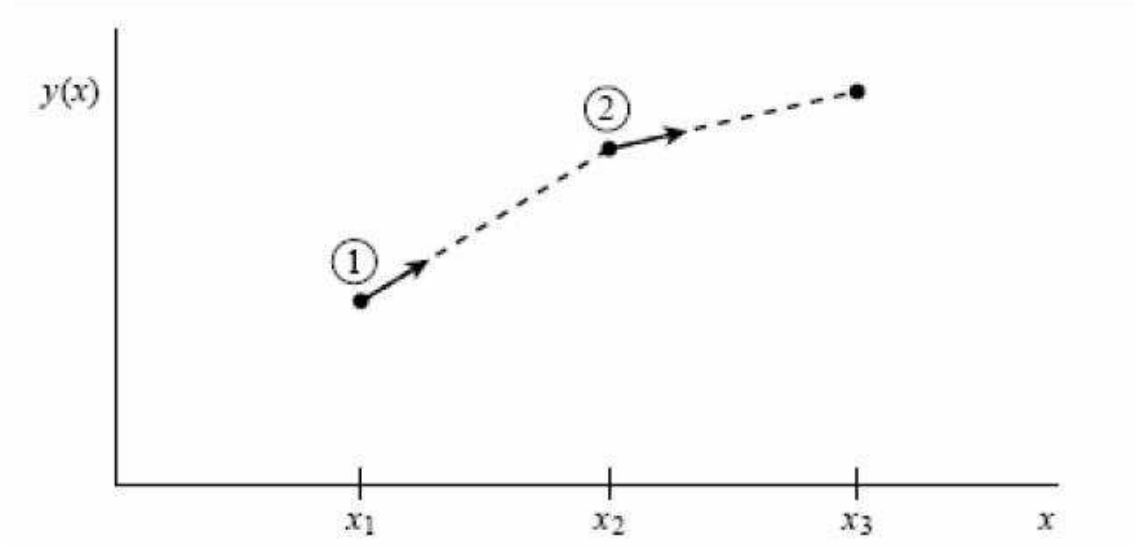

Figura 6. Método de Euler.

Este método tiene una precisión de primer orden. Por lo tanto el problema que presenta es que se trata de un método algo inexacto en un principio y, según va avanzando el método, la inexactitud va aumentando sistemáticamente. Por ello, es un método válido para valores de *h* no muy elevados, donde *h* es el paso de integración. Cuanto más pequeño sea el valor de *h* más exactitud habrá ya que con un valor de *h* elevado el error es también elevado como se puede observar en la figura 7:

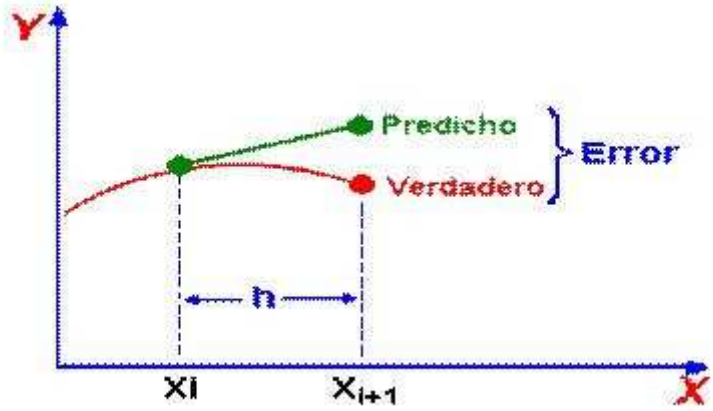

Figura 7. Aquí se puede apreciar el margen de error para un tramo *h* que presenta el método de Euler en relación con la función verdadera

Para un *h* constante el error será tanto mayor cuanto más nos alejemos del punto inicial, como puede apreciarse en la gráfica siguiente (figura 8) en la que comparamos las dos soluciones aproximadas con la solución exacta:

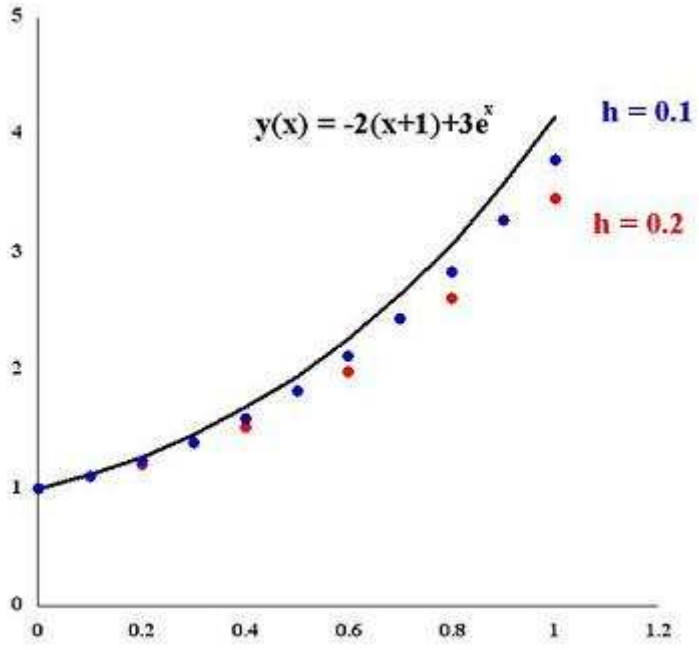

Figura 8. Los puntos azules y rojos son los que seguiría la función si se representara mediante el método de Euler para los valores de *h*=0.1 y h=0.2

Este método presenta una actualización o mejora llamada método de Euler mejorado. La diferencia con el método de Euler es que partiendo de un punto inicial calculamos también el valor de un punto medio para calcular el valor de la función para el tamaño real de ese intervalo como se muestra en la figura 9:

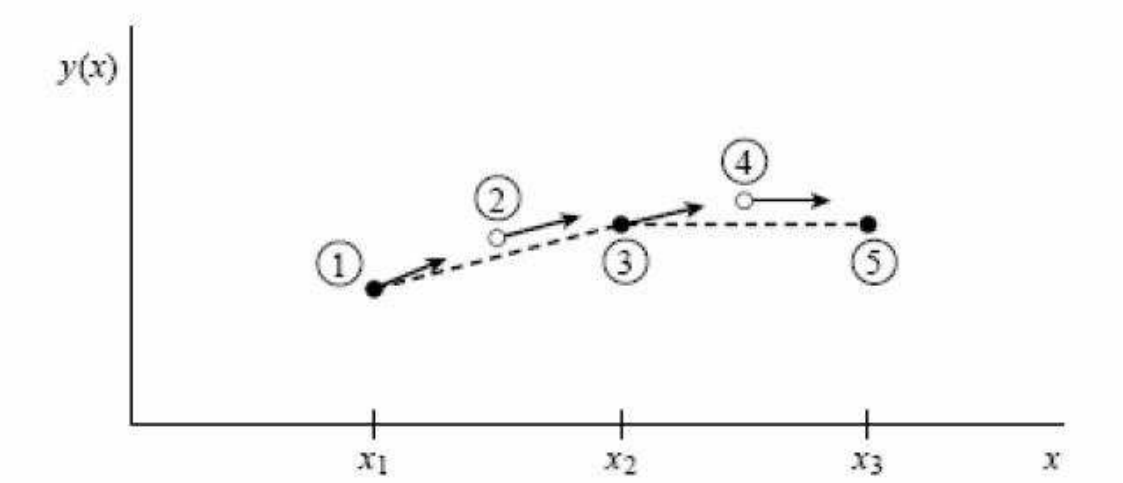

Figura 9. Método de Euler mejorado o de punto medio.

Las ecuaciones del método de Euler mejorado son las siguientes:

$$k_1 = hf\left(x_n, y_n\right)$$
$$k_2 = hf\left(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1\right)$$
$$y_{n+1} = y_n + k_2 + O\left(h^3\right)$$

Aún así la imprecisión sigue aumentando según avanza el método y, en mayor medida para valores elevados de h. Esta situación mejora al considerar métodos con un orden de convergencia más alto como los llamados métodos Runge-Kutta.

Métodos de Runge-Kutta

El método de Runge-Kutta es un refinamiento del método de Euler. No es sólo un método sino una importante familia de métodos iterativos tanto implícitos como explícitos para aproximar las soluciones de ecuaciones diferenciales ordinarias. Usa,

para cada paso, varios intermedios que ayudan a disminuir el error. El método que veremos será el de cuarto orden, que en cada paso evalúa la derivada 4 veces como podemos observar en la figura 7:

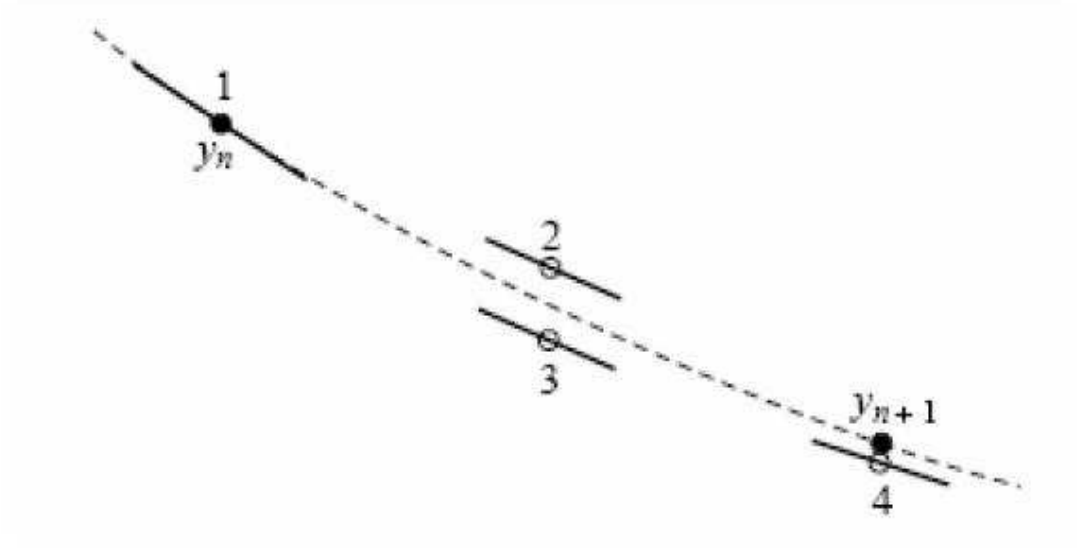

Figura 10. En cada paso se evalúa la derivada una vez para el punto inicial, otra para el punto final y otras 2 para los posibles puntos intermedios consiguiendo con todo ello el valor final de la función (en la imagen el punto negro)

En cada paso se deben calcular 4 valores k1, k2, k3, k4 y elegir un tamaño del intervalo llamado h. Según el procedimiento ordinario de Runge-Kutta, a partir del valor de y en el instante x se puede determinar el valor de y en el instante x+h según la siguiente fórmula:

$$k_1 = hf(x_n, y_n)$$
$$k_2 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right)$$
$$k_3 = hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right)$$
$$k_4 = hf(x_n + h, y_n + k_3)$$
$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

Así, el siguiente valor (y_{n+1}) es determinado por el presente valor (y_n) más el producto del tamaño del intervalo (h) por una pendiente estimada. La pendiente estimada es un promedio ponderado de pendientes k1, k2, k3 y k4. Se trata de cuatro coeficientes que se calculan mediante un algoritmo reiterativo.

De este modo se obtiene un valor con una mejor aproximación, de tal forma que el error acumulado con las sucesivas iteraciones para calcular el valor de la función a lo largo del tiempo disminuye respecto al método de Euler. Por ello este método ha sido el utilizado en nuestro applet consiguiendo una elevada aproximación de los valores.

Otras alternativas

El mundo de la informática está actualmente muy avanzado en lo que a aplicaciones gráficas se refiere. Por ello, existen muchas formas de crear una aplicación gráfica e insertarla en una página web. Entre estas formas se encuentra la que se ha elegido para el sistema que es el applet en Java, pero también se podían haber utilizado otros sistemas proporcionados por la informática como es el caso de programar directamente la página web en HTML e ir insertando en el código HTML funciones JavaScript (muy parecidas a las funciones de Java) y dándole color y forma utilizando hojas de estilo como por ejemplo CSS (hojas de estilo para personalizar el formato de una página web).

En particular, se ha escogido el applet de Java porque el proyecto se centra en la aplicación gráfica no en el formato de la página web donde sería más util HTML, javascript y CSS.

2.3. Metodología empleada

Para realizar este applet se ha escogido un sistema iterativo por etapas con 4 fases fundamentales:

Primera fase: análisis de requisitos

En esta primera etapa se plantean los objetivos que se quieren alcanzar con la ejecución del proyecto. Para ello se deben conocer los requisitos y necesidades que se

nos van a plantear. De este modo es importante describir bien los requisitos del sistema antes de avanzar con la siguiente fase.

Una de las primeras tareas a realizar es la comprensión del problema para poder saber que requisitos debe cumplir.

Segunda fase: Diseño

En esta fase hay que hacer hincapié en posibles soluciones que resuelvan el problema cumpliendo los requisitos de la primera fase. De este modo se van a plantear métodos que puedan ser óptimos para resolverlo. Después hay que describir las ventajas e inconvenientes de cada método y, analizando y comparando estas ventajas e inconvenientes con los requisitos que debe cumplir el sistema, poder elegir el método óptimo para abordar el problema. En esta fase se deben definir bien las clases que va a tener el applet y la función de cada una de ellas.

Tercera fase: Realización del código

En esta fase se codifica en lenguaje Java todas las funcionalidades de cada clase definida en el apartado anterior tratando cada clase como si fuera un objeto interrelacionado con las otras clases. Para ello, se realizan métodos y se utilizan los atributos o propiedades necesarias para que cada clase cumpla con lo descrito en la fase anterior.

Es de vital importancia ir probando cada modificación importante del código para que no afecte a lo hecho anteriormente ya que estamos programando un sistema basado en clases interrelacionadas entre sí y cada cambio en una clase afecta a las demás.

Cuarta fase: Control y optimización del Software

Se trata de una fase de control y optimización del código si es necesario. En esta fase se comprueba que los resultados obtenidos de nuestro código cumplan los requisitos iniciales y en definitiva resuelva el problema planteado en un inicio. En caso que no se cumplan todos los requisitos y objetivos iniciales habrá que introducir mejoras optimizando el código.

3. Descripción informática

Para poder entender cómo es el proyecto y la descripción informática que a continuación se va a exponer, lo primero es entender qué es un applet, las causas de escoger un applet y las ventajas que tiene éste.

Un ***applet*** es un componente de una *aplicación* que se ejecuta en el contexto de otro programa, por ejemplo, un navegador web. El *applet* debe ejecutarse en un *contenedor*, que lo proporciona un programa anfitrión, mediante un *plugin*, o en aplicaciones como teléfonos móviles que soportan el modelo de programación por *applets*.

Entonces, ¿qué diferencia hay entre un applet y un programa normal? A diferencia de un *programa*, un *applet* no puede ejecutarse de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un *applet* normalmente lleva a cabo una función muy específica que carece de uso independiente. En resumen, un applet es un programa que se beneficia de la universalidad y facilidad de la red sin perder la potencia que otorga un lenguaje como Java.

Como con cualquier programa escrito en el lenguaje que sea, un applet sigue una serie de pasos para el desarrollo y su depuración. En este caso concreto, se trata de un applet con una interacción con el usuario muy importante ya que finalmente es el que va utilizarlo. Para ello debe entenderlo sin tener conocimientos de programación, por lo que depurar el código es muy importante. A continuación se van a describir las funcionalidades, objetivos y pasos de desarrollo de dicho applet.

3.1 Especificación

En primer lugar hay que saber qué se puede esperar de nuestro programa y cómo se quiere que el usuario lo tenga ante sí. Para ello se deben cumplir tres requisitos fundamentales:

- Se deben cumplir los objetivos técnicos del programa como es el caso de simular con exactitud el movimiento de la partícula.

- Hay que conseguir que el programa sea entendible para futuras mejoras y posibles usos que pueda tener en un ámbito mayor. Por ello el código debe estar claro, limpio y estructurado además de comentado debidamente.
- Por último el programa tiene que ser accesible y fácilmente entendible por el usuario que al fin y al cabo es el que va utilizar el applet.
- El applet debe cumplir distintas funcionalidades ninguna más relevante que otra para alcanzar los requisitos arriba descritos.
- El programa debe estar desarrollado en forma de applet y en Java, siguiendo el paradigma de un lenguaje orientado a objetos.
- El applet debe estar integrado en un navegador web.
- El programa contará con una interfaz gráfica sencilla y clara.
- El usuario contará con una ventana dónde introducir los parámetros de entrada como 'x' inicial, 'y' inicial, energía inicial… que éste desee.
- Los datos se guardarán en una lista que el usuario podrá consultar para un mayor seguimiento del proceso.
- Cuenta con los botones de 'Grafica' y 'NuevaGrafica' para pintar y repintar, respectivamente.
- Este applet debe representar el movimiento de una partícula en el espacio según el Sistema de Hénon-Heiles, el de Barbanis o el de Contopoulos. El usuario elegirá qué sistema prefiere seleccionándolo en el applet con los botones para tal uso.
- La interfaz debe proporcionar al usuario la posibilidad de conocer la trayectoria que ha ido tomando la partícula en el espacio. Podrá hacerlo acelerando o ralentizando el movimiento e incluso parándolo para mejor observación de éste.
- También debe contar con la posibilidad de modificar el tamaño de la gráfica para observar con más detalle la trayectoria de la partícula.
- El applet debe tener un tamaño adecuado para poder ser fácilmente visible en el navegador sin necesidad de barras desplazadoras ni nada por el estilo que entorpezca su comprensión y uso.

A continuación se va a explicar la metodología del applet en sí con algo más de detalle para su fácil comprensión y uso. A pesar de que el uso del applet no reporta mucha complejidad sí que es conveniente una pequeña explicación de cómo funciona para que pueda utilizarlo convenientemente. No se va a profundizar demasiado ya que el applet va dirigido a gente que tenga ciertas nociones de Física o alguna noción técnica o informática a pesar que puede ser consultada por cualquier persona al estar en una dirección de Internet pública.

- En primer lugar hay que introducir los datos o parámetros de entrada necesarios. Esto te lo va a ir pidiendo una etiqueta o *label* comprendida dentro de la Ventana Interna llamada 'Datos de Entrada'. De este modo según vaya pidiendo un dato el usuario lo escribirá en el campo de texto editable justo debajo de dicha etiqueta.

- Cabe resaltar una restricción a la hora de introducir los datos. Éstos deben ser datos válidos o aparecerá un cuadro de advertencia avisando de que el dato introducido no es un dato numérico o ha sido introducido con un formato erróneo. Por ejemplo, para introducir el 'Número de Pasos' que debe seguir la gráfica, el usuario no puede escribir 100,5 pasos porque no es un formato válido.
- Antes de hacer nada el usuario deberá elegir el sistema caótico que seguirá la partícula. Para ello deberá elegir uno de los 3 posibles. Por defecto está seleccionado Hénon-Heiles.
- Una vez metidos todos los datos el usuario estará en disposición de dibujar la gráfica con el botón 'Gráfica'. Antes de presionar el botón el usuario puede elegir que el trazado de la gráfica se realice con animación para poder seguir la trayectoria de ésta. Para ello, antes de presionar el botón 'Gráfica', el usuario debe hacer clic sobre el *check* llamado 'animación'.
- Cuando comience a realizar el trazo de la trayectoria de la partícula el usuario podrá acelerar o ralentizar el trazo mediante una barra desplazadora ubicada justo debajo del *check* 'animación'. Incluso podrá pausar la ejecución del trazo para ver con más detalle la trayectoria seguida por la partícula. Para ello bastará con hacer clic en el "Lienzo" sobre el que se está dibujando la gráfica. Para reanudar la ejecución no hay más que hacer clic de nuevo.
- Una vez la gráfica esté dibujada, el usuario puede elegir una serie de opciones como, por ejemplo, hacer zoom sobre la gráfica mediante la barra desplazadora que está debajo de la gráfica. También podrá mover la gráfica para arriba, abajo, derecha o izquierda presionando los botones justo debajo de la barra de zoom.
- Si el usuario quiere comparar la gráfica que acaba de dibujar con la que va a dibujar, tendrá que seleccionar la opción 'Comparar' antes de dibujar la nueva gráfica.
- Para dibujar una nueva gráfica no hay más que presionar el botón 'NuevaGráfica' y comenzar de nuevo el proceso. Si antes de presionar 'NuevaGrafica', el usuario ha elegido la opción 'Comparar' se dibujarán 2 gráficas, la nueva y la anterior con la que se quiere comparar.
- A modo de facilitar el uso, ya que se trata de muchos parámetros de entrada que el usuario debe introducir, existe la opción de poder ver los datos introducidos en las ventanas 'Funcion1' y 'Funcion2' que tendrán una lista con todos los parámetros que actúan en el proceso. 'Función2' sólo se utilizará cuando se quieran comparar dos gráficas, siendo 'Función2' la nueva gráfica y 'Función1' la antigua. Para el resto de situaciones 'Funcion1' será la única que contendrá datos.

En la figura 8 se puede ver el diagrama de flujo para el uso del applet:

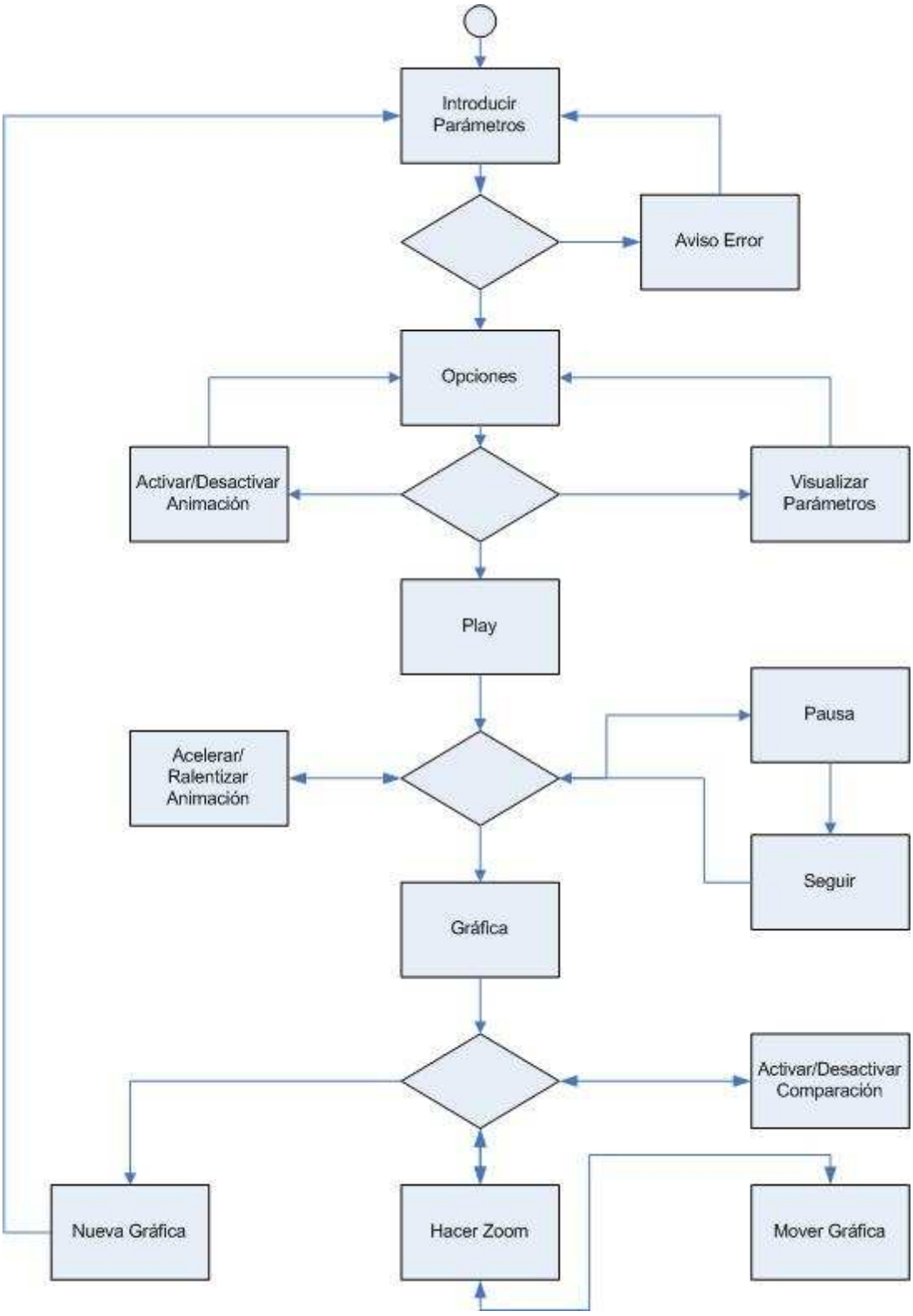

Figura 11. Diagrama de uso. Ejemplo de sesión de trabajo con el applet

3.2. Diseño

Una vez son conocidos las especificaciones y requisitos que el programa debe cumplir, ya se puede plantear el modelo que haga cumplir esas expectativas. Parece claro que debe haber una clase principal donde se gestione el applet controlando la distribución, creación y comportamiento de sus componentes. Por tanto, se ha decidido dividir el proyecto en dos grandes clases, una dedicada al aspecto técnico e informático llamada 'Main' y la otra dedicada al aspecto físico y matemático denominada 'Lienzo'. De esta manera poder simular nuevos modelos caóticos sería tan sencillo como poner una componente más en el applet con el nombre del nuevo modelo caótico y añadir, nunca quitar, código en la clase 'Lienzo'. De este modo, con la clase 'Main' se tiene un bosquejo para futuros proyectos y se ahorra así la costosa labor de creación, diseño y comportamiento del applet.

En un principio se pensó en dedicar una clase a cada una de las funciones de Rungekutta. Éste, por tanto, sería un método de la clase 'Lienzo' donde se crearían objetos de las clases 'Funcion_x', 'Funcion_y', 'Energia',…Estas clases contendrían un único método que devolviera a la clase el valor de 'Funcion_x', 'Funcion_y', 'Energia',…, respectivamente. A pesar de que el código quizás ganara en modularización y encapsulación, se determinó como una pérdida de recursos dedicar una clase a dichas funciones que no sobrepasan el rango de simples métodos, tanto por dimensiones del código como por relevancia y dificultad del mismo.

En resumen, el proyecto contará con las 2 clases antes mencionadas. A continuación se va a realizar una descripción más detallada de ambas:

Clase Main

Será lo primero que se ejecuta ya que la clase 'Lienzo' está conectada a ésta de tal manera que si la clase 'Main' desaparece 'Lienzo' también, debido a que no tiene vida sin su existencia, es decir, ambas clases estarán conectadas mediante una relación de agregación de composición.

Al ser la clase 'Main' lo primero que se ejecuta, debe contener el método 'init' que se invoca al inicializarse el applet, por lo que debe contener la configuración de

componentes (botones, imágenes, etc.). Es decir, es el método que funciona como gestor del applet. Se trata de la clase principal de donde surge todo, es decir, donde se crean los objetos predefinidos por Java, el objeto de la clase Lienzo para poder dibujar y las componentes de las que va a contar el Applet. También se encargará de la distribución dentro del Applet de dichas componentes y de su comportamiento ante acciones externas del usuario haciendo posible la interacción usuario/Applet.

A continuación se va a hacer una breve descripción de las componentes usadas en el proyecto para que sabiendo la funcionalidad de cada componente resulte más sencillo saber la funcionalidad del applet.

Java proporciona dos grandes paquetes para trabajar con interfaces gráficas AWT y Swing. En este proyecto se ha escogido Swing ya que se trata de la versión mejorada de AWT sobre todo en lo que concierne a *popups* o iconos para ayudar o como una guía que AWT no tiene.

Cabe destacar que para la clase 'Lienzo', encargada de dibujar los movimientos en el applet, se ha usado una clase del paquete AWT llamada 'Canvas'. Se ha utilizado frente a un *JPanel* de Swing debido a que 'Canvas' está específicamente diseñada para dibujar mientras que *JPanel* es un contenedor donde guardar cualquier cosa. Pero la causa principal que ha llevado a decantarse por el 'Canvas' ha sido que deja más libertad al programador frente al *JPanel* que está demasiado prefabricado ya que la clase 'Canvas' en sí no hace prácticamente nada; el programador debe definir una subclase de 'Canvas' a la que el AWT le envía todos los eventos de ratón y teclado. Por decirlo de alguna manera es un Lienzo en blanco donde dibujar cualquier cosa.

De las componentes del paquete Swing cabe destacar las siguientes:

- JInternalFrame: es una clase que se utiliza como una ventana dentro de otra ventana, es decir, como contenedor. Se ha utilizado con fines decorativos, ya que tiene más potencia en ese aspecto que un *JPanel*, otorgando un diseño más intuitivo y atractivo que éste.
- JPanel: es un contenedor que se ha utilizado para agrupar varias componentes y organizarlas, funcionando como el verdadero container de componentes.
- JLabel: se trata de una etiqueta representativa ideal para proporcionar instrucciones para el usuario y para mostrar información que no debe ser modificada por éste. En este caso ha sido utilizada para proporcionar instrucciones al usuario a la hora de introducir el valor de las variables y para mostrar el valor de cada variable en la ventana resumen.
- JButton: es una clase muy útil cuando una aplicación debe recibir instrucciones específicas por parte del usuario antes de proceder. Por

ejemplo, cuando un usuario tiene que introducir datos, y luego quiere realizar una tarea, un botón se puede utilizar para instruir el applet o la aplicación al proceder cuando la entrada de datos se ha completado. En el applet que nos ocupa se ha utilizado para ejecutar acciones como pintar o reiniciar el applet.

- JTextField: se trata de un campo de texto editable por el usuario y que resulta clave para hacer posible la interacción usuario-aplicación. En este caso se ha utilizado como introducción de las variables por parte del usuario. Por ello va asociado a un *JLabel* o etiqueta que va ordenando la variable cuyo valor debe introducir el usuario a través del *JTextField*.
- JRadioButton: es una clase destinada a la elección de un elemento entre un conjunto de elementos integrados en otro componente llamado *ButtonGroup* que otorga exclusividad a la elección de un elemento o de otro. En este caso, es utilizado para que el usuario le diga al applet que sistema caótico quiere seguir ya sea Barbanis o Hénon-Heiles.
- JCheckBox: al igual que en el caso de *JRadioButton* se trata de una clase para la elección de un elemento con la única diferencia que no está integrado en un conjunto de elementos. En este applet el usuario puede elegir si dibujar la gráfica con animación o sin ella, si compararla con otra…todo ello seleccionando o no elementos de esta clase.
- JComboBox: es una clase destinada también a la elección excluyente, es decir que está integrada en un conjunto de elementos. Para los casos que se traten de muchos elementos que puedan ocupar mucho espacio en el applet esta clase resulta muy útil. En este caso ha sido utilizado para mostrar el valor de cada una de las variables en la ventana informativa de resumen.
- JOptionPane: con esta clase se pueden controlar y gestionar *popups* o ventanas emergentes de información o de ayuda que pueden resultar muy útiles e intuitivas para el usuario. En el applet que nos ocupa lo hemos utilizado en la introducción de las variables por el usuario para los casos que hayan introducido algún dato erróneo: texto, números en formato erróneo…
- JScrollBar: esta clase define una barra desplazadora que permite al usuario elegir un valor oscilante entre dos extremos: un máximo y un mínimo. En este caso lo hemos utilizado para modificar el tamaño de la gráfica haciendo zoom sobre ella o para aumentar o disminuir la velocidad del trazado de la gráfica en caso de animación.

En el applet se han usado tres ventanas con sus tres paneles correspondientes:

- *JInternalFrame* 'frame_datosentrada': esta ventana contiene al panel 'datos_entrada' para la entrada de datos por parte del usuario. Éste está compuesto por un *JTextField* 'var' donde el usuario introduce los datos y un *JLabel* 'etiqueta_variable' donde el applet indica al usuario que dato debe introducir en cada momento.

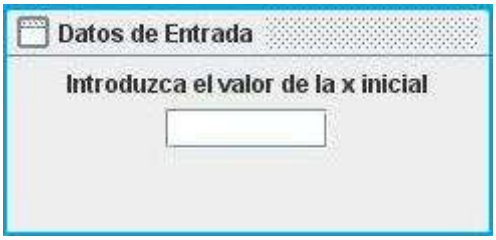

Figura 12. Ventana de datos de entrada

- *JInternalFrame* 'frame_resumen': esta ventana contiene el panel 'panel_resumen' para mostrar el valor de cada una de las variables después que el usuario introdujera los datos en la ventana anteriormente citada. Contiene un *JComboBox* 'selector' donde se elige la variable de la que se quiere saber el valor y un *JLabel* 'dato' que contiene el nombre de esa variable y su valor correspondiente. Para el caso de *JInternalFrame,* 'frame_resumen2' a nivel de diseño es lo mismo con excepción del nombre de las variables y a nivel de implementación muestra las variables y sus valores para el caso de una segunda función dibujada a la hora de realizar una comparación.

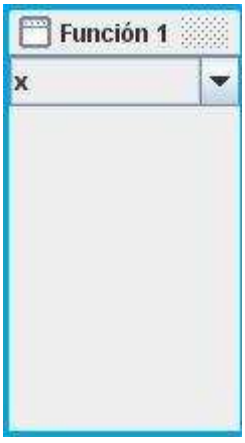

Figura 13.
Ventana Resumen 1

Figura 14.
Ventana Resumen 2

- *JInternalFrame* ‘dibujo’: esta ventana alberga la gráfica que se va dibujar y contiene un objeto de la clase ‘Lienzo’ que es una extensión de la clase predefinida por Java llamada ‘Canvas’.

Figura 15. Ventana donde se dibuja

Clase Lienzo

Esta clase es la encargada de integrar los sistemas (Hénon-Heiles, Barbanis y Contopoulos) con el método de Runge-Kutta a partir de los valores introducidos por el usuario y representar la gráfica obtenida de la integración de dichos valores. Una vez tenemos los valores para dibujar la función, tenemos que adaptarlos al sistema de coordenadas de Java que es diferente al habitual como se explicará más adelante.

El applet resultante en su estado inicial por defecto es el siguiente:

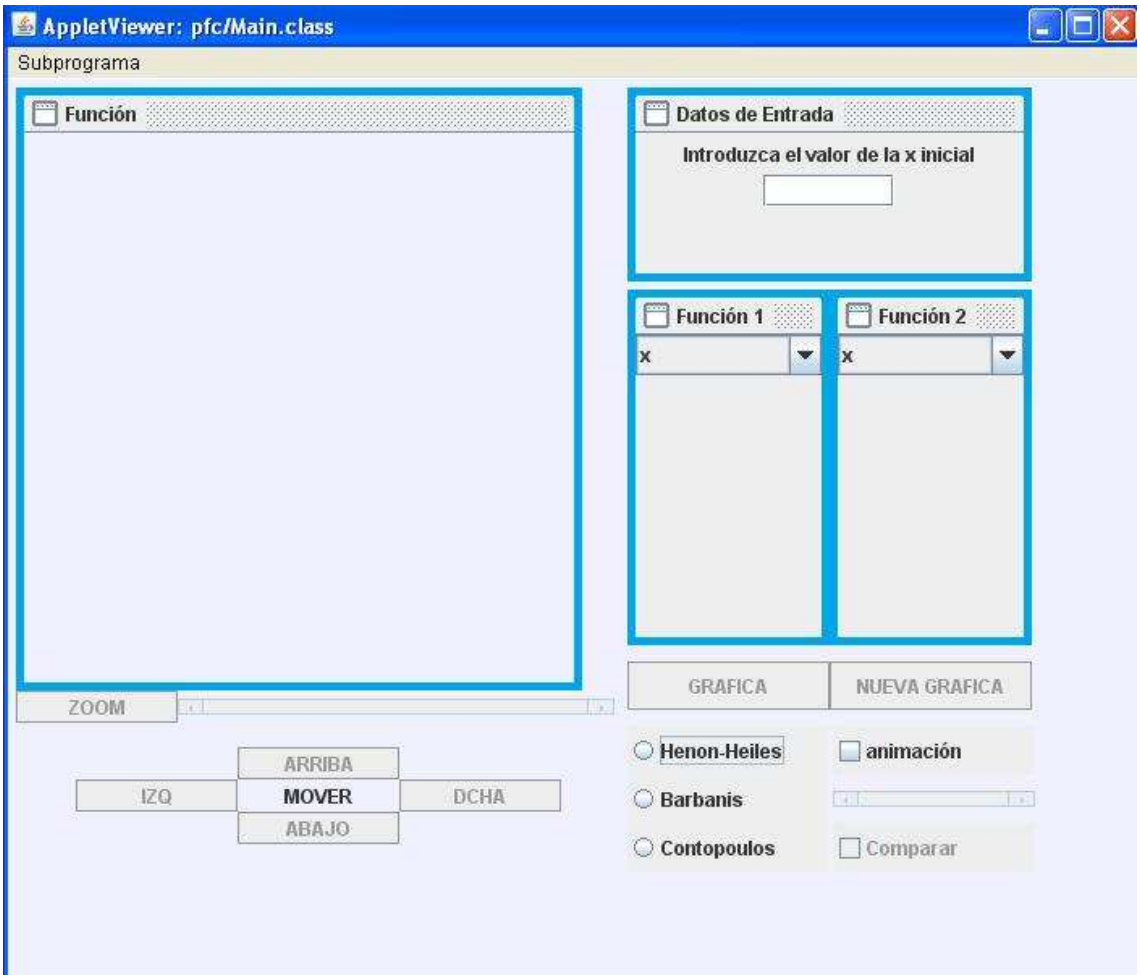

Figura 16. Versión final del applet

A continuación se muestra el diagrama estático de clases de la aplicación:

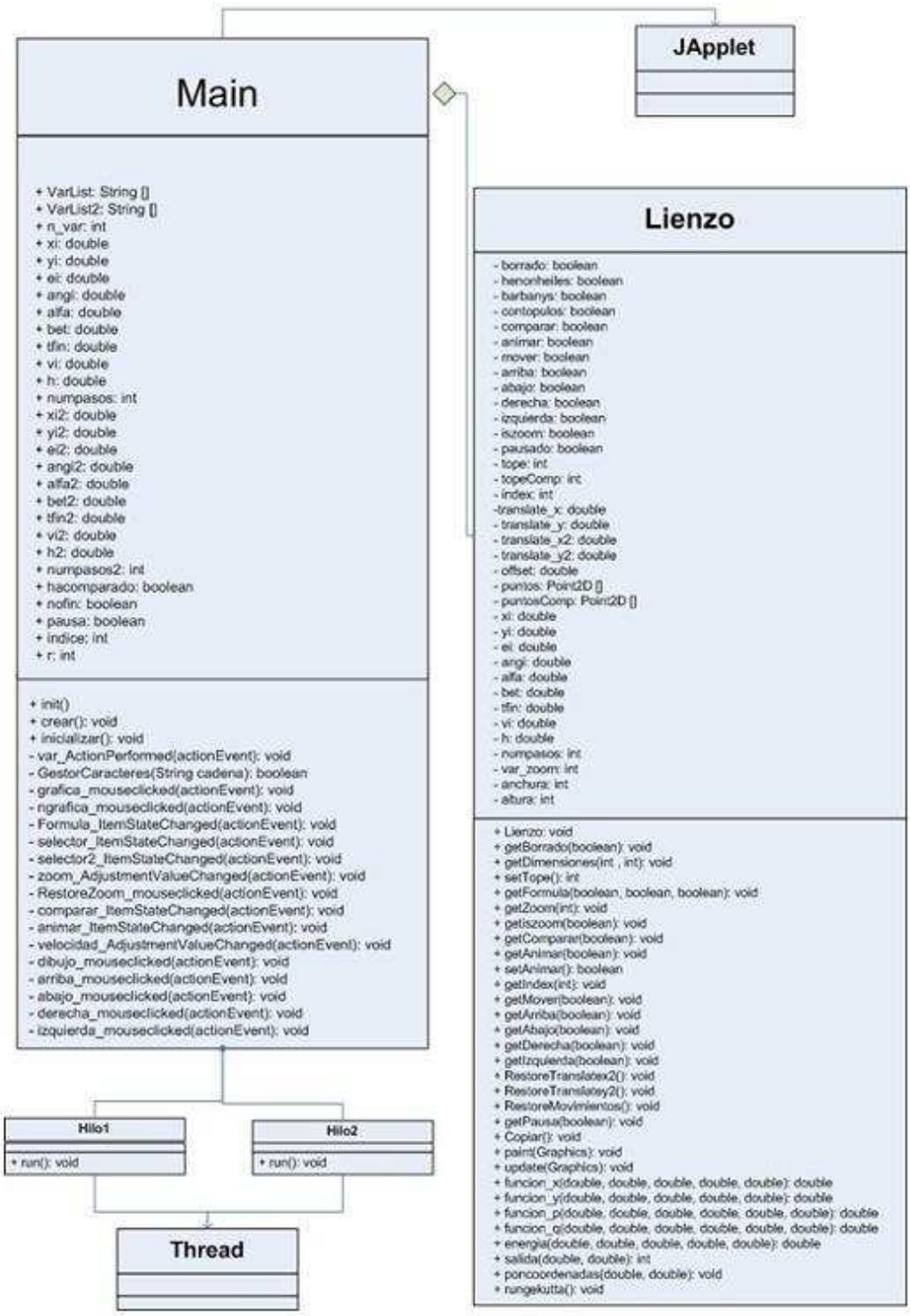

Figura 17. Diagrama estático de clases

3.3 Implementación

En este apartado se va a describir el funcionamiento de los métodos más relevantes de cada clase así como detallar la codificación de éstos. No se van a poner todos los métodos ya que sería extenderse demasiado en descripciones de ciertos métodos muy sencillos. Por ello únicamente se van a ver los métodos más importantes y que tienen más peso en el applet. También se detallarán algunas elecciones en cuanto a tipos de datos o estructuras de almacenamiento se refiere.

Clase Main

La cabecera de esta clase y del applet, ya que es la clase principal y que se ejecutará en primer lugar, es la siguiente:

```
public class Main extends javax.swing.JApplet implements Runnable
```

La clase 'Main' tiene como clase padre la clase predefinida por Java llamada JApplet ya que se va a implementar un applet. Para indicar que se va a heredar de la clase padre JApplet ha que utilizar la cláusula *extends*. También el applet va a trabajar con varios *threads* o hilos, que más tarde explicaré. Para ello Java aporta la interfaz predefinida *Runnable*. En este caso no se quiere heredar de ésta sino que se quiere implementar el método abstacto 'Run()' que en ella se especifica ya que al ser una interfaz contiene únicamente las cabeceras de los métodos sin implementarlos. Para poder hacer esto usamos la cláusula *implements*.

El cuerpo del programa se inicia con la parte de declaración de variables y acto seguido con la primera acción que hace cualquier applet al ejecutarse y que es inherente a su naturaleza que es el método 'init()'.

```
public void init() {
        try {
              javax.swing.SwingUtilities.invokeAndWait(new Runnable() {
                public void run() {
                   if(Main.WIDTH<200 || Main.HEIGHT<200)
                       resize(700,700);
                   crear();
                }
              });
         } catch (Exception e) {
                System.err.println("createGUI didn't successfully complete");
           }
}
```

En el método se especifica el tamaño que va tener la ventana del applet mediante la función predefinida *resize* y, lo más importante se llamará a la función 'crear()' que es el método gestor de todo el applet dividido en 3 partes :

1. La parte 1 donde se inicializan todos los componentes (botones, etiquetas, ventanas, paneles…) que van a formar el applet y se especifican sus características de inicio. En la siguiente captura de código se muestran algunas de las componentes que se inicializan en este método:

```
grafica=new JButton("GRAFICA");
grafica.setEnabled(false);
ngrafica=new JButton("NUEVA GRAFICA");
ngrafica.setEnabled(false);

HenonHeiles=new JRadioButton("Henon-Heiles");
Barbanys=new JRadioButton("Barbanis");
Contopoulos=new JRadioButton("Contopoulos");
Formula=new ButtonGroup();
Formula.add(HenonHeiles);
Formula.add(Barbanys);
Formula.add(Contopoulos);

zoom=new JScrollBar();
zoom.setOrientation(JScrollBar.HORIZONTAL);
zoom.setMaximum(300);
zoom.setMinimum(25);
zoom.setUnitIncrement(5);
zoom.setBlockIncrement(25);
zoom.setValue(25);
MiCanvas.getZoom(25);
zoom.setEnabled(false);

RestoreZoom=new JButton("ZOOM");
RestoreZoom.setEnabled(false);

compare=new JCheckBox("Comparar");
compare.setEnabled(false);

anime=new JCheckBox("animación");
anime.setEnabled(true);
```

2. En la segunda parte se invocará al método 'inicializar()' donde añadimos a cada componente un método de respuesta ante una acción determinada del usuario asegurando así la interacción usuario-applet. Más tarde explicaré con detalle los métodos más relevantes de 'inicializar()'.

3. Por último con los componentes ya creados y definidos, sólo falta distribuirlos por el applet. De eso se encarga el código de la tercera parte del método crear.

   Aquí se puede entender porqué no se ha dividido el applet en demasiadas ventanas o paneles a riesgo de resultar más complicado para el programador. La causa más importante es aumentar el dinamismo de las componentes del applet. Si el applet se organiza en su totalidad en paneles o ventanas, disminuyen las posibilidades de diferentes ubicaciones de éstas dentro del applet. Por ello en el applet que nos ocupa al tener únicamente 3 ventanas, las demás componentes se pueden colocar casi donde se quiera manualmente mediante el método *setBounds* proporcionado por Java al que se le pasa directamente las coordenadas del applet donde se quiere que vaya ubicada cierta componente. A continuación se puede observar el código para la ubicación dentro del applet de algunas de las componentes:

```
getContentPane().add(HenonHeiles);
HenonHeiles.setBounds(383,400,125,30);
getContentPane().add(Barbanys);
Barbanys.setBounds(383,430,125,30);
getContentPane().add(Contopoulos);
Contopoulos.setBounds(383,460,125,30);

getContentPane().add(velocidad);
velocidad.setBounds(510,440,125,10);

getContentPane().add(compare);
compare.setBounds(510,460,125,30);

getContentPane().add(anime);
anime.setBounds(510,400,125,30);

getContentPane().add(zoom);
zoom.setBounds(105,383,270,10);

getContentPane().add(RestoreZoom);
RestoreZoom.setBounds(5,378,100,20);

getContentPane().add(arriba);
arriba.setBounds(142,413,100,20);

getContentPane().add(mover);
mover.setBounds(142,433,100,20);

getContentPane().add(abajo);
abajo.setBounds(142,453,100,20);
```

Cuando se presiona el botón ‘Gráfica’ para que dibuje la gráfica el código que se ejecuta es el siguiente:

```
private void grafica_mouseclicked(java.awt.event.MouseEvent evt){

    MiCanvas.getDimensiones(340,340);
    if (HenonHeiles.isSelected())
        dibujo.setTitle("Hènon-Heiles");
    else if (Barbanys.isSelected())
        dibujo.setTitle("Barbanis");
    else
        dibujo.setTitle("Contopoulos");
    MiCanvas.getBorrado(false);
    MiCanvas.rungekutta();
    ngrafica.setEnabled(true);
    grafica.setEnabled(false);
    HenonHeiles.setEnabled(false);
    Barbanys.setEnabled(false);
    Contopoulos.setEnabled(false);
    zoom.setEnabled(true);
    RestoreZoom.setEnabled(true);
    compare.setEnabled(true);
    hacomparado=compare.isSelected();
    arriba.setEnabled(true);
    abajo.setEnabled(true);
    derecha.setEnabled(true);
    izquierda.setEnabled(true);
    hilo=new Thread(this);
    hilo.start();

}
```

Lo primero que hace el método de respuesta cuando el usuario presione el botón ‘Gráfica’ es calcular ‘Rungekutta’ con los valores que el usuario ha introducido por el campo de texto editable de ‘Datos de Entrada’ anteriormente. A continuación se van activando (*nombreComponente.setEnabled(true)*) los componentes del applet que queremos que el usuario pueda utilizar una vez dibujada la gráfica como por ejemplo el botón ‘ngrafica’ para reiniciar el applet o la barra desplazadora del zoom para que el usuario pueda aumentar o disminuir el tamaño de su gráfica; y desactivando (*nombreComponente.setEnabled(false)*) los que el usuario no debe tocar como es el caso de los botones dedicados a elegir el sistema caótico (Barbanis, Hénon-Heiles o Contopoulos) que no corresponde al usuario cambiarlo en ese instante. Estas decisiones del programador dan como resultado una más fácil comprensión de lo que debe o puede hacer el usuario en cada instante ya que de esta manera el usuario no se distrae con botones que realmente no van a tener ninguna

función en un momento concreto de la ejecución del applet como en este caso pueden ser los botones dedicados a elegir el sistema caótico antes mencionados.

En la parte inferior del método se van a utilizar *threads* o hilos de ejecución. Son segmentos de código que se ejecutan secuencialmente de modo independiente de las otras partes del programa. La Máquina Virtual Java (JVM) es un sistema *multi-thread*, es decir, que es capaz de ejecutar varias secuencias de ejecución simultáneamente. En este applet se va a necesitar varios *threads* de ejecución que nos darán la posibilidad de dibujar las gráficas de manera animada, es decir que el usuario pueda regula la velocidad de pintado para poder observar su trayectoria además de tener la posibilidad de pausar o reanudar la ejecución a su libre decisión.
Una vez explicado que es un *thread* y porqué es necesario la tecnología multihilo, se puede entender lo que en la parte inferior del método se describe. Se crea un nuevo hilo de ejecución y se ejecuta con la claúsula *hilo.start()* que llama al método 'run' que inicia la ejecución de ese nuevo hilo como podemos ver a continuación:

```
public void run() {
    nofin=true;
    indice=1;
    while(nofin){
        if(anime.isSelected()){
            MiCanvas.getIndex(indice);
            MiCanvas.repaint();
            nofin=indice<MiCanvas.setTope();
            try {
                hilo.sleep(r);
            } catch (InterruptedException ex) {
                ex.printStackTrace();
            }
        }
        else{
            MiCanvas.repaint();
            nofin=false;
        }
        indice++;
    }
    nofin=false;
    hilo.stop();
}
```

En este método el hilo entra en un bucle donde se pinta la gráfica de una sóla vez si el usuario no ha elegido ejecutarlo de manera animada o punto a punto si por el contrario el usuario antes de hacer clic en 'Grafica' había seleccionado 'Animacion'. Para el primer caso dentro del bucle se llama al método 'udpate()' del objeto 'MiCanvas' de la clase Lienzo destinado para pintar, mediante 'MiCanvas.repaint()'

y pintará la gráfica en su totalidad ya que así lo ha elegido el usuario. Una vez pintada, la variable *boolean* 'NoFin' se pondrá a *false*, indicando que ya se ha llegado al final del proceso de pintado, y finalizando así el bucle.
Para el segundo caso la gráfica se irá pintando punto a punto. Esto significa que cada vez se pintará una gráfica con un punto más, primero una gráfica que conste de un punto, luego de dos, luego de tres y así sucesivamente hasta que pinte todos los puntos de la gráfica. Una vez finalizado el bucle se matará el hilo mediante la orden predefinida por Java 'stop()'.

También es muy importante el botón 'NuevaGrafica' que sirve como se ha mencionado antes para reiniciar el applet y poder pintar otras gráficas. El código del método de respuesta al clicar dicho botón es el siguiente:

```
private void ngrafica_mouseclicked(java.awt.event.MouseEvent evt) {

    dibujo.setTitle("Función");
     if (!compare.isSelected()){
         MiCanvas.getBorrado(true);
         MiCanvas.repaint();
     }
     if(hacomparado){
         compare.setSelected(false);
         MiCanvas.getBorrado(true);
         MiCanvas.repaint();
     }
     etiqueta_variable.setText("Introduzca el valor de la x inicial");
     n_var=1;
     var.setEditable(true);
     ngrafica.setEnabled(false);
     HenonHeiles.setEnabled(true);
     Barbanys.setEnabled(true);
     Contopoulos.setEnabled(true);
     zoom.setEnabled(false);
     RestoreZoom.setEnabled(false);
     compare.setEnabled(false);
     dato.setText("");
     dato2.setText("");
     MiCanvas.RestoreMovimientos();
     MiCanvas.getMover(false);
     MiCanvas.RestoreTranslatex2();
     MiCanvas.RestoreTranslatey2();
     MiCanvas.restoreContEjes();
     arriba.setEnabled(false);
     abajo.setEnabled(false);
     derecha.setEnabled(false);
     izquierda.setEnabled(false);
     anime.setSelected(false);
     MiCanvas.getiszoom(false);
     hilo.stop();

}
```

Lo primero que hace el método es limpiar la superficie del lienzo donde se dibuja siempre que no está seleccionada la opción 'Comparar' ya que el usuario quiere que la gráfica quede pintada para así compararla con la próxima gráfica que dibuje. También se limpia el lienzo si lo que está pintado en ese momento son dos gráficas, ya que no se puede comparar más de dos gráficas. Para poder limpiar el lienzo se utiliza un método del objeto 'MiCanvas' de la clase 'Lienzo' al que le pasaré un *true* si quiero borrar o un *false* en caso contrario. Este método se aplica al procedimiento para pintar de la misma clase que explicaré más adelante.

El resto del método al igual que ocurría con el método del botón 'Grafica', se dedica a activar o desactivar las componentes del applet según sean las exigencias que el usuario va a tener. En este caso el usuario va a volver casi por completo a una situación similar al inicio del applet, por lo tanto las componentes deben adecuarse a ello y estar activadas/desactivadas de manera similar a cuando se inició el applet. Por ejemplo la parte de introducción de valores para las variables debe estar activada ya que se va pintar una nueva gráfica con esos nuevos valores como es el caso de los botones dedicados a elegir el sistema galáctico (Barbanis, Contopoulos o Hénon-Heiles) ya que el usuario debe elegir el sistema que desea. Por otro lado deberán estar desactivados por ejemplo la barra despalzadora de zoom ya que puede no haber nada sobre lo que hacer zoom.

Como ya se ha dicho anteriormente, los valores de las variables los debe introducir el usuario por medio de un campo de texto editable situado en la ventana 'Datos de Entrada'. Cuando se crea y se inicia el applet la etiqueta adjunta a ese campo de texto debe ser la de la primera variable a introducir que es la x inicial, por lo que se debe inicializar así la etiqueta cuando se construye en el método 'crear()'.
Cuando el usuario hace clic sobre el campo de texto el método de respuesta que se ejecuta es el siguiente:

```
private void var_ActionPerformed(java.awt.event.ActionEvent evt){

    if(n_var==1){
       String x=var.getText();
       if(!GestorCaracteres(x)){
           xi=Double.parseDouble(x);
           etiqueta_variable.setText("Introduzca el valor de la y inicial");
       }
       else{
          mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
          n_var--;
       }
       var.setText("");
    }
```

```
else if (n_var==2){
    String y=var.getText();
    if(!GestorCaracteres(y)){
       yi=Double.parseDouble(y);
       etiqueta_variable.setText("Introduzca el valor de la energia inicial");
    }
    else{
       mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
       n_var--;
    }
    var.setText("");
}
else if (n_var==3){
    String energia=var.getText();
    if(!GestorCaracteres(energia)){
        ei=Double.parseDouble(energia);
        etiqueta_variable.setText("Introduzca th inicial                    ");
    }
    else{
       mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
       n_var--;
    }
    var.setText("");
}
else if (n_var==4){
    String beta=var.getText();
    if(!GestorCaracteres(beta)){
        bet=Double.parseDouble(beta);
        etiqueta_variable.setText("Introduzca t final                    ");
    }
    else{
       mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
       n_var--;
    }
    var.setText("");
}
 else if (n_var==5){
     String tf=var.getText();
     if(!GestorCaracteres(tf)){
         tfin=Double.parseDouble(tf);
         etiqueta_variable.setText("Introduzca el numero de pasos");
     }
     else{
        mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
        n_var--;
     }
     var.setText("");
 }
 else {
     String npasos=var.getText();
     if(!GestorCaracteres(npasos)){
         numpasos=Integer.parseInt(npasos);
        vi=Math.sqrt(2.0*ei-xi*xi-yi*yi-2.0*xi*xi*yi+(2.0/3.0)*yi*yi*yi);
        h=tfin/numpasos;
        var.setText("");
        var.setEditable(false);
        grafica.setEnabled(true);
        MiCanvas.getValoresIniciales(xi,yi,ei,angi,alfa,bet,tfin,vi,h,numpasos);
     }
```

```
            else{
               mensaje.showMessageDialog(this,"Carácter incorrecto"," Error: ",JOptionPane.ERROR_MESSAGE);
               n_var--;
               var.setText("");
            }
         }
         n_var++;
      }
```

Primero lee el valor introducido por el usuario para la variable pedida y luego se modifica el texto de la etiqueta adjunta pidiendo el valor de la siguiente variable. Este proceso hay que realizarlo ocho veces que son las variables que el usuario debe introducir además de velocidad inicial y el coeficiente h que se calculan a partir de las variables antes mencionadas. Cada vez que se repite este proceso y lee una variable introducida por el usuario, debemos saber si lo introducido es válido según el formato exigido. Para ello la variable se somete a un filtro en forma de método llamado 'GestorCaracteres' y que determina si es correcto o no el formato. Este formato impide que una variable no sea numérica o que comience con coma y demás requisitos. Si la variable supera el filtro, el proceso pedirá el valor de la siguiente variable como ya hemos visto al principio; en caso contrario aparecerá un mensaje de aviso pidiendo un correcto formato y volverá a pedir el valor de la misma variable. Esto se consigue mediante el método predefinido por Java *showMessageDialo*g definido para la componente de la clase *OptionPane* llamada mensaje. Cabe decir que mientras el usuario no introduzca bien el valor no podrá avanzar en el applet evidentemente.

Clase Lienzo

Los métodos de dicha clase se pueden dividir en 2 grupos:

- Métodos destinados a la representación gráfica: Al tratarse de la clase encargada de ésta, contiene el método 'Paint()' ya que es un método de la clase padre 'Canvas' predefinida por Java y el método 'Update()' donde realmente se desarrollan todas las sentencias necesarias para dibujar la gráfica final. Dentro de este grupo se encuentran los pequeños métodos privados que interactúan con la clase 'Main'.
- Métodos destinados al cálculo de valores: Para poder hacer la representación gráfica primero hay que calcular los valores de dicha gráfica en el espacio según los parámetros de entrada introducidos por el usuario en la clase 'Main'. El método más importantes es Rungekutta que es donde realmente se calcularán todo los valores necesarios para que en 'update()' se dibuje la gráfica.

Cabe destacar el papel de los atributos en esta clase ya que el valor de éstos va ir cambiando continuamente dependiendo de las acciones del usuario.
Cuando el usuario pincha el botón 'Grafica' o hace clic sobre 'Animacion' o hace cualquier cosa en el applet, éste responde con un método asociado a la componente sobre la que haya actuado el usuario. Estos cambios se trasladan a la clase Lienzo mediante métodos públicos de tipo *Get* para modificar el valor del atributo de dicha clase y estar siempre actualizado. Estos métodos son necesarios ya que los atributos son privados siempre. Con ello se pueden asegurar tres de los pilares fundamentales del Lenguaje Orientado a Objetos: encapsulamiento, abstracción y principio de ocultación dando a conocer a otros objetos únicamente lo estrictamente necesario.
Por ejemplo si el usuario hace clic sobre 'Animacion', el método de respuesta de la componente será modificar el valor del atributo animar en la clase 'Lienzo' mediante el método 'GetAnimar' al que se le pasará *true* o *false* dependiendo del estado anterior. De este modo cuando va a pintar la gráfica mediante 'update()' si el atributo animar es *true* dibujará con animación y viceversa.

La estructura donde se van a ir guardando todos los valores 'x' e 'y' obtenidos al aplicar 'Rungekutta', se llama puntos y se trata de un tipo *Point2D*. Esta estructura predefinida por Java es una matriz cuyas dimensiones son la cantidad de puntos que tengamos como longitud, por una anchura de dos, 'x' e 'y' ya que cualquier punto en un plano está compuesto de esas dos coordenadas.
Al tratarse de una estructura estática, es decir la longitud del vector no se va modificando si no que está predefinida por el programador inicialmente, necesitamos el atributo tope que como su nombre indica sirve para saber hasta dónde hemos rellenado el *Point2D* puntos. Este atributo es sumamente importante y únicamente se podrá modificar en 'Rungekutta'. Por ello este atributo no tiene método *get* asociado. Sí tiene un método set para poder proporcionar el valor de tope a la clase 'Main' en un momento determinado.

Los métodos más importantes en esta clase son 'Rungekutta' y 'Update()'. El código del bucle *for* de 'Rungekutta' donde se calculan los valores de todos los puntos es el siguiente:

```
for (paso=1;paso<=numpasos;paso++){

    double k1=h*funcion_x(t,x,y,p,q);
    double l1=h*funcion_y(t,x,y,p,q);
    double m1=h*funcion_p(t,x,y,p,q,alfa);
    double n1=h*funcion_q(t,x,y,p,q,bet);

    double k2=h*funcion_x(t+(h/2.0),x+(k1/2.0),y+(l1/2.0),p+(m1/2.0),q+(n1/2.0));
    double l2=h*funcion_y(t+(h/2.0),x+(k1/2.0),y+(l1/2.0),p+(m1/2.0),q+(n1/2.0));
    double m2=h*funcion_p(t+(h/2.0),x+(k1/2.0),y+(l1/2.0),p+(m1/2.0),q+(n1/2.0),alfa);
    double n2=h*funcion_q(t+(h/2.0),x+(k1/2.0),y+(l1/2.0),p+(m1/2.0),q+(n1/2.0),bet);

    double k3=h*funcion_x(t+(h/2.0),x+(k2/2.0),y+(l2/2.0),p+(m2/2.0),q+(n2/2.0));
    double l3=h*funcion_y(t+(h/2.0),x+(k2/2.0),y+(l2/2.0),p+(m2/2.0),q+(n2/2.0));
    double m3=h*funcion_p(t+(h/2.0),x+(k2/2.0),y+(l2/2.0),p+(m2/2.0),q+(n2/2.0),alfa);
    double n3=h*funcion_q(t+(h/2.0),x+(k2/2.0),y+(l2/2.0),p+(m2/2.0),q+(n2/2.0),bet);

    double k4=h*funcion_x(t+h,x+k3,y+l3,p+m3,q+n3);
    double l4=h*funcion_y(t+h,x+k3,y+l3,p+m3,q+n3);
    double m4=h*funcion_p(t+h,x+k3,y+l3,p+m3,q+n3,alfa);
    double n4=h*funcion_q(t+h,x+k3,y+l3,p+m3,q+n3,bet);

    t=t+h;
    x=x+(k1+2.0*k2+2.0*k3+k4)/6.0;
    y=y+(l1+2.0*l2+2.0*l3+l4)/6.0;
    p=p+(m1+2.0*m2+2.0*m3+m4)/6.0;
    q=q+(n1+2.0*n2+2.0*n3+n4)/6.0;
    ener=energia(t,x,y,p,q);

    poncoordenadas(x,y);

    if (x*x+y*y>100) break;

}
```

En cada iteración del bucle se calculan las coordenadas ‘x’ e ‘y’ de cada punto que conforma la gráfica resultante. Esto depende del valor que contenga la variable ‘NumPasos’ introducido por el usuario en ‘Datos de Entrada’. Si ‘NumPasos’ es igual a 10.000 entonces la gráfica estará compuesta de 10.000 puntos que deberán ser calculados en 10.000 iteraciones de este bucle, punto a punto.
Por lo tanto en cada iteración del bucle, se obtiene un punto de la gráfica que se añade al vector puntos mediante el método ‘poncoordenadas’ incrementando en 1 el valor de tope ya que el vector contiene un punto más.

```
private void poncoordenadas (double x,double y){

    if (tope==0){
        puntos[tope]=new Point2D.Double (x, -y);
    }
    else{
        puntos[tope]=new Point2D.Double (x, -y);
    }
    tope++;

}
```

Hay que tener en cuenta que en la clase 'Canvas' predefinida de Java el origen de coordenadas es la parte superior derecha del lienzo sobre el que se va dibujar. Además, a pesar que la 'x' se incrementa de manera normal hacia la derecha, la 'y' se incrementa hacia abajo en contra de lo que habitualmente estamos acostumbrados a ver. Por ello para que un punto quede representado de acuerdo a un sistemas de coordenadas habitual con el origen de coordenadas en el centro del lienzo, debemos aplicar un desplazamiento a cada coordenada 'x' e 'y' e invertir el sentido de la 'y' para que se incremente hacia arriba y no al revés.
Esto último se consigue directamente al introducir el punto en el *Point2D* poniendo un signo '–' delante de la 'y' a pesar que el desplazamiento se va a realizar directamente en el 'update()'. El desplazamiento ha sido fijado de acuerdo con el origen de coordenadas. Si este origen para Java está ubicado en la parte superior derecha del lienzo y habitualmente se encuentra en el centro del lienzo, basta con incrementar cada 'x' en la mitad de la anchura del lienzo y cada 'y' en la mitad de la altura de éste. Así se consigue una representación acorde con el sistema habitual.

Una vez se tiene el vector lleno hasta el tope con nuestros puntos, sólo queda representarlos gráficamente en el 'Lienzo'. La gráfica estará formada por un conjunto de rectas. Cada recta tiene un punto origen y un punto final. De este modo se irán dibujando rectas desde el punto 'j-1' del vector puntos hasta el punto 'j' mediante el bucle siguiente:

```
for (int j = 1; j<=tope; j++)
 {
     if (j==1){
          g.drawLine (
               (int)(puntos[j-1].getX()*var_zoom+translate_x),
               (int)(puntos[j-1].getY()*var_zoom+translate_y),
               (int)(puntos[j].getX()*var_zoom+translate_x),
               (int)(puntos[j].getY()*var_zoom+translate_y));
     }
     else{
          g.drawLine(
               (int)(puntos[j-1].getX()*var_zoom+translate_x),
               (int)(puntos[j-1].getY()*var_zoom+translate_y),
               (int)(puntos[j].getX()*var_zoom+translate_x),
               (int)(puntos[j].getY()*var_zoom+translate_y));
     }

 }
```

Para pintar cada recta entre ‘j-1’ y ‘j’ en cada iteración del bucle, se utiliza el método *drawLine(x0,y0,x1,y1)* de la clase *Graphics* predefinida por Java. A cada punto se le aplica el zoom que en ese momento el usuario haya elegido (por defecto el zoom será 25) y el desplazamiento anteriormente explicado.

El método ‘update()’ está dividido en varias partes según la forma de pintar que el usuario haya elegido en el applet:

- Si el usuario ha elegido dibujar una nueva gráfica pulsando el botón ‘NuevaGrafica’ y no ha seleccionado la opción comparar previamente, la gráfica pintada tiene que ser borrada para poder pintar de nuevo. Esto se consigue poniendo el atributo borrado a *true* mediante el método ‘GetBorrado’ de la clase ‘Lienzo’ y seguidamente pintar mediante *repaint()* que es la forma de llamar a ‘update()’.
  En ‘update()’ entrará por esta opción ya que borrado está a *true* y pintará un rectángulo blanco encima de la gráfica que hubiera dibujada anteriormente consiguiendo así una superficie limpia para el ‘Lienzo’.

```
if (borrado){
    g.setColor(Color.WHITE);
    g.fillRect(0,0,anchura,altura);
}
```

- Si el usuario ha elegido pintar la gráfica en la forma por defecto , el código que se ejecutará previo al bucle anteriormente descrito ,es el siguiente:

```
if (comparar)
    g.setColor(Color.BLUE);
else{
   g.setColor(Color.WHITE);
   g.fillRect(0,0,anchura,altura);
   g.setColor(Color.BLACK);
   if (var_zoom<=170){
     CalcularEjes();
     int i=minimoDibujo;
     int min=minimo;
     for (int j=0;j<=resta;j++){

       g.drawLine(i,165,i,175);
       if (min!=0){
         g.drawString(""+min,i-5,190);
         g.drawString(""+(-min),180,i+5);
       }
       g.drawLine(165,i,175,i);
       i=i+divDibujo;
       min++;
     }
   }
```

```
            g.drawLine(0,170,340,170);
            g.drawLine(170,0,170,340);
            Font f=new Font("Helvetica",Font.ITALIC,15);
            g.setFont(f);
            g.drawString("X",310,160);
            g.drawString("Y",190,15);
            g.setColor(Color.RED);
        }
        for (int j = 1; j<=tope; j++)
         {
            if (j==1){
                g.drawLine (
                    (int)(puntos[j-1].getX()*var_zoom+translate_x),
                    (int)(puntos[j-1].getY()*var_zoom+translate_y),
                    (int)(puntos[j].getX()*var_zoom+translate_x),
                    (int)(puntos[j].getY()*var_zoom+translate_y));
            }
            else{
                g.drawLine(
                    (int)(puntos[j-1].getX()*var_zoom+translate_x),
                    (int)(puntos[j-1].getY()*var_zoom+translate_y),
                    (int)(puntos[j].getX()*var_zoom+translate_x),
                    (int)(puntos[j].getY()*var_zoom+translate_y));
            }

        }
    }
```

Si el lienzo tiene una gráfica dibujada porque el usuario ha elegido compararla con la que se va pintar, el color del trazado de dicha gráfica debe ser diferente (azul) a la ya dibujada para poder diferenciarlas. Si el usuario no eligió comparar dos gráficas, se limpia el lienzo y se pinta en el color rojo que es el habitual. Además se dibuja los ejes de la gráfica con sus correspondientes coordenadas. Las coordenadas dependerán del grado de zoom que haya. Para conseguir que los ejes contengan unas coordenadas acordes al gráfico y se amolden al zoom se utiliza el método CalcularEjes() que se describe a continuación:

```
private void CalcularEjes(){

    double min,max;
    min=-170.0/var_zoom;
    max=170.0/var_zoom;
    minimo=(int)Math.ceil(min);
    maximo=(int)Math.floor(max);
    int minDibujo,maxDibujo;
    minDibujo=(int)((170.0*minimo)/min);
    maxDibujo=(int)((170.0*maximo)/max);
    minimoDibujo=170-minDibujo;
    maximoDibujo=170+maxDibujo;
    restaDibujo=maximoDibujo-minimoDibujo;
    resta=maximo-minimo;
    divDibujo=(int)(Math.round(restaDibujo/resta));
}
```

- Si el usuario ha elegido dibujar con animación el bucle donde se van dibujando las rectas que conforman será diferente al habitual.

```
for (int j = 1; j<=index; j++)
{
```

La clase 'Lienzo' debe saber hasta que punto del *Point2D* debe pintar cada vez hasta llegar que el atributo índice sea igual al atributo 'tope' del vector 'puntos'. Esto es posible gracias al método 'GetIndice' que modifica el atributo índice de la clase 'Lienzo'. Dicho método se va ir invocando en el 'run' incrementando en uno el índice por cada iteración del bucle del 'run'. Así cada vez se pintará una gráfica con una recta más.

- Si el usuario aplica zoom a la gráfica, en el 'update()' se pinta una nueva gráfica con 'varZoom' modificada ya sea animada (por la rama del *if* anterior) ,por defecto o con dos gráficas previa comparación.
  Cuando se da esta última casuística, se repintan las dos gráficas cada una con un 'tope' diferente y unos puntos diferentes evidentemente. Por ello, cuando el usuario elige comparar la gráfica que acaba de dibujarse con la siguiente que va dibujar, en la clase 'Main' se invoca a un método de 'Lienzo' llamado 'copiar()' que como su nombre indica copia el vector 'puntos' de la gráfica recién dibujada en un vector auxiliar llamado 'puntosComp'. De esta manera cuando el usuario pinte la segunda gráfica para comparar, y quiere hacer zoom al lienzo con ambas gráficas, el método 'update()' deberá dibujar ambas gráficas según los vectores de ambas. La segunda gráfica estará contenida en el vector 'puntos' y se dibujará mediante el bucle por defecto visto al principio mientras que la primera gráfica cuyos puntos están almacenados en el vector 'puntosComp', se dibujará de acuerdo con este bucle:

```
for (int j = 1; j<=topeComp; j++)
{
    if (j==1){
        g.drawLine (
                (int)(puntosComp[j-1].getX()*var_zoom+translate_x),
                (int)(puntosComp[j-1].getY()*var_zoom+translate_y),
                (int)(puntosComp[j].getX()*var_zoom+translate_x),
                (int)(puntosComp[j].getY()*var_zoom+translate_y));
    }
    else{
        g.drawLine(
                (int)(puntosComp[j-1].getX()*var_zoom+translate_x),
                (int)(puntosComp[j-1].getY()*var_zoom+translate_y),
                (int)(puntosComp[j].getX()*var_zoom+translate_x),
                (int)(puntosComp[j].getY()*var_zoom+translate_y));
    }
}
```

- El usuario puede elegir mover la gráfica arriba, abajo o a los lados. Para ello tiene los botones que rodean la etiqueta ‘Mover’ del applet llamados ‘arriba’, ‘abajo’, ‘izquierda’ y ‘derecha’.
  Cuando el usuario pulsa uno de esos botones en el método de respuesta se pondrá a *true* el atributo mover de la clase ‘Lienzo’ mediante el método ‘getMover’ y el atributo acorde con el botón que haya pulsado mediante los métodos ‘getArriba’, ‘getAbajo’, ‘getDerecha’ y ‘getIzquierda’ para que el ‘update()’ a la hora de pintar la gráfica nuevamente sepa hacia donde tiene que modificar el desplazamiento de las ‘x’ y de las ‘y’.
  A continuación detallo el fragmento de código de la rama mover del método ‘update()’ donde se modifica el desplazamiento de acuerdo con un ‘Offset’ constante de 30.0:

```
if(arriba){
  translate_y2=translate_y2-offset;
  contY--;
}
else if (abajo){
    translate_y2=translate_y2+offset;
    contY++;
}
else if (derecha){
    translate_x2=translate_x2+offset;
    contX++;
}
else if(izquierda){
    translate_x2=translate_x2-offset;
    contX--;
}
```

3.4 Movimientos Representativos del Sistema

A continuación se van a mostrar varias capturas de pantalla de la ejecución del applet en 3 situaciones diferentes:

1. Trazado de la trayectoria de una partícula con comportamiento caótico mediante Hénon-Heiles. La partícula choca con las paredes varias veces hasta que escapa por una de las salidas.

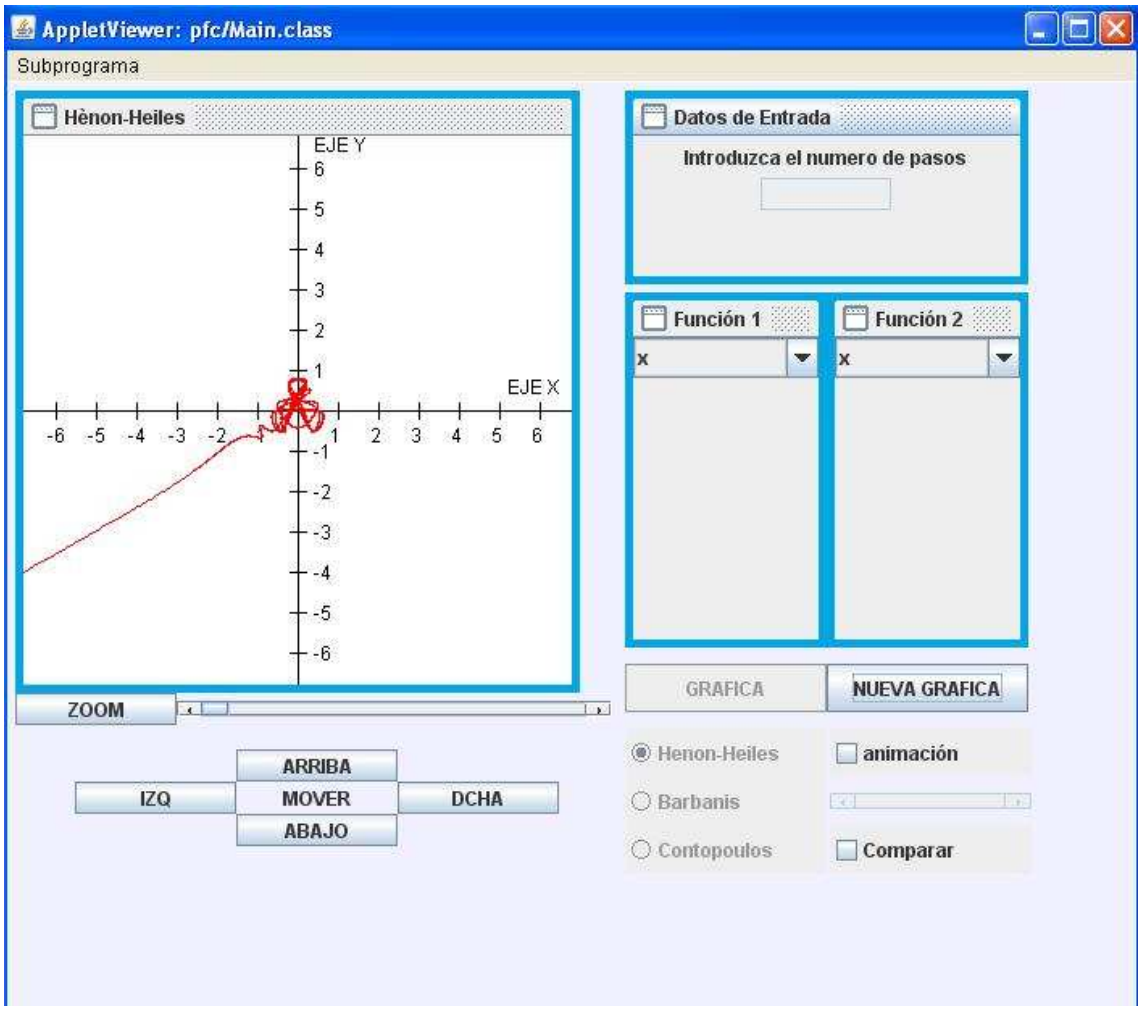

Figura 18. Representación dinámica de una trayectoria mediante Hénon-Heiles.

2. Trazado de la trayectoria de una partícula con comportamiento caótico pero esta vez mediante Barbanis.

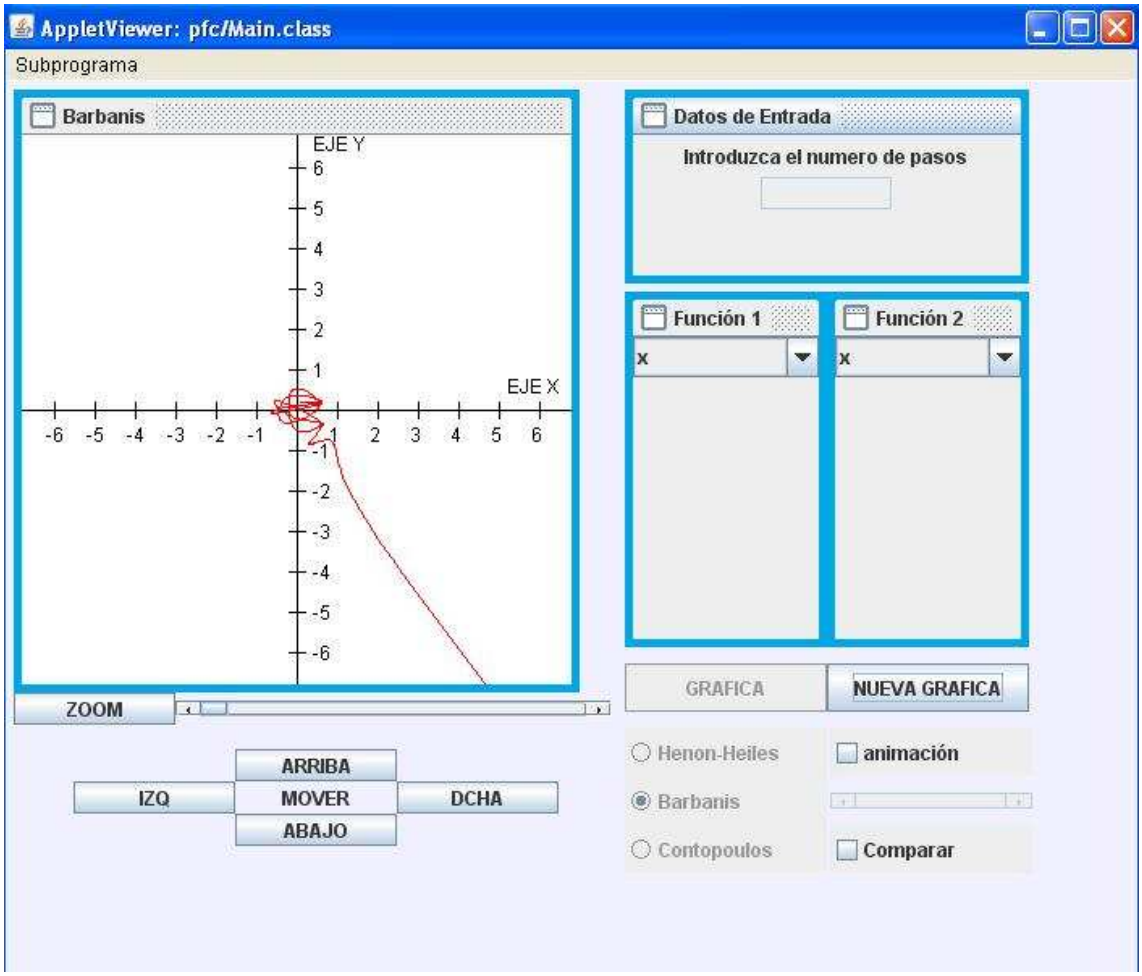

Figura 19. Representación dinámica de una trayectoria mediante Barbanis.

3. Trazado de la trayectoria de una partícula con comportamiento caótico mediante Contopoulos.

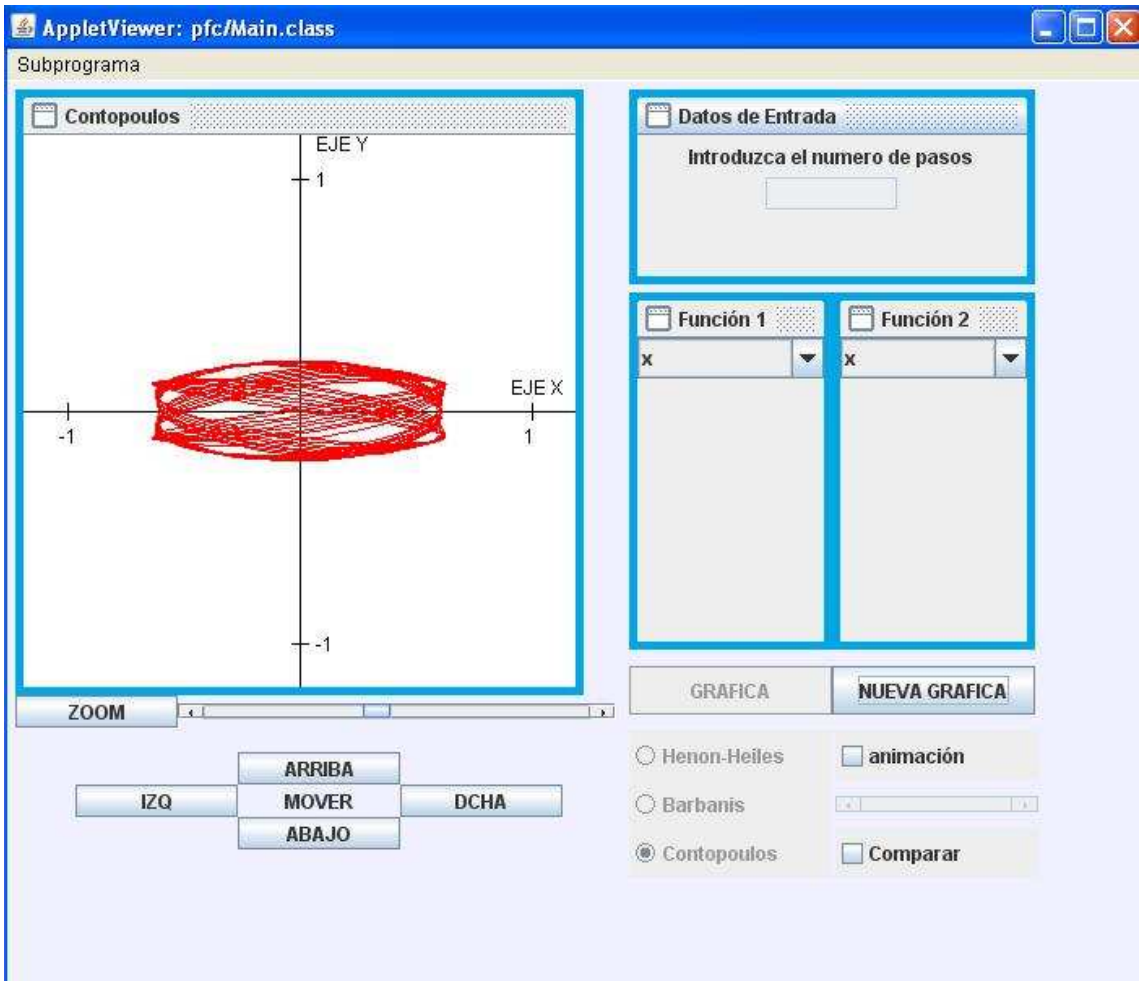

Figura 20. Representación dinámica de una trayectoria mediante Contopoulos.

4. Trazado de la trayectoria de 2 partículas con comportamiento caótico. La primera partícula con una traza roja describe la trayectoria según Barbanis y la segunda partícula con traza en color azul describe la trayectoria según Hénon-Heiles. Ambas partículas tienen los mismos parámetros de entrada y en ambos casos la partícula se queda atrapada sin salida en un movimiento caótico:

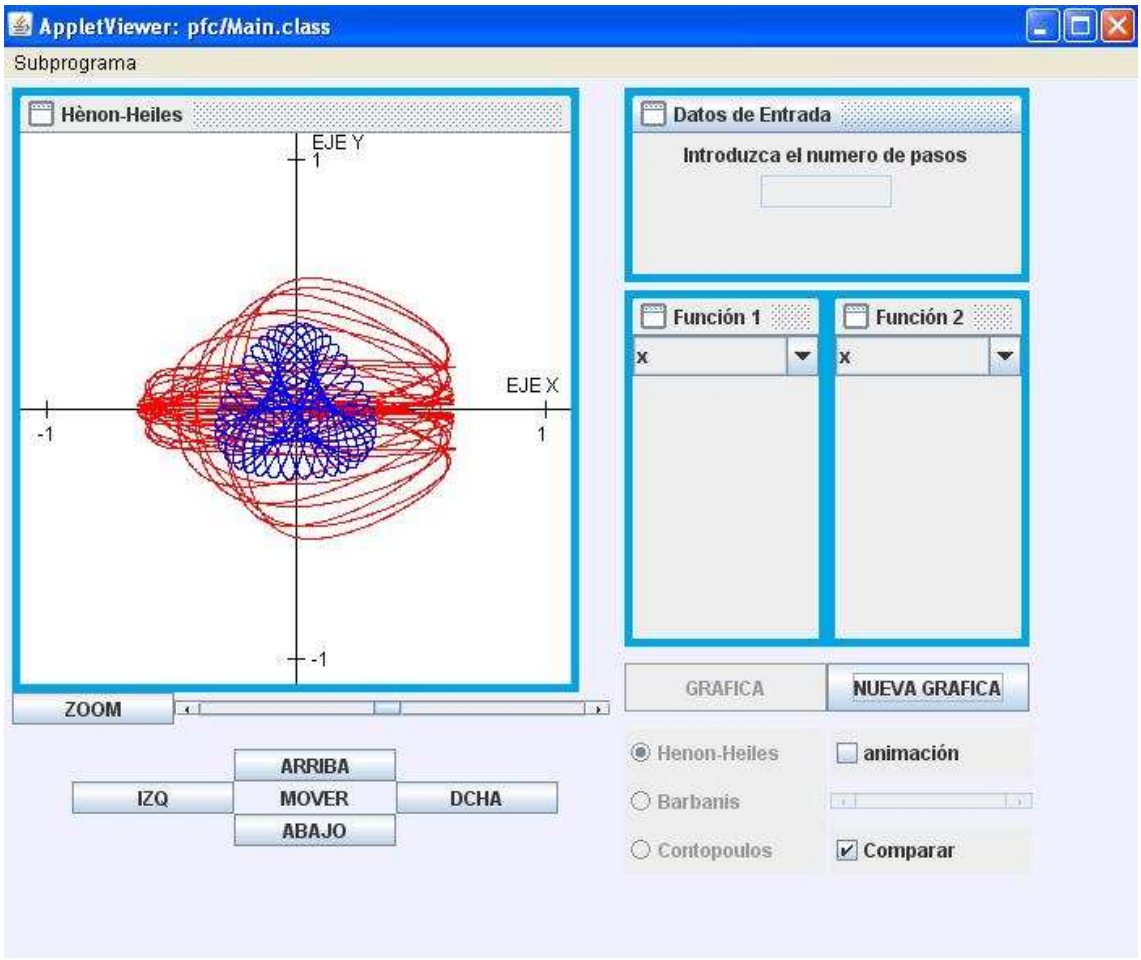

Figura 21. Representación de dos trayectorias mediante Hénon-Heiles y Barbanis.

5. Por último vamos a mostrar el applet en la situación que el usuario haya introducido mal un dato de entrada:

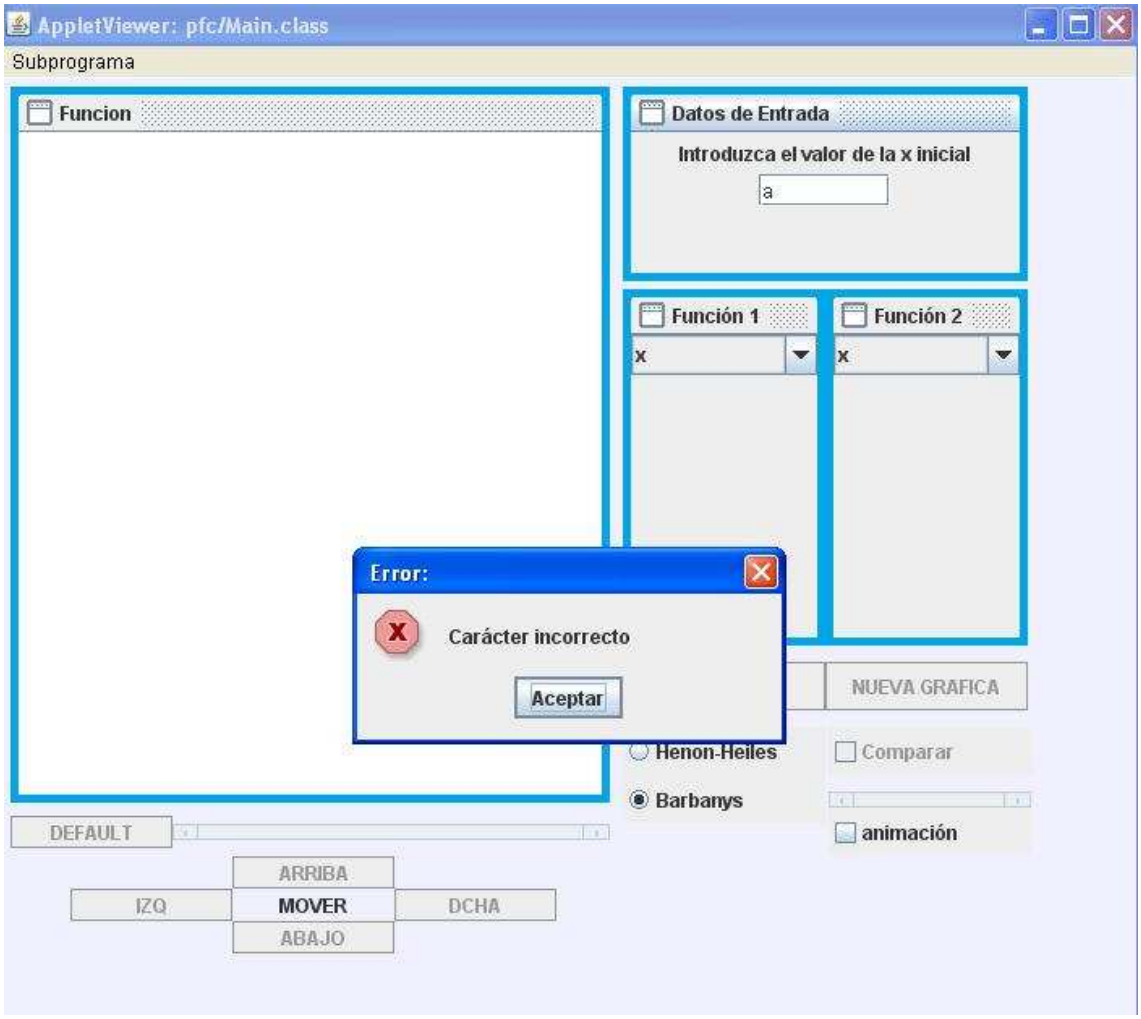

Figura 22. Introducción errónea de los datos de entrada.

4. Conclusiones

Evidentemente el logro más importante es la exitosa implementación de un applet en Java como solución para el problema de representación de tres sistemas galácticos con comportamiento caótico: Hénon-Heiles, Contopoulos y Barbanis.

También se ha conseguido diseñar el applet bastante sencillo e intuitivo cuyo uso no debe suponer ningún problema para el usuario, cumpliendo así otro de los objetivos propuestos.

Por último, el applet se ha integrado en una página web para que pueda ser utilizado por cualquiera que tenga acceso a Internet cumpliendo así el objetivo de accesibilidad propuesto inicialmente.

5. Trabajo futuro

Como cualquier aspecto relacionado con la informática y más tratándose de software, esta aplicación está abierta a mejoras y actualizaciones futuras. Por ejemplo, se pueden añadir más sistemas caóticos galácticos, además de Hénon-Heiles, Contopoulos y Barbanis, al applet en un futuro ya que la base ya de todos ellos está implementada. De esta manera se podría conseguir una aplicación con todos los sistemas caóticos galácticos.

Una propiedad inherente a los applets es que se pueden incrustar en el marco de una página web. Esto abre un gran abanico de posibilidades de uso para este applet. Una de ellas es crear una página web en un futuro donde se pueda obtener documentación de los sistemas caóticos y poder representarlos gráficamente gracias a este applet además de todos los applets que han ido realizando los demás alumnos de la Universidad.

6. Experiencia Personal

Una vez finalizado el proyecto puedo catalogar la experiencia de muy satisfactoria en términos generales.

En particular, me ha resultado especialmente interesante desde el punto de vista de mi formación académica ya que me ha supuesto realizar un importante esfuerzo cuya recompensa es, además de la consecución del applet, la obtención de nuevos conociemientos sobre Java y sobre los applets. Personalmente este aspecto resulta muy útil ya que en mi lugar de trabajo es muy habitual entre los informáticos dominar PHP, JavaScript, HTML y Hojas de Estilo, y conocer Java y los applets es un plus para mí que no todos conocen. De este modo, puedo implementar futuros applets como posibles proyectos internos en la empresa.

Fuera del ámbito profesional, cabe destacar que mis conocimientos acerca de los sistemas caóticos antes de realizar este proyecto eran casi nulos y una vez finalizado puedo tener una discusión sobre estos sistemas. Además, personalmente es un tema que que me atrae en gran medida por lo que en un futuro no cabe duda que, como entretenimiento, ya que no soy físico, seguiré consultando páginas web y libros acerca de este tema.

7. Bibliografía

[1] "Bifurcations and safe regions in open Hamilonians". R.Barrio, F.Blesa y S.Serrano. New Journal of Physics. 11,053004 (2009)

[2] "Basin topology in dissipative chaotic scattering". Jesús M. Seoane, Jacobo Aguirre y Miguel A.F. Sanjuán. Chaos. 16,023101 (2006)

[3] "Diffusion and scaling in escapes from two-degrees-of-freedom Hamiltonians systems". Henry E. Kandrup, Christos Siopis, G. Contopoulos, Rudolf Dvorak (1998). Chaos. 9,381 (1999)

[4] "Numerical Analysis". R.L.Burden, J.D. Farves. ITP Publishing (1997)

[5] "Técnicas avanzadas de diseño de software: Orientación a objetos , UML, patrones de diseño y Java. José F. Vélez Serrano, Ángel Sáchez Calle, Alfredo Casado . Bernárdez, Santiago Doblas Álvarez. Universidad Rey Juan Carlos.

[6] "Teoría del Caos". Wikipedia:
http://es.wikipedia.org/wiki/Teor%C3%ADa_del_Caos

[7] "Sistemas dinamicos y teoria del caos". Wikipedia:
http://es.wikipedia.org/wiki/Sistemas_din%C3%A1micos_y_teor%C3%ADa_del_caos

[9] Introducción a los applets en Java:
http://sunsite.dcc.uchile.cl/java/docs/JavaTut/Cap2/holamapp.html

[10] Manejo de Componentes y Eventos en un JApplet de Java:
http://java.comsci.us/examples/swing/index.html

[11] Control de la animación en Java:
http://www.sc.ehu.es/sbweb/fisica/cursoJava/applets/threads/moviendo.htm

[12] Creación y control de threads:
http://www.wikilearning.com/tutorial/tutorial_de_java/3938-141