

Documento fuertemente basado en el documento de la práctica WALL·E de la universidad Complutense de Madrid.

Autores: Álvaro Quesada Pimentel y Daniel Serrano Torres.

Acercamiento al lenguaje de programación Java y a sus herramientas de desarrollo.

En este documento se explica en que consiste la práctica que ha realizado, su alcance, sus objetivos, las soluciones dadas, etcétera…

Tecnología de la programación:  
Practica cero.

Índice

[1. Requisitos 3](#_Toc341342821)

[2. Objetivos 5](#_Toc341342822)

[3. Resumen 5](#_Toc341342823)

[4. Abstract 6](#_Toc341342824)

[5. Observaciones 6](#_Toc341342825)

[6. Sugerencias 6](#_Toc341342826)

[7. Introducción 6](#_Toc341342827)

[8. Objetivos que se alcanzarán 10](#_Toc341342828)

[9. Relación con la docencia cursada 10](#_Toc341342829)

[10. Viabilidad 11](#_Toc341342830)

[11. Estado del arte y fundamentación teórica 11](#_Toc341342831)

[12. Conclusión 14](#_Toc341342832)

[13. Bibliografía 15](#_Toc341342833)

**PRACTICA CERO**

Acercamiento al lenguaje de programación Java y a sus herramientas de desarrollo

**Fecha de entrega**: 22 de Noviembre de 2012

1. **Requisitos**

Para realizar esta práctica es preciso conocer el temario impartido en clase, comprendido en los temas primero y segundo.

En el tema primero y segundo podemos encontrar los puntos que definen un software de calidad, los cuales es importante tener en cuenta a la hora de realizar un desarrollo de software:

* Corrección: Realizar las tareas descritas en la especificación.
* Robustez: Capacidad de responder adecuadamente ante condiciones excepcionales
* Eficiencia: Utilizar de forma correcta los recursos de los que disponemos.
* Probabilidad: Poder ejecutar la aplicación en cualquier plataforma.
* Integridad: Es la característica de un sistema para protegerse de elementos o hechos que no tengan acceso.
* Facilidad de uso: La facilidad que pueda tener cualquier usuario de manejar la aplicación, sobre todo, sin conocer los estándares.
* Verificabilidad: La facilidad de verificar el programa.
* Compatibilidad: Posibilidad de poder cambiar los diferentes elementos de software, frente a cambios de la especificación.
* Reutilización: La posibilidad de usar el sistema o parte de él cuantas veces se quiera.
* Extensibilidad: La capacidad de ser escalable.
* Facilidad de mantenimiento del software: La capacidad de modificación del sistema o parte de él teniendo el menor impacto posible.

Es necesario conocer el paradigma de la programación orientada a objetos, así como los elementos que la componen, y los procesos para utilizarla. La programación orientada a objetos es un paradigma que usa objetos y sus interacciones, se basa en la idea natural de la existencia de un mundo lleno de objetos que relacionándose e interactuando entre sí alcanzan un fin u objetivo común.

Es una forma de programar que permite que el código se pueda reutilizar. También es una manera de representar el problema lo más parecido a la realidad.

Los elementos que componen el paradigma de la programación orientada a objetos son las clases, los objetos, el paso de mensajes, los métodos, los atributos, y el estado. Estos son los conceptos que se deben conocer de cara a esta práctica.

* Clase:

Es un conjunto coherente que consiste en un tipo particular de metadatos (información sobre los datos, nos permite obtener datos).

Es un modelo que describe el estado y el comportamiento que tienen todos los objetos del mundo real.

Encapsula el estado y el comportamiento del concepto que representa, encapsula los atributos y comportamientos.

* Objeto:

Es el encapsulamiento de un conjunto de operaciones (métodos) que pueden ser invocados externamente, y de un estado que recuerda el efecto de los servicios.

Propiedades de un objeto, el tiempo de vida, estado, y el comportamiento definido por los métodos.

* Paso de mensajes:

Mediante un mensaje un objeto solicita a otro objeto que realice una acción determinada o que modifique su estado.

Toda comunicación entre objetos debe realizarse en forma de mensajes.

Un mensaje en un objeto es la acción de efectuar una llamada a un método de otro objeto.

* Instancia:

Un objeto se define como la instancia de una clase, lo conseguimos mediante un proceso de instanciación, y cuando dejan de existir se dice que son destruidos. Guardan el invariante de la clase.

* Método:

Es la formalización de la abstracción del comportamiento. Un método es una subrutina asociada exclusivamente a una clase.

* Estado:

Todo objeto posee un estado, definido por sus atributos y que define las propiedades del objeto.

El estado de un objeto es la apariencia que el objeto presenta al usuario.

También es necesario conocer que es la abstracción, ya que es el medio por que se consigue crear las clases que representan un problema. La abstracción es el proceso de simplificar o modelar la realidad.

La abstracción se define como, separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción. También es una operación intelectual que ignora selectivamente partes de un todo para facilitar su comprensión.

* La abstracción es un proceso natural y mental.
* El lenguaje natural es un proceso de abstracción.
* Una aplicación es una descripción abstracta de un fenómeno que existe en el mundo real.
* La abstracción se realiza en un grado o nivel.

Tipos de abstracción: Abstracción procedimental, abstracción de datos, y abstracción de iteración.

Es preciso conocer que tipos de dependencia existen, ya que hay que tenerlos en cuenta durante el proceso de abstracción:

* Cohesión: Cada módulo se refiere a un único proceso o entidad. Nos dice que la información que almacena una clase debe de ser coherente y debe estar (en la medida de lo posible) relacionada con la clase.
* Acoplamiento: Mide el grado de relación de un módulo con los demás. Es la idea de tener las clases lo menos ligadas entre sí que se pueda.

También es importante conoces la sobrecarga. Es la posibilidad de tener dos o más métodos con el mismo nombre pero funcionalidad diferente. Se usará una u otra dependiendo de la firma del método. La sobrecarga se da siempre dentro de una sola clase.

Para realizar esta práctica se precisa de una plataforma con el sistema Java instalado, tanto el sistema de ejecución (JRE, Java Runtime Enviroment) como el conjunto de herramientas y librerías de desarrollo de Java (JDK, Java Developer Kit).

Es necesario conocer algunas las herramientas de desarrollo de java para poder realizar la práctica: java, javac, y javadoc.

* java: es la máquina virtual que interpreta los ficheros pre-compilados de java.
* javac: es la herramienta que compila las fuentes de java para que sean interpretadas por la máquina virtual.
* javadoc: es la herramienta que genera la documentación a partir de las fuentes, esta documentación es generada en formato html (página web).
* jar: es la herramienta que combina múltiples ficheros in un archivo de extensión jar. Este tipo de ficheros contienen las clases compiladas, y permiten su ejecución.

1. **Objetivos**

El objetivo de la práctica es la introducción al leguaje java. Para ello vamos a hacer una aplicación que sea capad de sumar, restar, multiplicar, dividir números racionales. Tiene que ser capad de leer datos tipo int, float y double, y trabajar con arrays de tipos de dato básicos.

1. **Resumen**

Se quiere hacer una aplicación que sume, reste, divide y multiplique números racionales y que opere con números enteros (int), con números de coma flotante (float y double).

También comprenderá operaciones con listas de datos (arrays), tales como la copia de una lista y la inserción de un elemento.

1. **Abstract**

It must make an application that add, subtract, divide and multiply rational numbers, operating in int, float and double in.

Also include operations with data list (arrays), such as a copy of a list and the inclusion of an item.

1. **Observaciones**

Esta práctica no debería ser una práctica que acarree muchos problemas, ya que es una práctica introductoria al lenguaje Java y sus herramientas.

1. **Sugerencias**

Lo que más nos está costando es hacer el enunciado del ejercicio, la próxima vez podríamos tener uno de ejemplo para tenerlo de referencia.

1. **Introducción**
2. Paradigma de la programación

Paradigma de programación, es una propuesta tecnológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que unívocamente trata de resolver uno o varios problemas claramente delimitados. La resolución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software. Tiene una estrecha relación con la formalización de determinados lenguajes en su momento de definición. Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso ya que nuevos paradigmas aportan nuevas o mejores soluciones que la sustituyen parcial o totalmente.

* 1. Paradigma imperativo

Es un [paradigma de programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al [computador](http://es.wikipedia.org/wiki/Computador) cómo realizar una tarea.

Los lenguajes imperativos de alto nivel usan [variables](http://es.wikipedia.org/wiki/Variable_%28programaci%C3%B3n%29) y sentencias más complejas, pero aún siguen el mismo paradigma. Las recetas y las listas de revisión de procesos, a pesar de no ser programas de computadora, son también conceptos familiares similares en estilo a la programación imperativa; cada paso es una instrucción, y el mundo físico guarda el estado (Zoom).

* 1. Paradigma funcional

Es un [paradigma de programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) declarativa basado en la utilización de [funciones aritméticas](http://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica) que no maneja datos mutables o de estado. Enfatiza la aplicación de funciones, en contraste con el estilo de [programación imperativa](http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa), que enfatiza los cambios de estado. La programación funcional tiene sus raíces en el [cálculo lambda](http://es.wikipedia.org/wiki/C%C3%A1lculo_lambda), un sistema formal desarrollado en los 1930s para investigar la definición de función, la aplicación de las funciones y la recursión. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

En la práctica, la diferencia entre una función matemática y la noción de una "función" utilizada en la programación imperativa es que las funciones imperativas pueden tener efectos secundarios, al cambiar el valor de cálculos realizados previamente. Por esta razón carecen de transparencia referencial, es decir, la misma expresión sintáctica puede resultar en valores diferentes en diferentes momentos dependiendo del estado del programa siendo ejecutado. Con código funcional, en contraste, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función. Al eliminar los efectos secundarios se puede entender y predecir el comportamiento de un programa mucho más fácilmente, y esta es una de las principales motivaciones para utilizar la programación funcional.

* 1. Paradigma lógico

La programación lógica gira en torno al concepto de predicado, o relación entre elementos. La programación funcional se basa en el concepto de función (que no es más que una evolución de los predicados), de corte más matemático.

La mayoría de los lenguajes de programación lógica se basan en la [teoría lógica de primer orden](http://es.wikipedia.org/wiki/Teor%C3%ADa_del_orden), aunque también incorporan algunos comportamientos de orden superior como la [lógica difusa](http://es.wikipedia.org/wiki/L%C3%B3gica_difusa). En este sentido, destacan los lenguajes funcionales, ya que se basan en el [cálculo lambda](http://es.wikipedia.org/wiki/C%C3%A1lculo_lambda), que es la única teoría lógica de orden superior que es demostradamente [computable](http://es.wikipedia.org/wiki/Teor%C3%ADa_de_la_computabilidad) (hasta el momento).

* 1. Paradigma declarativo

En contraposición a la [programación imperativa](http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) es un [paradigma de programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que está basado en el desarrollo de [programas](http://es.wikipedia.org/wiki/Software) especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla (tan sólo se le indica a la [computadora](http://es.wikipedia.org/wiki/Computadora) que es lo que se desea obtener o que es lo que se está buscando). No existen asignaciones destructivas, y las variables son utilizadas con [Transparencia referencial](http://es.wikipedia.org/wiki/Transparencia_referencial).

En la [programación imperativa](http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa) se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un [algoritmo](http://es.wikipedia.org/wiki/Algoritmo) en el que se describen los pasos necesarios para solucionar el problema.

En la programación declarativa las sentencias que se utilizan lo que hacen es describir el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de [inferencia](http://es.wikipedia.org/wiki/Inferencia) de información a partir de la descripción realizada.

* 1. Paradigma orientado a objetos

Es un [paradigma de programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n) que usa los [objetos](http://es.wikipedia.org/wiki/Objetos_%28programaci%C3%B3n_orientada_a_objetos%29) en sus interacciones, para diseñar aplicaciones y programas [informáticos](http://es.wikipedia.org/wiki/Inform%C3%A1tica). Está basado en varias técnicas, incluyendo [herencia](http://es.wikipedia.org/wiki/Herencia_%28inform%C3%A1tica%29), [cohesión](http://es.wikipedia.org/w/index.php?title=Cohesi%C3%B3n_%28inform%C3%A1tica%29&action=edit&redlink=1), [abstracción](http://es.wikipedia.org/wiki/Abstracci%C3%B3n_%28inform%C3%A1tica%29), [polimorfismo](http://es.wikipedia.org/wiki/Polimorfismo_%28inform%C3%A1tica%29), [acoplamiento](http://es.wikipedia.org/w/index.php?title=Acoplamiento_%28inform%C3%A1tica%29&action=edit&redlink=1) y [encapsulamiento](http://es.wikipedia.org/wiki/Encapsulamiento_%28inform%C3%A1tica%29). Su uso se popularizó a principios de la década de los años 1990. En la actualidad, existe variedad de lenguajes de programación que soportan la orientación a objetos.

Los objetos son entidades que tienen un determinado estado, comportamiento (método) e identidad:

* El estado está compuesto de datos o informaciones; serán uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
* El comportamiento está definido por los [métodos](http://es.wikipedia.org/wiki/M%C3%A9todo_%28inform%C3%A1tica%29) o mensajes a los que sabe responder dicho objeto, es decir, qué operaciones se pueden realizar con él.
* La identidad es una propiedad de un objeto que lo diferencia del resto; dicho con otras palabras, es su identificador (concepto análogo al de identificador de una [variable](http://es.wikipedia.org/wiki/Variable_%28programaci%C3%B3n%29) o una [constante](http://es.wikipedia.org/wiki/Constante_%28programaci%C3%B3n%29)).

Un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados [métodos](http://es.wikipedia.org/wiki/M%C3%A9todo_%28programaci%C3%B3n_orientada_a_objetos%29), que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separa el estado y el comportamiento.

Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El [programador](http://es.wikipedia.org/wiki/Programador) debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a alguno de ellos. Hacerlo podría producir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que manejen a las primeras por el otro. De esta manera se estaría realizando una [programación estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) camuflada en un lenguaje de programación orientado a objetos.

La **POO** (Programación orientada a objetos) difiere de la [programación estructurada](http://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada) tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada solo se escriben funciones que procesan datos. Los programadores que emplean POO, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

1. Paradigma que se va a emplear

Se va a emplear en esta práctica el paradigma orientado a objetos, por las siguientes ventajas:

La programación orientada a objetos (POO) permite acortar la distancia entre el software que estamos desarrollando y la realidad, debido a que en la POO se busca representar las entidades como existen en la realidad, con sus características y comportamiento. Además, gracias a la posibilidad de heredar atributos y métodos de otras clases, lograremos reutilizar código fuente, por lo que los tiempos de desarrollo bajarán.

La POO también incorpora el concepto de encapsulamiento, lo cual nos permite proteger las propiedades de un objeto. La protección la obtenemos definiendo a los atributos como privados y a los métodos como públicos, los cuales serán los que accederán a las propiedades. De esta forma, la manipulación de los valores de los atributos estará controlada.

La POO busca abstraer los objetos del contexto, de forma de que se puedan diferenciar los distintos objetos que están interactuando y luego representar a cada uno de manera separada. Esta interacción luego se realizara a través de los métodos.

Además cumple o se adapta a cuatro puntos fundamentales para el desarrollo de un proyecto de software:

* Complejidad del dominio del problema.
* Dificultad de gestionar el proceso de desarrollo.
* La flexibilidad del software para expresar cualquier abstracción.
* Los problemas de comportamiento de los sistemas.

1. Introducción a la práctica

Los números reales en el ámbito de la informática no son representables con toda la precisión que se necesita, por que un número de este conjunto puede tener dígitos hasta el infinito y las computadoras no tienen espacio suficiente para almacenar estos datos.

Por tanto cuando se realizan operaciones, como puede ser la comparación de dos números reales de igual valor, la igualdad de dichos números puede no ser cierta.

En el campo de las matemáticas es muy habitual encontrarse con agrupaciones de número u otros elementos debido a que comparten entre sí las mismas características y/o propiedades.

Por tanto en el ámbito de la informática se hace necesario representar agrupaciones de elementos, bajo un criterio semejante. En el ámbito de esta práctica están comprendidas las listas de elementos que son del mismo tipo, es decir, tienen las mismas características.

1. Conceptos y definiciones

Números racionales: Todo número que puede representarse como el cociente de dos números enteros, engloba a los números enteros y naturales.

Int: Es un número entero de 32 bits.

Float: Es un número en coma flotante de 32 bits.

Double: Es un número en coma flotante de 64 bits.

Array: Es un conjunto contiguo de elementos del mismo tipo (enteros, float, double...).

1. **Objetivos que se alcanzarán**

* Acercamiento a la programación orientada a objetos, y a la plataforma de desarrollo de java.
* Aprender a documentar una práctica para futuros proyectos.

1. **Relación con la docencia cursada**

* Esta práctica nos introduce al lenguaje Java y al uso de las herramientas de desarrollo de Java.
* En ella van a estar involucrados los conocimientos adquiridos sobre la programación orientada a objetos. Como el proceso de abstracción sobre problema del que se va a abordar.

El modelado de clases, la utilización de objetos como elementos concretos del problema. La interacción de estos elementos por el sistema de paso de mensajes que caracteriza a la programación orientada a objetos.

1. **Viabilidad**

Esta práctica resulta viable dado que no tiene una gran complejidad en cuanto a las operaciones que se tienen que realizar.

Pero desde el punto de vista de los conocimientos específicos sobre el lenguaje Java impartidos en las sesiones, la práctica puede tornarse compleja, dado que lo único que se conoce sobre la sintaxis del lenguaje son los tipos de comentarios y los modificadores de atributos y métodos.

1. **Estado del arte y fundamentación teórica**

A la hora de realizar las operaciones con números de coma flotante existe un problema de precisión que no permite una comparación directa entre estos.

* **Antecedente**

Buscando se ha encontrado una clase de números racionales ya hecha y con todas las operaciones que se especifican para esa clase perteneciente a la Universidad de Stanford. Por eso nuestra clase números racionales está fuertemente basada en la de universidad Stanford (véase la bibliografía, sección de fuentes de la práctica).

De los arrays no se ha encontrado nada útil puesto que ya está definido en java, lo mismo ocurre con los números en coma flotante.

* **Fundamentos teóricos**
* Números racionales: Se llama número racional a todo número que puede representarse como el cociente de dos números enteros. El término «racional» alude a fracción o parte de un todo. El conjunto de los números racionales se denota por Q. Este conjunto de números incluye a los números enteros (Z), y es un subconjunto de los números reales (R).

La escritura decimal de un número racional es, o bien un número decimal finito, o bien periódico.

Un número real que no es racional, se llama número irracional.

* Coma flotantes: La representación de coma flotante, es una forma de notación científica usada en los CPU, GPU, FPU, etc, con la cual se pueden representar números reales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas.
* Arrays: En programación, una matriz o vector (llamados en inglés arrays) es una zona de almacenamiento continuo, que contiene una serie de elementos del mismo tipo, los elementos de la matriz. Desde el punto de vista lógico una matriz se puede ver como un conjunto de elementos ordenados en fila (o filas y columnas si tuviera dos dimensiones).

Estas estructuras de datos son adecuadas para situaciones en las que el acceso a los datos se realice de forma aleatoria e impredecible. Por el contrario, si los elementos pueden estar ordenados y se va a utilizar acceso secuencial sería más adecuado utilizar una lista, ya que esta estructura puede cambiar de tamaño fácilmente durante la ejecución de un programa.

* **Propuestas de solución**:
  1. Primera solución:
     + Diseñar mediante el proceso de abstracción una clase que represente los números reales.
     + Crear diferentes clases que alberguen las operaciones básicas comprendidas en la práctica. Realizar una clase por cada elemento del problema involucrado en las operaciones que se tratarán.
     + Para tratar el problema de precisión de los números de coma flotante se tratará de descomponer el número en su parte entera, en la mantisa, y en la característica.
  2. Segunda solución:
     + Diseñar mediante el proceso de abstracción una clase que represente los números reales, la cual comprenda el numero racional que representa y las operaciones que se van a realizar con él.
     + Diseñar una clase que representará los números de coma flotante, esta clase tendrá los métodos que permiten realizar las operaciones comprendidas en esta práctica (suma, multiplicación, etc…).
     + Realizar un clase que contenga métodos para poder trabajar con listas de elementos (arrays). Dado que existen muchos tipos de elementos no se realizará una clase que contenga tantos atributos como tipos de datos se vaya a trabajar. Solo contendrá los métodos de forma estática.
     + Para tratar el problema de precisión de los número de coma flotante se utilizará los tipos básicos de Java (float, double) ya que el compilador de Java está optimizado para no tener pérdidas de precisión y asegurar que dos números de cómo flotante son iguales.
* **Desestimaciones**

Se desestima la primera solución por no cumplir con las directrices de la programación orientada a objetos, ya que uno de los principales problemas de esta solución es que las clases que comprenden las operaciones están diseñadas de forma estática, por tanto ni son un objeto ni se manejan de forma dinámica en la memoria.

* **Solución escogida**

Se ha elegido la segunda solución porque se cree que se cumple con el paradigma de la programación orientada a objetos.

Con esta solución se ha podido realizar la práctica cumpliendo con las especificaciones de esta, y superando las pruebas realizadas para garantizar que se cumple con la especificación y que su funcionamiento es correcto (dentro del ámbito de las pruebas desarrolladas para esta práctica).

En cuanto a la viabilidad, no se han encontrado grandes problemas al tratar de resolver la práctica, la envergadura de esta es pequeña. En conclusión la viabilidad es alta, y a demás para un corto plazo de tiempo.

1. **Conclusión**

Al final de la práctica hemos mejorado nuestro nivel de java, entendemos mejor la instanciación de objetos, creación de clases, utilización de métodos, etc. Además hemos comprendido mejor las pruebas necesarias para testear bien un programa y como realizarlas correctamente.

También nos hemos familiarizado con la sintaxis de java y la edición y compilación de este sin un entorno de desarrollo como es eclipse.

Respecto a la documentación necesaria, ahora sabemos con más precisión qué clase de documentación se necesita y para que se requiere.

* **Líneas Futuras**

De cara al futuro se espera seguir mejorando en el lenguaje java, y seguir aprendiendo más sobre la programación orientada a objetos.

También se espera aprender a manejar editores de lenguaje java como por ejemplo eclipse y las diversas herramientas presentes en este y en java.

1. **Bibliografía**

Estructura del documento actual:

* Práctica WALL·E de la universidad Complutense de Madrid.
* Transparencias práctica cero: Aproximación al lenguaje Java, del profesor Francisco Javier Crespo.

Contenido de las fuentes de la aplicación:

* <http://www.java-forums.org/new-java/32898-biginteger-rational-cs106a-stanford-university.html>

Datos:

* Números Racionales: <http://es.wikipedia.org/wiki/N%C3%BAmero_racional>
* Coma flotante: <http://es.wikipedia.org/wiki/Coma_flotante>
* Arrays: <http://es.wikipedia.org/wiki/Vector_(inform%C3%A1tica>)

Herramientas de Java:

* <http://docs.oracle.com/javase/1.5.0/docs/tooldocs/>

Paradigmas de la programación:

* [http://es.wikipedia.org/wiki/Paradigma\_de\_programación](http://es.wikipedia.org/wiki/Paradigma_de_programaci%C3%B3n)
* [http://es.wikipedia.org/wiki/Programación\_imperativa](http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa)
* [http://es.wikipedia.org/wiki/Programación\_funcional](http://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional)
* [http://es.wikipedia.org/wiki/Programación\_lógica](http://es.wikipedia.org/wiki/Programaci%C3%B3n_l%C3%B3gica)
* [http://es.wikipedia.org/wiki/Programación\_declarativa](http://es.wikipedia.org/wiki/Programaci%C3%B3n_declarativa)
* [http://es.wikipedia.org/wiki/Programación\_orientada\_a\_objetos](http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos)
* <http://gonblog22.blogspot.com.es/2010/07/ventajas-de-la-programacion-orientada.html>
* [http://es.wikipedia.org/wiki/Grasp - A.lta\_cohesi.C3.B3n](http://es.wikipedia.org/wiki/Grasp#A.lta_cohesi.C3.B3n)

Citas de autores:

* Saussure Chomsky.
* Mario G. Piattini.