Prof. Danilo Alvarellos
Prof. Gonzalo Pastor

CETP 2012

INDICE

Introducción

La presente Guía está destinada a los profesores de Programación 1 del CETP, cuyo contenido para el año 2012 estará orientado al Lenguaje Java de Programación.
Esta asignatura mantiene desde 1997 en el bachillerato de Informática, un contenido orientado al Lenguaje C que es necesario modificar para mantener moderno y actualizado este curso.
Como el lenguaje Java está basado en conceptos de C, el docente de Lenguaje C, necesitará una leve actualización para los nuevos contenidos, siendo este texto una orientación para el mismo.
Como el lenguaje Java está orientado a objetos esto plantea algunas problemáticas sobre el curso:

- La opción de dictar un lenguaje orientado a objetos pretende preparar de mejor manera al alumnado cuando llega a Programación III, donde se encontrará con un lenguaje Visual orientado a objetos. De esta manera se intenta obtener resultados más óptimos en este año culminante del bachillerato.
- Como la mayoría del alumnado recién se inicia en la programación, comenzar trabajando con profundidad en objetos agrega más problemáticas a nuestro trabajo, por lo que se pretende apostar a la programación estructurada, trabajando los objetos Java al mínimo, dejando para Programación 2 una profundización del tema.
- La documentación adecuada para programación orientada a objetos es UML, pero se considera agregar demasiada complejidad al primer curso de programación, por lo tanto se aconseja trabajar con Pseudocódigo (y solo Pseudocódigo) como herramienta de documentación y razonamiento para el alumno que se inicia.

Esta guía pretende aportar contenidos del curso y sugerencias metodológicas al docente, dando algunos elementos que permitan guiar el curso de Programación 1. Como sugerencia, la misma no es definitiva, permitiendo al docente agregar elementos al curso, pero considerando que esta guía es mínima, por lo tanto todo el contenido propuesto debe ser dictado en el curso.
Existe en Internet y en formato impreso mucho material sobre Java. Esta guía permite orientar al docente sobre los temas a dictar y el encare de los mismos.

NOTA: Para el 2011 se va a implementar Java en todas las Programación 1 del país. Esto incluye Programación 1 del EMT de Informática y Programacióndel EMP de Reparación PC. Se implementará Java en Programación 2 a partir de 2012.

Inspección Informática
CETP

Algunos Comentarios

Un programa en Java se puede escribir en cualquier editor de textos (el block de notas por ejemplo), luego se puede compilar por separado y ejecutar.
Tomando en cuenta al alumnado que recibirá el curso, consideramos adecuado usar una IDE más atractiva y apropiada. Para ello este curso estará basado en ECLIPSE, ya que es una IDE de software libre, pensada y documentada hacia Java.
Como se considera que al instalar ECLIPSE ya se incluirán una serie de componentes de Java, estos mismos no se mencionarán en este curso.
Se le sugerirá al alumno que simplemente baje ECLIPSE y lo instale.

- Se puede bajar ECLIPSE gratuito de www.eclipse.org/downloads/ (recomendamos Eclipse Classic 3.7.1 que ya contiene el compilador de Java).
- Se puede bajar el software gratuito de JAVA de http://www.java.com/es/download/

Encontrará en el presente trabajo que en un recuadro de doble línea se irán agregando comentarios de índole didáctico destinados al docente.
Todo docente debe trabajar con el alumnado en el uso de lenguaje informático adecuado. Sin exceder en esta tarea el alumno debe llamar a las cosas por su nombre evitando el uso de términos genéricos como “eso”, “ahí”, “cosa”...
Entre las palabras que debe resaltar es el uso de “computador” o “computadora” en vez de “ordenador”. A pesar de que “ordenador” es una palabra correcta, la misma no es de uso en Uruguay siendo común en otros países.

El cuaderno debe ser una herramienta de consulta para el alumno.
El docente debe guiarlo para que mantenga un cuaderno actualizado dictándole los conceptos indispensables que en él deben existir.
El alumno debe copiar en el cuaderno todos aquellos elementos que necesitará consultar a futuro como ser estructuras de las sentencias, ejemplos básicos u otros.
Es parte de la formación del alumno tener sus propios materiales y aprender a discernir lo indispensable, lo importante, lo básico y lo innecesario.

Les deseamos éxito en esta enorme tarea como docente.

Tema 1: ¿Qué es un programa?

SUGERENCIA: Ya que el curso es de programación, utilice esta palabra como motivador y comience por hacer pensar a los estudiantes donde escuchó la palabra programa.
Tome nota en el pizarrón de los ejemplos y ayúdelos a razonar que características tienen los programas que ellos mencionaron.

Ejercicio 1:
Pídale a los alumnos que:

- Piense en la vida diaria donde aparece la palabra Programa
- Cite ejemplos
- Piense que características comunes tienen las distintas acepciones
- Busque una definición más acertada de lo que es un programa.

Intente usar los conceptos que los propios alumnos usaron.

Posible definición de Programa Informático:

- Instrucciones de computación estructuradas y ordenadas que al ejecutarse hacen que una computadora realice una función particular
- Un Programa informático (software) es la unión de una secuencia de instrucciones que una computadora puede interpretar y ejecutar y una (o varias) estructuras de datos que almacena la información independiente de las instrucciones que dicha secuencia de instrucciones maneja. Para ello se usan **lenguajes de programación** que sirven para programar la secuencia de instrucciones requerida.

Características de un Lenguaje de Programación

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora.
Consiste en un conjunto de reglas sintácticas y semánticas que definen un programa informático.

Relacione los conceptos de sintaxis y semántica con el lenguaje hablado.

Sintaxis (extraído de la RAE: Real Academia Española)
(Del lat. syntaxis, y este del gr. σύνταξις, de συντάσσειν, coordinar).
1. f. Gram. Parte de la gramática que enseña a coordinar y unir las palabras para formar las oraciones y expresar conceptos.
2. f. Inform. Conjunto de reglas que definen las secuencias correctas de los elementos de un lenguaje de programación.

Semántico, ca.
(Del gr. σημαντικός, significativo).
1. adj. Perteneciente o relativo a la significación de las palabras.
2. f. Estudio del significado de los signos lingüísticos y de sus combinaciones, desde un punto de vista sincrónico o diacrónico

Afine el concepto de Lenguaje de Programación. Discuta con los alumnos hasta que logre una definición aceptable.

Lenguaje de Programación

- Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que definen un programa informático.

- Aunque muchas veces se usa lenguaje de programación y lenguaje informático como si fuesen sinónimos, no tiene por qué ser así, ya que los lenguajes informáticos engloban a los lenguajes de programación y a otros más, como, por ejemplo, el HTML.

Relacione programas con software. Delimite ambos.

Software

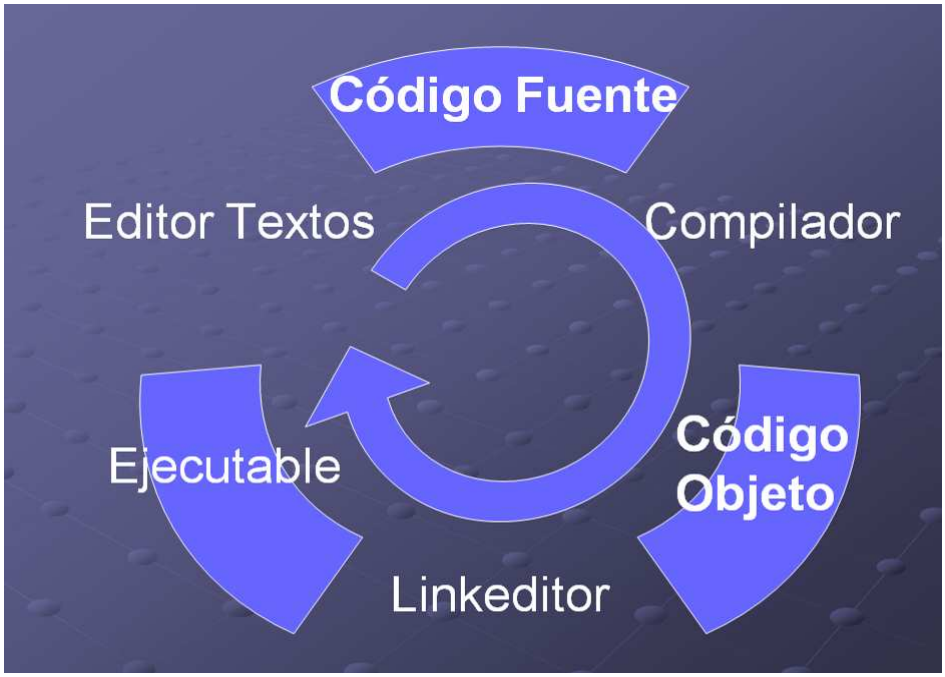
- Software es el conjunto de programas que puede ejecutar el hardware para la realización de las tareas de computación a las que se destina.
- Se trata del conjunto de instrucciones que permite la utilización del computador. El software es la parte intangible de la computadora, es decir: programas, aplicaciones etc.

Intente definir las etapas de un programa. Diferencie el usuario del programador. Haga un acercamiento al rol del programador, háblele de entender al usuario y de las dos visiones, comunes y diferentes de usuario y programador.

Formas

El software adopta varias formas en distintos momentos de su ciclo de vida:

- Código fuente: escrito por programadores. Contiene el conjunto de instrucciones, inteligibles por el ser humano, destinadas a la computadora.
- Código objeto: resultado del uso de un compilador sobre el código fuente. El código objeto no es directamente inteligible por el ser humano, pero tampoco es directamente entendible por la computadora. Se trata de una representación intermedia del código fuente. En Java se llama bytecode.
- Código ejecutable: resultado de linkeditar uno o varios fragmentos de código objeto. Constituye un archivo binario con un formato tal que el sistema operativo es capaz de cargarlo en la memoria de un computador, y proceder a su ejecución. El código ejecutable es directamente inteligible por la computadora.



Licencia

Trabaje con los alumnos sobre el papel que juega el software propietario y el libre. Analice con ellos el rol del programador en la licencia del software.

Una Licencia de Software es la autorización o permiso concedida por el autor para utilizar su obra de una forma convenida habiendo marcado unos límites y derechos respecto a su uso.
La Licencia puede, restringir el territorio de aplicación del programa, su plazo de duración o cualquier otra cláusula que el autor decida incluir.

Las licencias sobre obras intelectuales originales son una forma de protección proporcionada por las leyes vigentes que incluyen tanto los trabajos publicados como los pendientes de publicación, y otorgan al autor el derecho exclusivo para autorizar a otros a utilizar, modificar y/o redistribuir su obra original.

El autor del software puede autorizar o limitar el uso, modificación y/o redistribución de su obra adscribiéndose a un determinado tipo de licencia.

El software propietario (también llamado software no libre, software privativo, software privado, software con propietario o software de propiedad) se refiere a cualquier programa informático en el que los usuarios tienen limitadas las posibilidades de usarlo, modificarlo o redistribuirlo (con o sin modificaciones), o cuyo código fuente no está disponible o el acceso a éste se encuentra restringido.
En el software no libre de una persona física o jurídica (compañía, corporación, fundación, etc.) posee los derechos de autor sobre un software negando o no otorgando, al mismo tiempo, los derechos de usar el programa con cualquier propósito; de estudiar cómo funciona el programa y adaptarlo a las propias necesidades; de distribuir copias; o de mejorar el programa y hacer públicas las mejoras (para esto el acceso al código fuente es un requisito previo).
De esta manera, un software sigue siendo no libre aún si el código fuente es hecho público, cuando se mantiene la reserva de derechos sobre el uso, modificación o distribución (por ejemplo, la versión comercial de SSH de Microsoft).

Freeware es un software de computadora que se distribuye sin cargo. A veces se incluye el código fuente, pero no es lo usual.
El freeware suele incluir una licencia de uso, que permite su redistribución pero con algunas restricciones, como no modificar la aplicación en sí, ni venderla.

Shareware es una modalidad de distribución de software (juegos o programas) para que el mismo pueda ser evaluado de forma gratuita, pero generalmente por un tiempo especificado, aunque también las limitaciones pueden estar en algunas de las formas de uso o las capacidades finales. Para adquirir una licencia de software que permite el uso del software de manera completa se requiere de un pago

Software Libre

Software libre es el aquel que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente.
El software libre suele estar disponible gratuitamente en Internet, o a precio de la distribución a través de otros medios; sin embargo no es obligatorio que sea así y, aunque conserve su carácter de libre, puede ser vendido comercialmente.

El software libre, garantiza los derechos de modificación y redistribución de dichas versiones modificadas del programa.
No debe confundirse "software libre" con software de dominio público. Éste último es aquél por el que no es necesario solicitar ninguna licencia y cuyos derechos de explotación son para toda la humanidad, porque pertenece a todos por igual.

Acerquemos al alumno al lugar que ocupa Java.
Comente las características de los otros lenguajes, pero superficialmente.
Más adelante se trabajará en profundidad.

Una clasificación de los lenguajes de programación

Nivel	Nombre	Velocidad	Programación
Primer Nivel	Lenguaje máquina. 0 / 1	Máxima	Muy difícil
Segundo Nivel	Lenguaje ensamblador Assembler.	Máxima	Difícil
Tercer Nivel	Lenguajes de alto nivel. C, Pascal, Visual Basic	Rápida	Fácil
Cuarto Nivel	Lenguajes de 4a. Generación GeneXus	Lenta	Muy Rápida
Quinto Nivel???			

Al entrar en la parte de documentación es necesario tener en cuenta qué técnica usar.
A pesar que UML es el modelo adecuado para trabajar objetos, por su complejidad y la falta de conocimientos del alumno se usará pseudocódigo como base del razonamiento.

Pseudocódigo

- Herramienta que permite pasar las ideas al papel, en español y siguiendo unas pocas reglas.
- Es el código no ejecutable de un programa que se usa como una ayuda para desarrollar y documentar programas estructurados
- Herramienta de análisis de programación. Versiones falsificadas y abreviadas de las actuales instrucciones de computadora que son escritas en lenguaje ordinario natural.

Ejemplo de Pseudocódigo

- Calentar comida con un microondas
  - Inicio
    Colocar dentro la comida a calentar
    Cerrar la puerta del microondas
    Seleccionar la potencia
    Seleccionar el tiempo
    Iniciar el microondas
    Esperar a que termine
    Sacar la comida
  - Fin

Ejercicio 2

- Hacer el pseudocódigo del proceso de lavado de un lavarropas automático
- Considerar que está cargado: tiene ropa, puerta cerrada, agua conectada, luz, se pone el programa más largo y comienza
- ¿Cómo sigue?
- Discutirlo
- Pasarlo en limpio

Pseudocódigo (Posible solución)

- Proceso de lavado de un lavarropas autom.
- Inicio
  Carga el agua
  Remoja
  Prelavado
  Saca el agua
  Carga el agua
  Lavado
  Saca el agua
  Carga el agua
  Enjuague
  Saca el agua
  Centrifugado
- Fin

Tema 2: Algoritmos y Programación Estructurada

Algoritmos

Usualmente, nos enfrentamos a situaciones que requieren solucionarse.
Por ejemplo: cómo llamar por teléfono.
Conforme estas situaciones se hacen más complejas, notamos que encontrar la solución se puede volver menos evidente.
Es por eso, que se han desarrollado métodos para ayudarnos a encontrar dichas soluciones, los cuales son estudiados en una disciplina conocida como Algoritmia.
Ésta se centra en el estudio de los algoritmos.

¿Qué es un algoritmo?

Un algoritmo es un conjunto de instrucciones que conducen a la solución de un problema o situación por resolver.
Un algoritmo expresa un procedimiento paso a paso para solucionar un problema dado.

El primer algoritmo fue escrito por Al-Khwarizmi quien nació en 780 en Bagdad y murió en 850. La palabra algoritmo deriva de su nombre.

Supongamos que deseamos hacer una llamada telefónica.
Elaboremos una lista de pasos, para que podamos hacerla, digamos algo similar a lo siguiente:

1. Tomar el teléfono,
2. Marcar el número telefónico,
3. Esperar a que contesten,
4. Hablar un rato y
5. Terminar la llamada.

Características:

Los algoritmos pueden considerar diferentes situaciones que pudieran presentarse.
En nuestro ejemplo de la llamada, es posible que no contesten o que la persona a la que llamamos no se encuentre. También, podemos haber olvidado su número telefónico, por lo que podríamos ocuparnos primero en obtener dicho número, antes de realizar la llamada.
En fin, el algoritmo será tan complejo o simple como se necesite (o prefiera).

Las **características** que debe cumplir un algoritmo son ser:

- *Preciso*, para indicar el orden de realización de cada paso. Sólo se realiza un paso por vez.

• *Definido*, de manera que si un algoritmo se sigue dos veces, se debe obtener el mismo resultado cada vez.
• *Finito*, es decir, debe tener un número limitado de pasos.

Partes del algoritmo
Un algoritmo debe de constar de tres partes:
- Entrada, que es la información que se le proporciona al algoritmo.
- Proceso, que es la realización de las operaciones necesarias para producir la salida.
- Salida, que es la información producida por el algoritmo.

Para ilustrar las características y las partes de un algoritmo, veamos un ejemplo.
Escribamos un algoritmo para calcular el cambio (si hubiera) al pagar por un producto.
Primero, debemos conocer los datos de entrada:
- 1. Cuál es el precio de venta del artículo y
- 2. Cuánto dinero estamos entregando al pagar.

El proceso del algoritmo consiste en aplicar la resta:
- Cambio = Dinero Entregado – Precio

Finalmente, la salida del algoritmo es: el cambio recibido (si fuera el caso).

De modo que podemos expresar el algoritmo como:
1. Tomar los datos de entrada: Dinero Entregado y Precio.
2. Realizar la resta: Cambio = Dinero Entregado – Precio.
3. Decir cuánto es el cambio o que no hay tal.

También, podemos verificar que este algoritmo es: preciso (pues indica el orden de realización de los pasos), definido (pues para cierto Dinero Entregado y cierto Precio, siempre da la misma respuesta) y finito (pues consta de tres pasos).

NOTA:
- El pseudocódigo describe un algoritmo utilizando una mezcla de frases en lenguaje común, instrucciones de programación y palabras clave que definen las estructuras básicas.
- Su objetivo es permitir que el programador se centre en los aspectos lógicos de la solución a un problema.
- No siendo el pseudocódigo un lenguaje formal, varía de un programador a otro, es decir, no hay una estructura semántica ni arquitectura estándar.
- Es una herramienta ágil para el estudio y diseño de aplicaciones, en por ejemplo, un lenguaje imperativo, de tercera generación, según el método de programación estructurada.
- Es necesario limitar la profundidad del pseudocódigo yendo de lo general a lo particular. Una línea del pseudocódigo puede generar un pseudocódigo aparte para expresar su algoritmo

Programación Estructurada

- La programación estructurada es una teoría de programación que consiste en construir programas de fácil comprensión.
- La programación estructurada es especialmente útil, cuando se necesitan realizar correcciones o modificaciones después de haber concluido un programa o aplicación.
- La programación estructurada se basa en una metodología de desarrollo de programas llamada refinamiento sucesivo:

Diseño estructurado (extraído de wikipedia)

Descomposición
¿Por qué descomponer un problema en partes? Experimentalmente está comprobado que:

- Un problema complejo cuesta más de resolver que otro más sencillo
- La complejidad de un problema global es mayor que el valor de las complejidades de cada una de sus partes por separado.

Según esto, merece la pena el esfuerzo de dividir un problema grande en subproblemas más pequeños.
Si el objetivo es elaborar un programa para resolver dicho problema *grande*, cada subproblema (menos complejo) podrá ser resuelto por un módulo relativamente fácil de implementar (más que el programa global **No** dividido).
Ahora la cuestión es *¿cómo realizar la descomposición?*; realizando un estudio descendente Top-Down que nos lleve desde la concepción del problema (programa o algoritmo) global hasta identificar sus partes (módulos).
Esta técnica se repite aplicando una estrategia llamada de refinamiento sucesivo propuesta por **Niklaus Wirth**, que consiste precisamente en volver a aplicar el estudio descendente Top-Down a cada subproblema una y otra vez hasta obtener subproblemas suficientemente pequeños, que puedan ser resueltos por módulos que cumplan, en la medida de lo posible, las características deseables en un módulo en el ámbito de la programación.
En palabras del propio **Niklaus Wirth**:

- En cada paso (del refinamiento), una o varias instrucciones del programa dado, se descomponen en instrucciones más detalladas. Esta descomposición sucesiva o refinamiento de especificaciones termina cuanto todas las instrucciones están expresadas en términos de la computadora usada o del lenguaje de programación...
- Conforme se refinan las tareas, también los datos pueden ser refinados, descompuestos o estructurados, siendo lo natural refinar las especificaciones del programa y de los datos en paralelo.
- Cada paso de refinamiento implica algunas decisiones de diseño. Es importante que el programador sea consciente de los criterios subyacentes (en las decisiones de diseño adoptadas) y de la existencia de soluciones alternativas...

Problema del refinamiento sucesivo

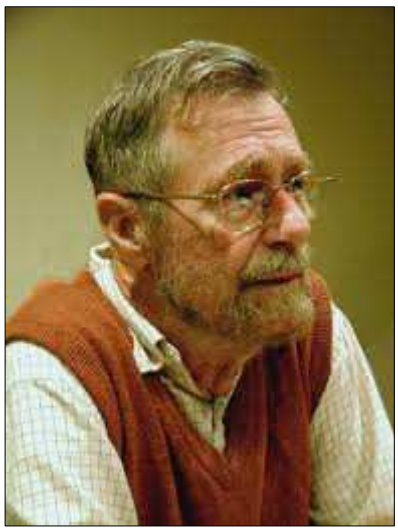
¿Cuándo parar el refinamiento?. Un refinamiento excesivo podría dar lugar a un número tan grande de módulos que haría poco práctica la descomposición. Se tendrán en cuenta estos criterios para dejar de descomponer:

- Cuando no haya tareas bien definidas.
- Cuando la interfaz de un módulo sea tan complicada como el propio módulo

Programación Estructurada

Se plantea una operación como un todo y se divide en segmentos más sencillos o de menor complejidad. Una vez terminado todos los segmentos del programa, se procede a unificar las partes.

La programación estructurada propone segregar los procesos en estructuras lo más simple posibles, las cuales se conocen como secuencia, selección e iteración.
Están disponibles en todos los lenguajes modernos de programación imperativa en forma de sentencias.
Combinando esquemas sencillos se pueden llegar a construir sistemas amplios y complejos pero de fácil entendimiento.



Dijkstra

Un famoso teorema, demostrado por Edsger Dijkstra (1930-2002) en los años sesenta, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

1. El bloque secuencial de instrucciones: órdenes ejecutadas sucesivamente.
2. La instrucción condicional: de la forma
   SI condición
   instrucción si es cierta
   SINO
   instrucción si es falsa
   FINSI
3. El bucle condicional MIENTRAS: que ejecuta una o varias órdenes repetidamente mientras la condición se cumpla.

GOTO

Los programas estructurados utilizan sólo estas tres instrucciones de control básicas o sus variantes, pero no se usa la instrucción de bifurcación incondicional GOTO
Los programas que usan GOTO se vuelven inentendibles e indepurables
Hace que el programa "salte" para cualquier lado sin ningún control ni sentido lógico.

En el sistema operativo DOS se la usa porque no es un lenguaje de programación propiamente dicho. Contiene pocas órdenes de programación muy precarias

El Lenguaje Java no contiene la sentencia GOTO.
La eliminó expresamente a diferencia de otros lenguajes que la mantienen por compatibilidad.

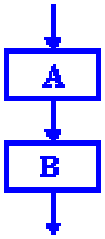
Preguntas

- Qué es una bifurcación?
- Cuándo es condicional?
- Cuándo es incondicional?
- Qué es un bucle?
- Cuáles con las sentencias de selección?
- Cuál es la de iteración?

PSEUDOCÓDIGO

Secuencia

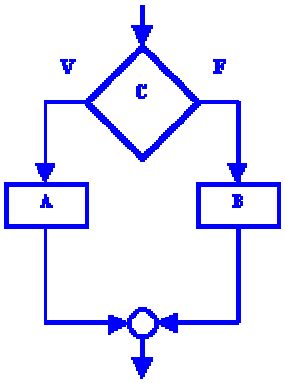
- Inicio
- Sentencia 1
- Sentencia 2
- Sentencia 3
- …
- Fin



Todas las sentencias son ejecutadas: unas tras otra.

Selección

- Inicio
  SI condición
  Sentencias 1
  SINO
  Sentencias2
  FIN SI
- Fin



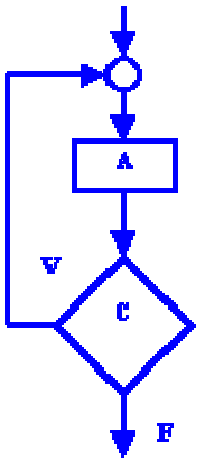
Se evalúa la condición: si es cierta se ejecutan las Sentencias 1 y si es falsa las Sentencias 2. Se ejecuta una o la otra.

PREGUNTAR:

¿Qué pasa en la selección incompleta?

Iteracción

- Inicio
  - MIENTRAS condición
    - Sentencias
  - FIN MIENTRAS
- Fin



Las sentencias dentro del MIENTRAS se ejecutan repetidamente
mientras que la condición es cierta.
Si al empezar la condición es falsa las sentencias nunca son ejecutadas

Los diagramas de flujo usados en los ejemplos anteriores son ilustrativos. No se recomienda su uso en este curso

Ejercicio 3

- Hacer un pseudocódigo para guiar a una persona a salir de esta clase hasta la calle.
- Considerar:
  - Que pueden haber puertas cerradas.
  - Doblar a la izquierda o derecha solamente, no es una orden precisa. Indicar el giro en grados
  - Este ejercicio requiere órdenes secuenciales y selectivas
- Discutirlo
- Copiarlo en el pizarrón

Ejercicio 4

- Hacer el pseudocódigo para cruzar la calle, con semáforos, de una vereda hasta la opuesta en diagonal
- Nota: Existen básicamente 2 soluciones

Tema 3: Introducción a Java

INFORMACION PARA EL DOCENTE:

Introduzca al estudiante en las características más destacables del Lenguaje Java.
No dedique demasiado tiempo a esta tarea, sino que trate de motivar al alumno a interesarse por la asignatura y el lenguaje sin profundizar demasiado.
Pero, no sea tan superficial que el estudiante no logre entender por qué trabajamos en Java y cuales son sus principales características

A continuación se agregan 8 hojas con material que el docente debe conocer, resumir y transmitir a su alumnado lo más importante.

HISTORIA DE JAVA

A. ¿Por qué se diseñó Java?

Los lenguajes de programación C y Fortran se han utilizado para diseñar algunos de los sistemas más complejos en lenguajes de programación estructurada, creciendo hasta formar complicados procedimientos. De ahí provienen términos como "código de espagueti" o "canguros" referentes a programas con múltiples saltos y un control de flujo difícilmente trazable.
No sólo se necesitaba un lenguaje de programación para tratar esta complejidad, sino un nuevo estilo de programación. Este cambio de paradigma de la programación estructurada a la programación orientada a objetos, comenzó hace 30 años con un lenguaje llamado Simula67.
El lenguaje C++ fue un intento de tomar estos principios y emplearlos dentro de las restricciones de C. Todos los compiladores de C++ eran capaces de compilar programas de C sin clases, es decir, un lenguaje capaz de interpretar dos estilos diferentes de programación. Esta compatibilidad (*"hacia atrás"*) que habitualmente se vende como una característica de C++ es precisamente su punto más débil. No es necesario utilizar un diseño orientado a objetos para programar en C++, razón por la que muchas veces las aplicaciones en este lenguaje no son realmente orientadas al objeto, perdiendo así los beneficios que este paradigma aporta.
Así Java utiliza convenciones casi idénticas para declaración de variables, paso de parámetros, y demás, pero sólo considera las partes de C++ que no estaban ya en C.
Las principales características que Java no hereda de C++ son:

- Punteros: Las direcciones de memoria son la característica más poderosa de C++. El inadecuado uso de los punteros provoca la mayoría de los errores de colisión de memoria, errores muy difíciles de detectar. Además, casi todos los virus que se han escrito aprovechan la capacidad de un programa para acceder a la memoria volátil (RAM) utilizando punteros. En Java, no existen punteros, evitando el acceso directo a la memoria volátil.
- Variables globales: Con ellas cualquier función puede producir efectos laterales, e incluso se pueden producir fallos catastróficos cuando algún otro método cambia el estado de la variable global necesaria para la realización de otros procesos. En Java lo único global es el nombre de las clases.
- *goto*: Manera rápida de arreglar un programa sin estructurar el código. Java no tiene ninguna sentencia *goto*. Sin embargo Java tiene las sentencias *break* y *continue* que cubren los casos importantes de *goto*.
- Asignación de memoria: La función *malloc* de C, asigna un número especificado de bytes de memoria devolviendo la dirección de ese bloque. La función *free* devuelve un bloque asignado al sistema para que lo utilice. Si se olvida de llamar a *free* para liberar un bloque de memoria, se están limitando los recursos del sistema, ralentizando progresivamente los programas. Si por el contrario se hace un *free* sobre un puntero ya liberado, puede ocurrir cualquier cosa. Más tarde C++ añadió *new* y *delete*, que se usan de forma similar, siendo todavía el programador, el responsable de liberar el espacio de memoria. Java no tiene funciones *malloc* ni *free*. Se utiliza el operador *new* para asignar un espacio de memoria a un objeto en el *montículo* de memoria. Con *new* no se obtiene una dirección de memoria sino un descriptor al objeto del *montículo*. La memoria real asignada a ese objeto se puede mover a la vez que el programa se ejecuta, pero sin tener

que preocuparse de ello. Cuando no tenga ninguna referencia de ningún objeto, la memoria ocupada estará disponible para que la reutilice el resto del sistema sin tener que llamar a *free* o *delete*. A esto se le llama *recogida de basura*. El recolector de basura se ejecuta siempre que el sistema esté libre, o cuando una asignación solicitada no encuentre asignación suficiente.

- Conversión de tipos insegura: Los moldeados de tipo (*type casting*) son un mecanismo poderoso de C y C++ que permite cambiar el tipo de un puntero. Esto requiere extremada precaución puesto que no hay nada previsto para detectar si la conversión es correcta en tiempo de ejecución. En Java se puede hacer una comprobación en tiempo de ejecución de la compatibilidad de tipos y emitir una excepción cuando falla.

B. Comienzos

Java fue diseñado en 1990 por James Gosling, de Sun Microsystems, como software para dispositivos electrónicos de consumo. Curiosamente, todo este lenguaje fue diseñado antes de que diese comienzo la era World Wide Web, puesto que fue diseñado para dispositivos electrónicos como calculadoras, microondas y la televisión interactiva.

En los primeros años de la década de los noventa, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc., para dar margen de maniobra al equipo responsable del proyecto.

Inicialmente Java se llamó Oak (roble en inglés), aunque tuvo que cambiar de denominación, debido a que dicho nombre ya estaba registrado por otra empresa. Se dice este nombre se le puso debido a la existencia de tal árbol en los alrededores del lugar de trabajo de los promotores del lenguaje.

Tres de las principales razones que llevaron a crear Java son:

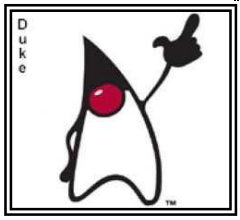
1. Creciente necesidad de interfaces mucho más cómodas e intuitivas que los sistemas de ventanas que proliferaban hasta el momento.
2. Fiabilidad del código y facilidad de desarrollo. Gosling observó que muchas de las características que ofrecían C o C++ aumentaban de forma alarmante el gran coste de pruebas y depuración. Por ello en los sus ratos libres creó un lenguaje de programación donde intentaba solucionar los fallos que encontraba en C++.
3. Enorme diversidad de controladores electrónicos. Los dispositivos electrónicos se controlan mediante la utilización de microprocesadores de bajo precio y reducidas prestaciones, que varían cada poco tiempo y que utilizan diversos conjuntos de instrucciones. Java permite escribir un código común para todos los dispositivos.

Por todo ello, en lugar de tratar únicamente de optimizar las técnicas de desarrollo y dar por sentada la utilización de C o C++, el equipo de Gosling se planteó que tal vez los lenguajes existentes eran demasiado complicados como para conseguir reducir de forma apreciable la complejidad de desarrollo asociada a ese campo. Por este motivo, su primera propuesta fue idear un nuevo lenguaje de programación lo más sencillo posible, con el objeto de que se pudiese adaptar con facilidad a cualquier entorno de ejecución.

Basándose en el conocimiento y estudio de gran cantidad de lenguajes, este grupo decidió recoger las características esenciales que debía tener un lenguaje de programación moderno y potente, pero eliminando todas aquellas funciones que no eran absolutamente imprescindibles.

C. Primeros proyectos en que se aplicó Java

El proyecto Green fue el primero en el que se aplicó Java, y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Con este fin se construyó un computador experimental denominado *7 (Star Seven). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía ya *Duke*, la actual mascota de Java.



Más tarde Java se aplicó a otro proyecto denominado VOD (Video On Demand) en el que se empleaba como interfaz para la televisión interactiva que se pensaba iba a ser el principal campo de aplicación de Java. Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en un Java primitivo.
Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, instaron a FirstPerson a desarrollar nuevas estrategias que produjeran beneficios. Entre ellas se encontraba la aplicación de Java a Internet, la cual no se consideró productiva en ese momento.

D. Resurgimiento de Java

Aunque muchas de las fuentes consultadas señalan que Java no llegó a caer en un olvido, lo cierto es que tuvo que ser Bill Joy (cofundador de Sun y uno de los desarrolladores principales del sistema operativo Unix de Berckley) el que sacó a Java del letargo en que estaba sumido. Joy juzgó que Internet podría llegar a ser el campo adecuado para disputar a Microsoft su primacía en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes.
Para poder presentarlo en sociedad se tuvo que modificar el nombre de este lenguaje de programación y se tuvo que realizar una serie de modificaciones de diseño para poderlo adaptar al propósito mencionado. Así Java fue presentado en sociedad en agosto de 1995.
Algunas de las razones que llevaron a Bill Joy a pensar que Java podría llegar a ser rentable son:

- Java es un lenguaje orientado a objetos: Esto es lo que facilita abordar la resolución de cualquier tipo de problema.
- Es un lenguaje sencillo, aunque sin duda potente.
- La ejecución del código Java es segura y fiable: Los programas no acceden directamente a la memoria del computador, siendo imposible que un programa escrito en Java pueda acceder a los recursos del computador sin que esta operación le sea permitida de forma explícita. De este modo, los datos del usuario quedan a salvo de la existencia de virus escritos en Java. La ejecución segura y controlada del código Java es una característica única, que no puede encontrarse en ninguna otra tecnología.
- Es totalmente multiplataforma: Es un lenguaje sencillo, por lo que el entorno necesario para su ejecución es de pequeño tamaño y puede adaptarse incluso al interior de un navegador.

E. Futuro de Java

Existen muchas críticas a Java debido a su lenta velocidad de ejecución, aproximadamente unas 20 veces más lento que un programa en lenguaje C. Sun está trabajando intensamente en crear versiones de Java con una velocidad mayor.
El problema fundamental de Java es que utiliza una representación intermedia denominada *código de byte* para solventar los problemas de portabilidad. Los *códigos de byte* posteriormente se tendrán que transformar en código máquina en cada máquina en que son utilizados, lo que enlentece considerablemente el proceso de ejecución.
La solución que se deriva de esto parece bastante obvia: fabricar computadores capaces de comprender directamente los códigos de byte. Éstas serían unas máquinas que utilizaran Java como sistema operativo y que no requerirían en principio de disco duro porque obtendrían sus recursos de la red.
A los computadores que utilizan Java como sistema operativo se les llama Network Computer, WebPC o WebTop. La primera gran empresa que ha apostado por este tipo de máquinas ha sido Oracle, que en enero de 1996 presentó en Japón su primer NC (Network Computer), basado en un procesador RISC con 8 Megabytes de RAM. Tras Oracle, han sido compañías del tamaño de Sun, Apple e IBM las que han anunciado desarrollos similares.
La principal empresa en el mundo del software, Microsoft, que en los comienzos de Java no estaba a favor de su utilización, ha licenciado Java, lo ha incluido en Internet Explorer (versión 3.0 y posteriores), y ha lanzado un entorno de desarrollo para Java, que se denomina Visual J++.

El único problema aparente es la seguridad para que Java se pueda utilizar para transacciones críticas. Sun va a apostar por firmas digitales, que serán clave en el desarrollo no sólo de Java, sino de Internet.

CARACTERÍSTICAS DE JAVA

A. Introducción

Se puede deci que Java supone un significativo avance en el mundo de los entornos de software, y esto viene avalado por tres elementos claves que diferencian a este lenguaje desde un punto de vista tecnológico:

- Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.
- Proporciona un conjunto de clases potente y flexible.
- Pone al alcance de cualquiera la utilización de aplicaciones que se pueden incluir directamente en páginas Web (aplicaciones denominadas *applets*).

Java aporta a la Web una interactividad que se había buscado durante mucho tiempo entre usuario y aplicación.

B. Potente

a.) Orientación a objetos
En este aspecto Java fue diseñado partiendo de cero, no siendo derivado de otro lenguaje anterior y no tiene compatibilidad con ninguno de ellos.
En Java el concepto de objeto resulta sencillo y fácil de ampliar. Además se conservan elementos "no objetos", como números, caracteres y otros tipos de datos simples.

b.) Riqueza semántica
Pese a su simpleza se ha conseguido un considerable potencial, y aunque cada tarea se puede realizar de un número reducido de formas, se ha conseguido un gran potencial de expresión e innovación desde el punto de vista del programador.

c.) Robusto
Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones, y así lo que es rigidez y falta de flexibilidad se convierte en eficacia.
Respecto a la gestión de memoria, Java libera al programador del compromiso de tener que controlar especialmente la asignación que de ésta hace a sus necesidades específicas. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura, y un manejo de excepciones orientado a objetos integrados. Estos elementos realizarán muchas tareas antes tediosas a la vez que obligadas para el programador.

d.) Modelo de objeto rico
Existen varias clases que contienen las abstracciones básicas para facilitar a los programas una gran capacidad de representación. Para ello se contará con un conjunto de clases comunes que pueden crecer para admitir todas las necesidades del programador.
Además la biblioteca de clases de Java proporciona un conjunto único de protocolos de Internet.
El conjunto de clases más complicado de Java son sus paquetes gráficos AWT (*Abstract Window Toolkit*) y *Swing*. Estos paquetes implementan componentes de una interfaz de usuario gráfica básica común a todos los computadores personales modernos.
NOTA: El AWT es tema de estudio en Programación II

C. Simple

a.) Fácil aprendizaje
El único requerimiento para aprender Java es tener una comprensión de los conceptos básicos de la programación orientada a objetos. Así se ha creado un lenguaje simple (aunque eficaz y expresivo) pudiendo mostrarse cualquier planteamiento por parte del programador sin que las interioridades del sistema subyacente sean desveladas.
Java es más complejo que un lenguaje simple, pero más sencillo que cualquier otro entorno de programación. El único obstáculo que se puede presentar es conseguir comprender la programación orientada a objetos, aspecto que, al ser independiente del lenguaje, se presenta como insalvable.

b.) Completado con utilidades
El paquete de utilidades de Java viene con un conjunto completo de estructuras de datos complejas y sus métodos asociados, que serán de ayuda para implementar *applets* y otras aplicaciones más complejas. Se dispone también de estructuras de datos habituales, como *pilas* y *tablas hash*, como clases ya implementadas.
El JDK (*Java Development Kit)* suministrado por Sun Microsystems incluye un compilador, un intérprete de aplicaciones, un depurador en línea de comandos, y un visualizador de *applets* entre otros elementos.
A su vez ECLIPSE contiene todos los elementos para escribir, compilar y ejecutar programas.

D. Interactivo y orientado a red

a.) Interactivo y animado
Uno de los requisitos de Java desde sus inicios fue la posibilidad de crear programas en red interactivos, por lo que es capaz de hacer varias cosas a la vez sin perder rastro de lo que debería suceder y cuándo. Para se da soporte a la utilización de múltiples hilos de programación (*multithread*).
Las aplicaciones de Java permiten situar figuras animadas en las páginas Web, y éstas pueden concebirse con logotipos animados o con texto que se desplace por la pantalla. También pueden tratarse gráficos generados por algún proceso. Estas animaciones pueden ser interactivas, permitiendo al usuario un control sobre su apariencia.

b.) Arquitectura neutral
Java está diseñado para que un programa escrito en este lenguaje sea ejecutado correctamente independientemente de la plataforma en la que se esté actuando (Macintosh, PC, UNIX…). Para conseguir esto utiliza una compilación en una representación intermedia que recibe el nombre de *códigos de byte*, que pueden interpretarse en cualquier sistema operativo con un intérprete de Java. La desventaja de un sistema de este tipo es el rendimiento; sin embargo, el hecho de que Java fuese diseñado para funcionar razonablemente bien en microprocesadores de escasa potencia, unido a la sencillez de traducción a código máquina hacen que Java supere esa desventaja sin problemas.

c.) Trabajo en red
Java anima las páginas Web y hace posible la incorporación de aplicaciones interactivas y especializadas. Aporta la posibilidad de distribuir contenidos ejecutables, de manera que los suministradores de información de la Web pueden crear una página de hipertexto (*página Web*) con una interacción continuada y compleja en tiempo real; el contenido ejecutable es transferido literalmente al computador del usuario.
Los protocolos básicos para trabajar en Internet están encapsulados en unas cuantas clases simples. Se incluyen implementaciones ampliables de los protocolos FTP, HTTP, NNTP y SMTP junto con conectores de red de bajo nivel e interfaces de nombrado. Esto le permite interactuar con esos servicios de red poderosos sin tener que comprender realmente los detalles de bajo nivel de esos protocolos. Este lenguaje está diseñado para cumplir los requisitos de entrega de contenidos interactivos mediante el uso de applets insertados en sus páginas HTML. Además, las clases de Java admiten muy bien estos protocolos y formatos. El envío de las clases de Java a través de Internet se realiza con gran facilidad, ya que existe una interfaz unificada, resolviendo así los típicos problemas de diferencia de versiones.

Java proporciona un conjunto de clases para tratar con una abstracción de los conectores de red (*sockets)* originales de la versión UNIX de Berckley, encapsular la noción de una dirección de Internet o conectar sockets con flujos de datos de Entrada/Salida.
Con todas estas posibilidades aumenta el dinamismo y competitividad de la Web, puesto que es capaz de captar el interés del usuario durante largo tiempo y permite a los programadores convertir la Web en un sistema de entrega de software.

d.) Applets
Una *applet* (miniaplicación) es un pequeño programa en Java transferido dinámicamente a través de Internet. Presentan un comportamiento inteligente, pudiendo reaccionar a la entrada de un usuario y cambiar de forma dinámica. Sin embargo, la verdadera novedad es el gran potencial que Java proporciona en este aspecto, haciendo posible que los programadores ejerzan un control sobre los programas ejecutables de Java que no es posible encontrar en otros lenguajes.

E. Aún más

a.) Seguridad
Existe una preocupación lógica en Internet por el tema de la seguridad: virus, caballos de Troya, y programas similares navegan de forma usual por la red, constituyendo una amenaza palpable. Java ha sido diseñado poniendo un énfasis especial en el tema de la seguridad, y se ha conseguido lograr cierta inmunidad en el aspecto de que un programa realizado en Java no puede realizar llamadas a funciones globales ni acceder a recursos arbitrarios del sistema, por lo que el control sobre los programas ejecutables no es equiparable a otros lenguajes.
Los niveles de seguridad que presenta son:
- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de los *códigos de byte* que asegura que no se viole ninguna construcción del lenguaje.
- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles.

b.) Lenguaje basado en C++
Java fue desarrollado basándose en C++, pero eliminando rasgos del mismo poco empleados, optándose por una codificación comprensible. Básicamente, encontramos las siguientes diferencias con C++:
- Java no soporta los tipos *struct, union* ni punteros.
- No soporta *typedef* ni *#define*.
- Se distingue por su forma de manejar ciertos operadores y no permite una sobrecarga de operadores.
- No soporta herencia múltiple.
- Java maneja argumentos en la línea de comandos de forma diversa a como lo hacen C o C++.
- Tiene una clase *String* que es parte del paquete *java.lang* y se diferencia de la matriz de caracteres terminada con un nulo que usan C y C++.
- Java cuenta con un sistema automático para asignar y liberar memoria, con lo que no es necesario utilizar las funciones previstas con este fin en C y C++.

c.) Gestión de la Entrada/Salida
En lugar de utilizar primitivas como las de C para trabajar con archivos, se utlizan primitivas similares a las de C++, mucho más elegantes, que permiten tratar los archivos, sockets, teclado y monitor como flujos de datos.
De este modo se pueden utilizar dichas primitivas para cualquier operación de Entrada/Salida.

d.) Diferentes tipos de aplicaciones

En Java podemos crear los siguientes tipos de aplicaciones:

- Aplicaciones: Se ejecutan sin necesidad de un navegador.
- *Applets*: Se pueden descargar de Internet y se observan en un navegador.
- *JavaBeans*: Componentes software Java, que se puedan incorporar gráficamente a otros componentes.
- *JavaScript*: Conjunto del lenguaje Java que puede codificarse directamente sobre cualquier documento HTML
- *Servlets:* Módulos que permiten sustituir o utilizar el lenguaje Java en lugar de programas CGI (Common Gateway Interface) a la hora de dotar de interactividad a las páginas Web.

VERSIONES Y DISTRIBUCIONES DE JAVA

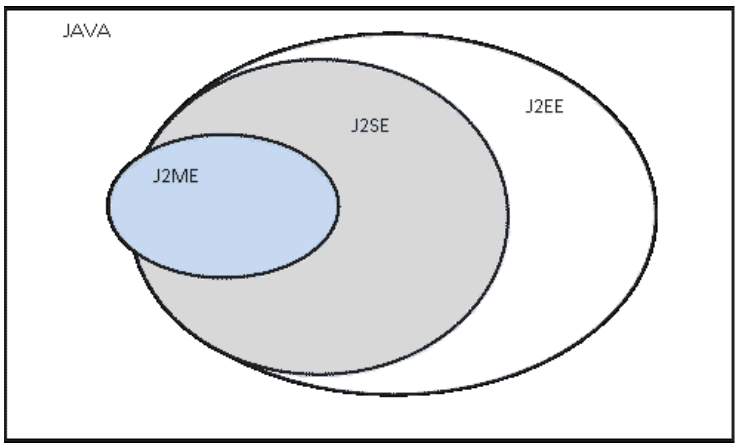
Java, como la mayoría de los lenguajes, ha sufrido diversos cambios a lo largo de su historia. Además, en cada momento han coexistido distintas versiones o distribuciones de Java con distintos fines. Actualmente puede considerarse que el Java vigente se denomina Java 2 y existen 3 distribuciones principales de Java 2, con ciertos aspectos comunes y ciertos aspectos divergentes.

Estas tres distribuciones son:

a) J2SE o simplemente Java SE: Java 2 Standard Edition o Java Standard Edition. Orientado al desarrollo de aplicaciones cliente / servidor. No incluye soporte a tecnologías para internet. Es la base para las otras distribuciones Java y es la plataforma que utilizaremos en este curso por ser la más utilizada.

b) J2EE: Java 2 Enterprise Edition. Orientado a empresas y a la integración entre sistemas. Incluye soporte a tecnologías para internet. Su base es J2SE.

c) J2ME: Java 2 Micro Edition. Orientado a pequeños dispositivos móviles (teléfonos, tablet, etc.).



En esta imagen vemos, de forma orientativa, como J2EE "expande" a J2SE, mientras que J2ME "recorta" a J2SE al tiempo que tiene una fracción de contenido diferenciada exclusiva de J2ME. En realidad hablar de expansiones y recortes no es correcto, porque cada distribución es en sí misma distinta puesto que están concebidas con distintas finalidades. Por tanto no puede decirse que sean expansiones o recortes, pero de forma coloquial muchas veces se interpreta así.

Java hoy en día es más que un lenguaje de programación. El lenguaje Java estándar ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de recursos disponibles para los programadores Java. Podemos citar en la evolución del Java estándar:

- JDK 1.0 (1996): primer lanzamiento del lenguaje Java.
- JDK 1.1 (1997): mejora de la versión anterior.
- J2SE 1.2 (1998): ésta y las siguientes versiones fueron recogidas bajo la denominación Java 2 y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Incluyó distintas mejoras.
- J2SE 1.3 (2000): mejora de la versión anterior.

- J2SE 1.4 (2002): mejora de la versión anterior.
- J2SE 5.0 (2004): originalmente numerada 1.5, esta notación aún es usada en ocasiones. Mejora de la versión anterior.
- Java SE 6 (2006): en esta versión, Sun cambió el nombre "J2SE" por Java SE y eliminó el ".0" del número de versión. Mejora de la versión anterior.
- Java SE 7 (2011): nueva versión que mejora la anterior.
- Java SE 8: nueva versión que se considera pueda hacer aparición dentro de varios años.

En Java todas las versiones siguen los mismos estándares de datos, esto permite que un programa que hayamos hecho con una versión antigua, pueda ser ejecutado con una versión más nueva sin necesidad de ningún cambio.

Además de los cambios en el lenguaje en sí, con el paso de los años los recursos disponibles para los programadores Java que ofrece la empresa que desarrolla el lenguaje (antiguamente Sun MicroSystems, actualmente Oracle) han crecido enormemente. La denominada "biblioteca de clases de Java" (Java class library) ha pasado de ofrecer unos pocos cientos de clases en JDK 1.0 hasta cerca de 4000 en Java SE 6. Se han introducido recursos completamente nuevos, como Swing y Java2D, mientras que muchos de los métodos y clases originales de JDK 1.0 han dejado de utilizarse.

Cuando trabajamos con Java será frecuente que busquemos información "oficial" en internet. Cuando decimos oficial nos referimos a la que ofrece la propia empresa desarrolladora de Java. Cuando buscamos información sobre Java hay que tener cuidado respecto a qué versión hace alusión la información. Por ejemplo, prueba a buscar "ArrayList java" o "ArrayList api java" en google, yahoo, bing o cualquier otro buscador y encontrará resultados de diferentes versiones.

En este curso trabajaremos con Java Platform SE 6 (Standard Edition) o Java SE 7 por ser las versiones más usadas hoy en día: si miramos la documentación correspondiente a versiones anteriores podemos confundirnos. Los ejemplos que mostramos en el curso son de Java SE 6. Por tanto una búsqueda más correcta sería "ArrayList api java 6", y en todo caso estar atentos a la especificación de la documentación para comprobar que efectivamente se corresponde con la versión con la que estemos trabajando. Si quieres utilizar otra versión Java no hay problema. Los cambios entre versiones no suelen ser tan importantes como para afectar a una persona que aprende el lenguaje por primera vez: en realidad nos daría igual usar una versión u otra. Sin embargo, hay que tener claro qué versión es la que usamos.

INFORMACION PARA EL ALUMNO (puede ser complementada con la anterior u otra):

EL LENGUAJE JAVA

El lenguaje Java es a la vez compilado e interpretado.
Con el compilador se convierte el código fuente que reside en archivos cuya extensión es .java, a un conjunto de instrucciones que recibe el nombre de bytecodes que se guardan en un archivo cuya extensión es .class.
Estas instrucciones son independientes del tipo de computador.
El intérprete ejecuta cada una de estas instrucciones en un computador específico (Windows, Macintosh, etc). Solamente es necesario, por tanto, compilar una vez el programa, pero se interpreta cada vez que se ejecuta en un computador.

Cada intérprete Java es una implementación de la Máquina Virtual Java (JVM). Los bytecodes posibilitan el objetivo de "write once, run anywhere", de escribir el programa una vez y que se pueda correr en cualquier plataforma que disponga de una implementación de la JVM. Por ejemplo, el mismo programa Java puede correr en Windows 98, Solaris, Macintosh, etc.

Java es, por tanto, algo más que un lenguaje, ya que la palabra Java se refiere a dos cosas inseparables: el lenguaje que nos sirve para crear programas y la Máquina Virtual Java que sirve para ejecutarlos. El API de Java y la Máquina Virtual Java forman una capa intermedia

(Java platform) que aísla el programa Java de las especificidades del hardware (hardware-based platform).

La Máquina Virtual Java

La Máquina Virtual Java (JVM) es el entorno en el que se ejecutan los programas Java, su misión principal es la de garantizar la portabilidad de las aplicaciones Java. Define esencialmente un computador abstracto y especifica las instrucciones (bytecodes) que este computador puede ejecutar. El intérprete Java específico ejecuta las instrucciones que se guardan en los archivos cuya extensión es .class.
Las tareas principales de la JVM son las siguientes:

- Reservar espacio en memoria para los objetos creados
- Liberar la memoria no usada (garbage collection).
- Asignar variables a registros y pilas
- Llamar al sistema huésped para ciertas funciones, como los accesos a los dispositivos
- Vigilar el cumplimiento de las normas de seguridad de las aplicaciones Java

Esta última tarea, es una de las más importantes que realiza la JVM. Además, las propias especificaciones del lenguaje Java contribuyen extraordinariamente a este objetivo:

- Las referencias a arrays son verificadas en el momento de la ejecución del programa
- No hay manera de manipular de forma directa los punteros
- La JVM gestiona automáticamente el uso de la memoria, de modo que no queden huecos.
- No se permiten realizar ciertas conversiones (casting) entre distintos tipos de datos.

Por ejemplo, cuando el navegador encuentra una página web con un applet, pone en marcha la JVM y proporciona la información necesaria. El cargador de clases dentro de la JVM ve que clases necesita el applet. Dentro del proceso de carga, las clases se examinan mediante un verificador que asegura que las clases contienen código válido y no malicioso. Finalmente, se ejecuta el applet.

El lenguaje Java

El lenguaje Java no está diseñado solamente para crear applets que corren en la ventana del navegador. Java es un lenguaje de propósito general, de alto nivel, y orientado a objetos.

Java es un lenguaje de programación orientado a objetos puro, en el sentido de que no hay ninguna variable, función o constante que no esté dentro de una clase. Se accede a los miembros dato y las funciones miembro a través de los objetos y de las clases. Por razones de eficiencia, se han conservado los tipos básicos de datos, int, float, double, char, etc, similares a los del lenguaje C/C++.

Los tipos de programas más comunes que se pueden hacer con Java son los applets (se ejecutan en el navegador de la máquina cliente) y las aplicaciones (programas que se ejecutan directamente en la JVM). Otro tipo especial de programa se denomina servlet que es similar a los applets pero se ejecutan en los servidores Java.

La API de Java es muy rica, está formada un conjunto de paquetes de clases que le proporcionan una gran funcionalidad. El núcleo de la API viene con cada una de las implementaciones de la JVM:
Lo esencial: tipos de datos, clases y objetos, arrays, cadenas de caracteres (strings), subprocesos (threads), entrada/salida, propiedades del sistema, etc.

Java proporciona también extensiones, por ejemplo define un API para 3D, para los servidores, telefonía, reconocimiento de voz, etc.

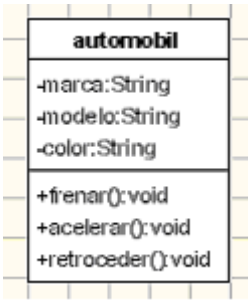
Conceptos Básicos de Objetos

El concepto de Objeto se debe analizar en forma simple (dejando para segundo año una profundización del mismo).

1. Objetos

Entender que es un objeto es la clave para entender cualquier lenguaje orientado a objetos.
Existen muchas definiciones que se le ha dado al Objeto. Primero empecemos entendiendo que es un objeto del mundo real.
Un objeto del mundo real es cualquier cosa que vemos a nuestro alrededor. Digamos que para leer este artículo lo hacemos a través del monitor y una computadora, ambos son objetos, al igual que nuestro teléfono celular, un árbol o un automóvil.
No necesitamos ser expertos en hardware para saber que una computadora está compuesta internamente por varios componentes: la tarjeta madre, el chip del procesador, un disco duro, una tarjeta de video, y otras partes más. El trabajo en conjunto de todos estos componentes hace operar a una computadora.
Internamente, cada uno de estos componentes puede ser sumamente complicado y puede ser fabricado por diversas compañías con diversos métodos de diseño. Pero nosotros no necesitamos saber cómo trabajan cada uno de estos componentes, como saber que hace cada uno de los chips de la tarjeta madre, o cómo funciona internamente el procesador. Cada componente es una unidad autónoma, y todo lo que necesitamos saber de adentro es cómo interactúan entre sí los componentes, saber por ejemplo si el procesador y las memorias son compatibles con la tarjeta madre, o conocer donde se coloca la tarjeta de video. Cuando conocemos como interaccionan los componentes entre sí, podremos armar fácilmente una computadora.

¿Que tiene que ver esto con la programación? La programación orientada a objetos trabaja de esta manera. Todo el programa está construido en base a diferentes componentes (Objetos), cada uno tiene un rol específico en el programa y todos los componentes pueden comunicarse entre ellos de formas predefinidas.
Todo objeto del mundo real tiene 2 componentes: características y comportamiento.
Por ejemplo, los automóviles tienen características (marca, modelo, color, velocidad máxima, etc.) y comportamiento (frenar, acelerar, retroceder, llenar combustible, cambiar llantas, etc.).
Los Objetos de Software, al igual que los objetos del mundo real, también tienen características y comportamientos. Un objeto de software mantiene sus características en una o más "variables", e implementa su comportamiento con "métodos". Un método es una función o subrutina asociada a un objeto.



Para redondear estas ideas, imaginemos que hay estacionado AUDI A4 color negro que corre hasta 260 km/h. Si pasamos ese objeto del mundo real al mundo del

software, tendremos un objeto Automóvil con sus características predeterminadas:

Marca	=	Audi
Modelo	=	A4
Color	=	Negro
Velocidad Máxima	=	260 km/h

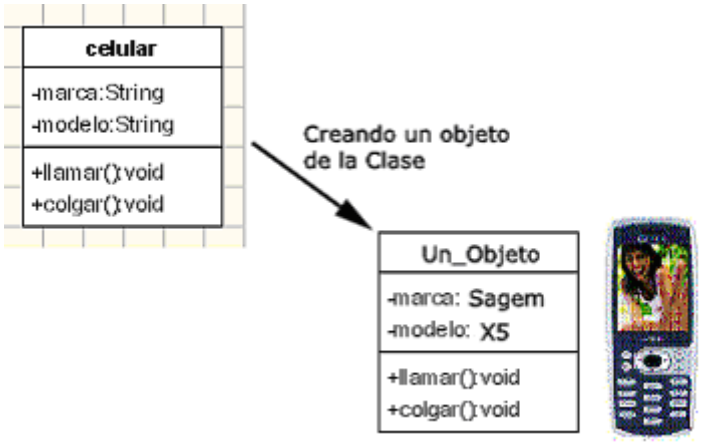
Cuando a las características del objeto le ponemos valores decimos que el objeto tiene estados.

Las variables almacenan los estados de un objeto en un determinado momento.

Definición teórica: Un objeto es una unidad de código compuesto de variables y métodos relacionados.

2. Las Clases

En el mundo real, normalmente tenemos muchos objetos del mismo tipo. Por ejemplo, nuestro teléfono celular es sólo uno de los miles que hay en el mundo. Si hablamos en términos de la programación orientada a objetos, podemos decir que nuestro objeto celular es una instancia de una clase conocida como "celular". Los celulares tienen características (marca, modelo, sistema operativo, pantalla, teclado, etc.) y comportamientos (hacer y recibir llamadas, enviar mensajes multimedia, transmisión de datos, etc.).



Cuando se fabrican los celulares, los fabricantes aprovechan el hecho de que los celulares comparten esas características comunes y construyen modelos o plantillas comunes, para que a partir de esas se puedan crear muchos equipos celulares del mismo modelo. A ese modelo o plantilla le llamamos CLASE, y a los equipos que sacamos a partir de ella la llamamos OBJETOS .
Esto mismo se aplica a los objetos de software, se puede tener muchos objetos del mismo tipo y mismas características.

Definición teórica: La clase es un modelo o prototipo que define las variables y métodos comunes a todos los objetos de cierta clase. También se puede decir que una clase es una plantilla genérica para un conjunto de objetos de similares características.

Por otro lado, una instancia de una clase es otra forma de llamar a un objeto. En realidad no existe diferencia entre un objeto y una instancia. Sólo que el objeto es un término más general, pero los objetos y las instancias son ambas representación de una clase.

Definición Teórica: Una instancia es un objeto de una clase en particular.

Eclipse

Para trabajar cómodamente en Java necesitamos una IDE (INTERFAZ DE DESARROLLO INTEGRADO).
O sea un programa que me permita escribir el programa, guardarlo, compilarlo y ejecutarlo.
Existen varios programas populares que realizan esta tarea: recomendamos el uso de ECLIPSE.
ECLIPSE es un programa gratuito que se descarga de Internet de:

http://www.eclipse.org/downloads/

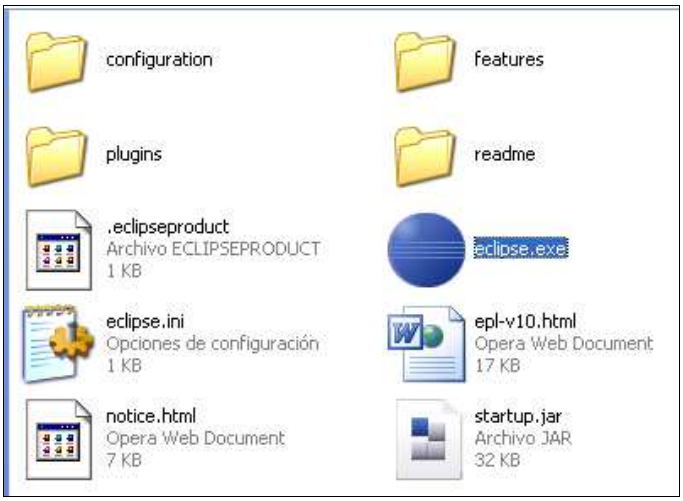
Se recomienda descargar desde ahí Eclipse Classic (174 Mb):

Es un archivo comprimido con .ZIP que se puede descomprimir en el Escritorio o en Disco Duro, buscando un lugar de cómodo acceso a futuro.
Como no tiene instalador, se ejecuta directamente de dicha carpeta.

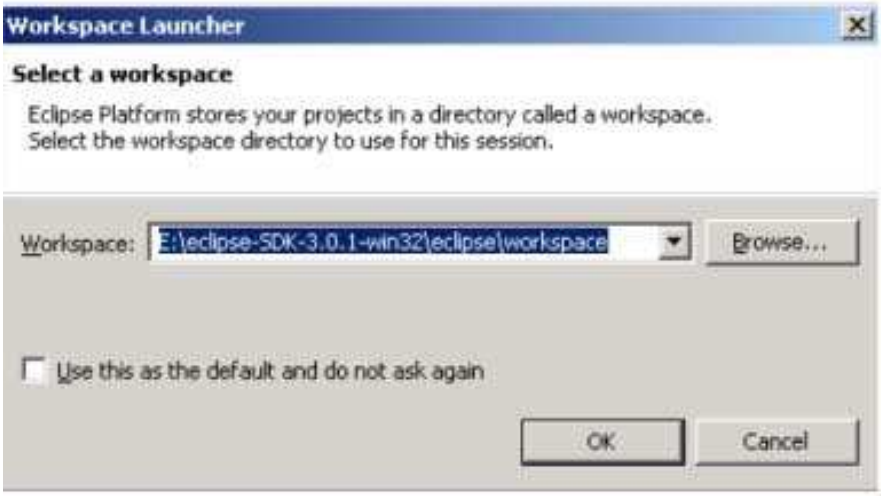
Usando Eclipse.

Como hemos dicho, un IDE puede hacernos el trabajo mucho más sencillo, sobretodo si nuestro desarrollo ya va manejando un buen número de Clases. Además estos entornos nos permiten mucha más versatilidad para depurar nuestros programas puesto que tienen debbugers mucho más avanzados, cosa que nos ayuda a buscar y corregir errores.
Eclipse es un IDE de código abierto. Hay más herramientas similares de código abierto disponibles pero ésta tiene la mejor relación calidad-facilidad.

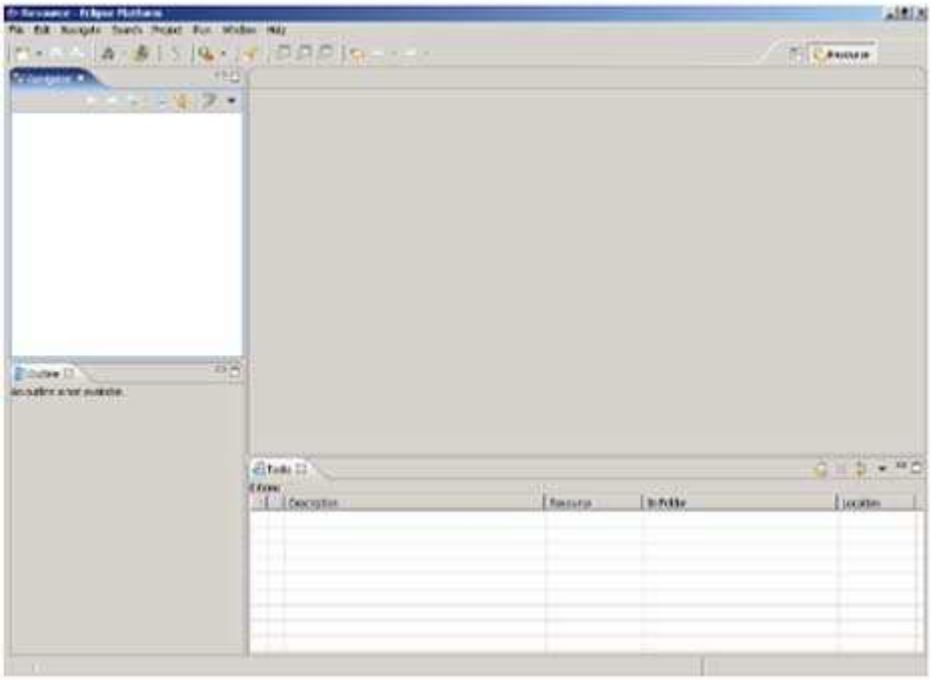
Se descarga de www.eclipse.org en forma de archivo ZIP y solo tenemos que descomprimirlo en la carpeta donde queramos tenerlo instalado. Para ejecutarlo solo hay que arrancar el fichero Eclipse.exe



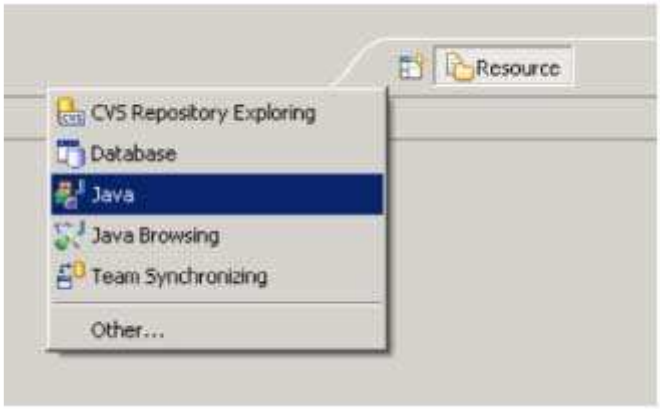
Una vez arrancado lo único que nos pedirá es que le demos la ruta por defecto donde queramos que eclipse nos vaya guardando los proyectos que creemos:

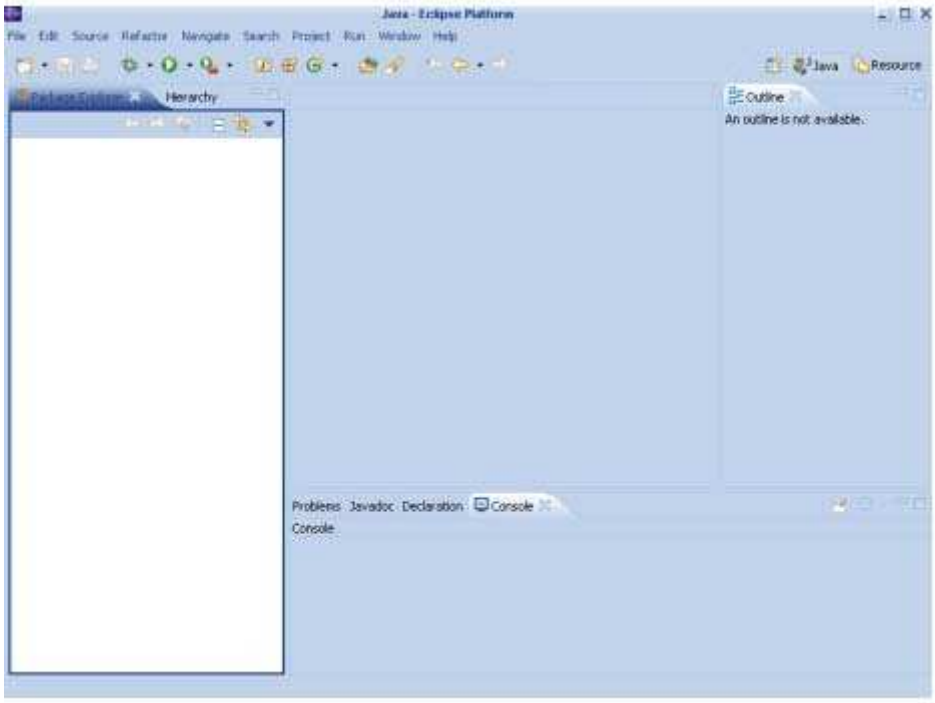


Después de esto nos aparecerá la ventana principal de Eclipse:



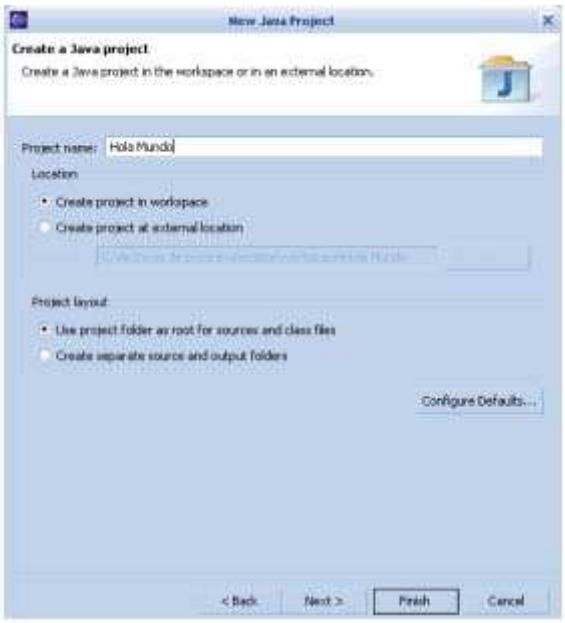
Eclipse puede usar varias perspectivas en su ventana principal dependiendo del tipo de desarrollo que vayamos a realizar. Abriremos la perspectiva "Java":





A continuación hay que crear un un proyecto para nuestro desarrollo:

Para esto entramos en "File/New Proyect". Elegimos "Java proyect" y le damos a siguiente. En la ventana en la que estamos ahora podemos darle un nombre a nuestro proyecto y nos aparecen dos opciones relativas a la organización de nuestro proyecto. Las dejamos tal y como está para que simplemente nos coloque nuestros archivos .java y .class (fuentes y ejecutables java) en la carpeta que hemos escogido para el entorno de trabajo al arrancar eclipse, y le damos "Finish".



El siguiente paso es ir añadiendo nuestras clases al proyecto.

Pinchando con el botón derecho en la carpeta del proyecto que se nos ha creado en la parte izquierda de la ventana principal podemos ir a "New Class"

Esta ventana tiene varias opciones que iremos entendiendo poco a poco. Por ahora simplemente pondremos los nombres de la clase, del paquete donde queramos incluirla (podemos dejar el paquete por defecto dejando este campo en blanco) y marcaremos las opciones que vemos en la ilustración.

Al escribir los nombres de la clase y del paquete nos avisa de ciertas reglas

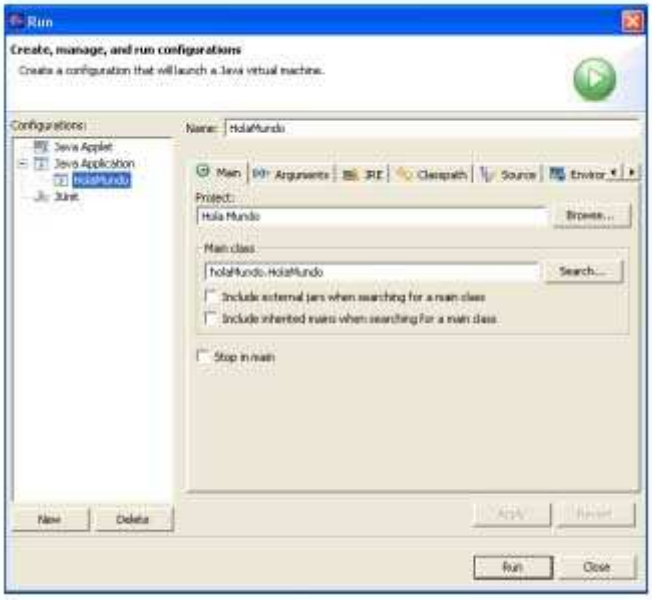
para la nomenclatura de estos. Los nombres de las clases siempre empiezan en mayúscula y los paquetes en minúscula.

Al darle a finalizar nos crea una plantilla que podemos ver en el centro de la pantalla. Esta área es la que corresponde al editor y aquí es donde escribiremos nuestros programas en Java. La plantilla creada nos añade las líneas básicas en el tipo de clase Java que hemos creado con todos los pasos anteriores.

A continuación se escribe el programa que queremos probar.

Nos queda ejecutar el programa para ver que funciona. Para hacerlo funcionar podemos utilizar el menú "run" o directamente mediante los iconos de la barra de herramientas.

Al ejecutar el "run" un asistente nos dará a elegir el tipo de ejecución que queremos para nuestro código en Java. Elegimos "Java Application" en el menú con un doble "clic" y nos creará un "apéndice" de configuración de ejecución para nuestro código como el siguiente:



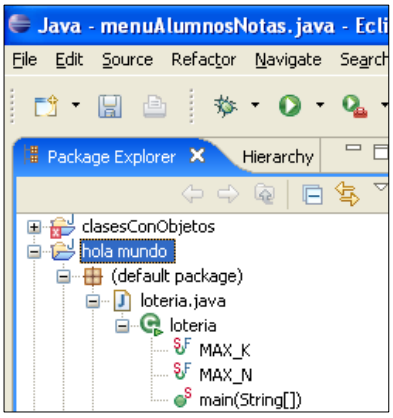
En principio y sin más detalles le damos a "Run" y vemos los resultados:

Abajo hay una pestaña llamada "console". Esa pestaña nos muestra una consola que hace las veces de la línea de comandos desde la que ejecutamos nuestro programa de Java en el primer ejemplo

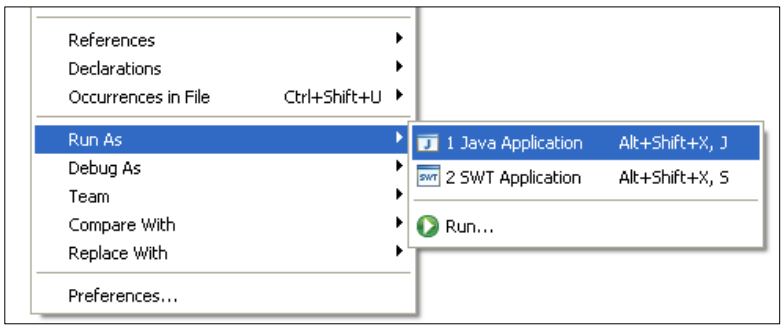
Jerarquía de Eclipse:

En la ventana de la izquierda de Eclipse puede observar la jerarquía del IDE:

- Eclipse puede tener n Proyectos (Ej Hola Mundo)
- Cada proyecto puede tener n librerias (Ej: (default Package))
- Cada libreria puede tener n programas java (Ej: loteria.java)
- Cada programa puede tener n clases (Ej: loteria)
- Cada clase puede tener n métodos (Ej: main())



NOTA: Cuando esta jerarquía le genere problemas para ejecutar un programa (Control+F11) puede pulsar botón derecho y elegir Run As/Java Application



JDK

El Eclipse suele traer casi todo lo necesario para programar en Java, pero te puede pasar que el compilador de Java no funcione, entonces es necesario instalar el JDK (Java Development Kit) y ajustar las variables de entorno del Sistema Operativo.

El JDK se descarga gratuitamente de http://www.oracle.com/technetwork/java/javase/downloads/index.html

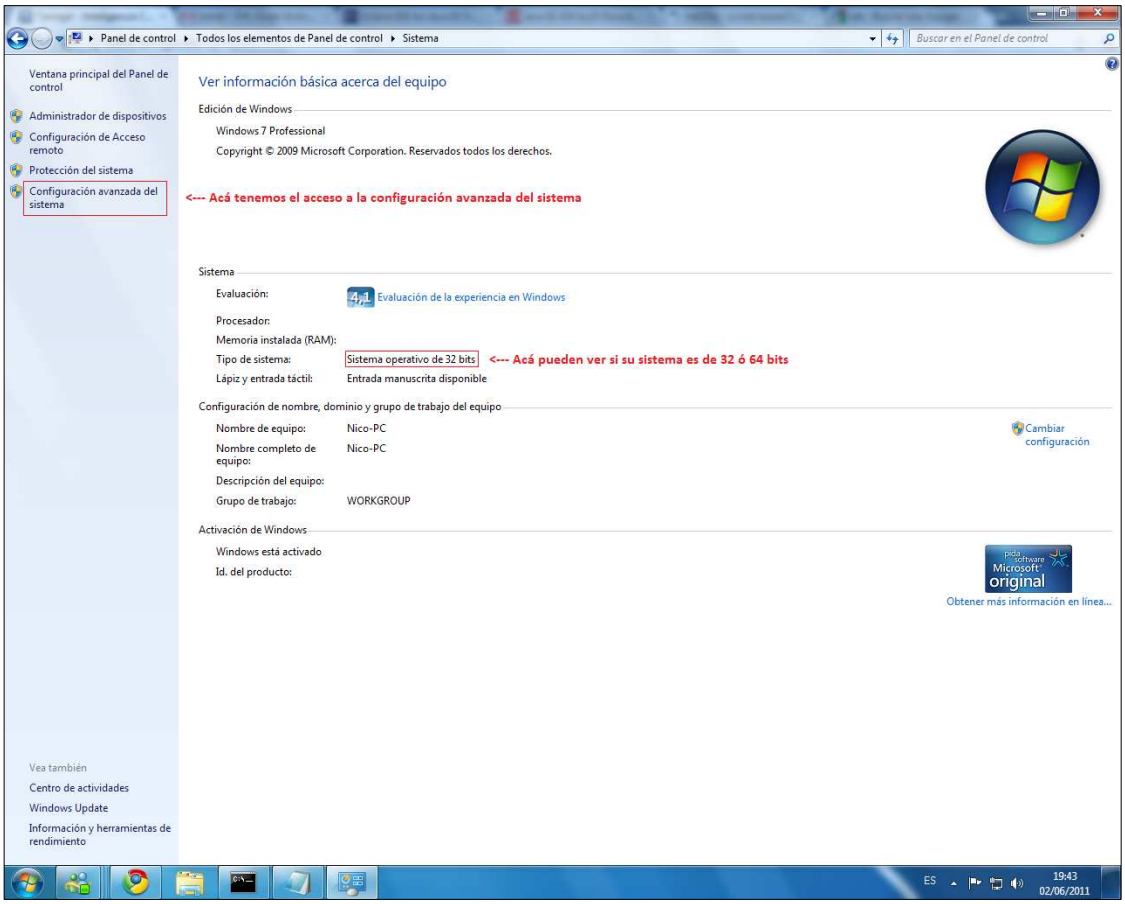
Luego de descargado se instala y queda pronto.

Variables de Entorno

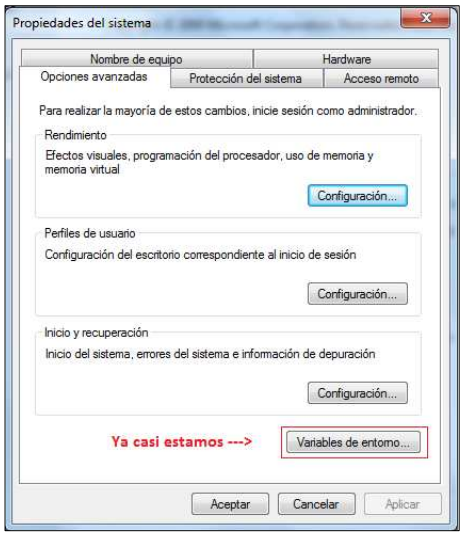
Este paso es opcional, y sirve únicamente para que desde la consola de Windows se puedan compilar y ejecutar archivos java.

Para agregar una variable de entorno se debe ir (en **Windows Seven**) a Panel de control > Sistema > Configuración avanzada del sistema > Variables de entorno

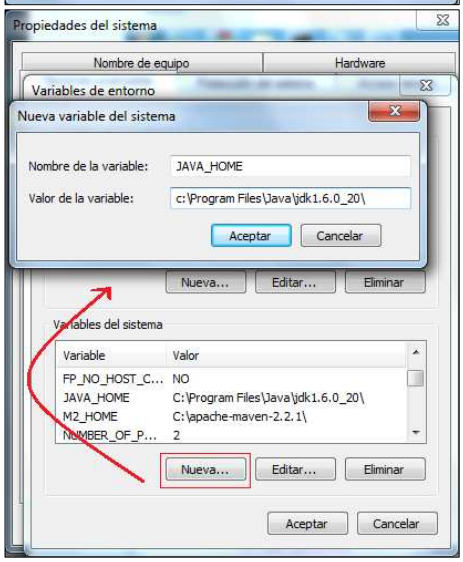
La pantalla de **sistema** es la siguiente:



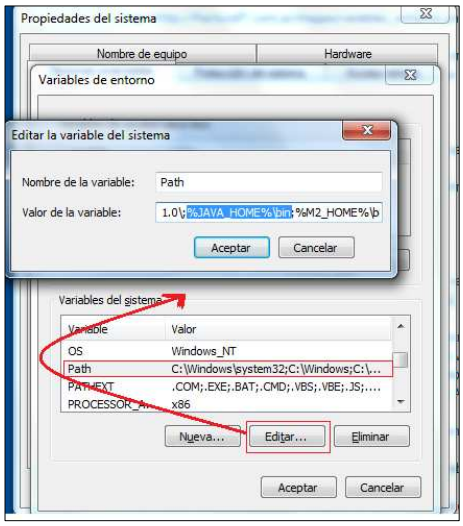
Al acceder a la **Configuración avanzada del sistema** vemos la siguiente pantalla:



En este paso vamos a crear una **nueva variable de entorno** presionando en el botón "Nueva...". Como muestra la imagen nos sale una nueva ventanita donde especificamos el nombre de la variable y el valor de la variable.
En el nombre pueden poner lo que quieran, es una simple referencia que utilizaremos luego (les recomiendo "JAVA_HOME"). El valor deberá ser sí o sí la **ruta de acceso a la carpeta donde instalaron el JDK** en su sistema. Les debería quedar algo similar a la imagen.
Al presionar aceptar verán la nueva variable dentro de la lista de "Variables del Sistema" tal cuál figura en la imagen.

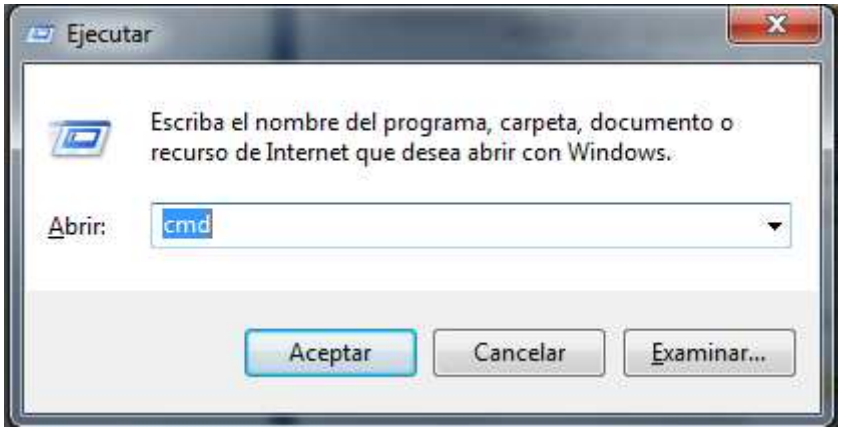


Ahora tenemos que **modificar la variable de sistema ya existente "Path"**. Para ello seleccionamos la variable de la lista de variables y presionamos el botón "Editar...". Esto abrirá una ventana emergente donde podremos ver el valor de la variable y editarlo. Vamos al final del todo de la variable y agregamos lo siguiente "**;%JAVA_HOME%bin;**", siendo "JAVA_HOME" el nombre de la variable que agregaron anteriormente.
Presionan aceptar en ambas pantallas y terminaron.



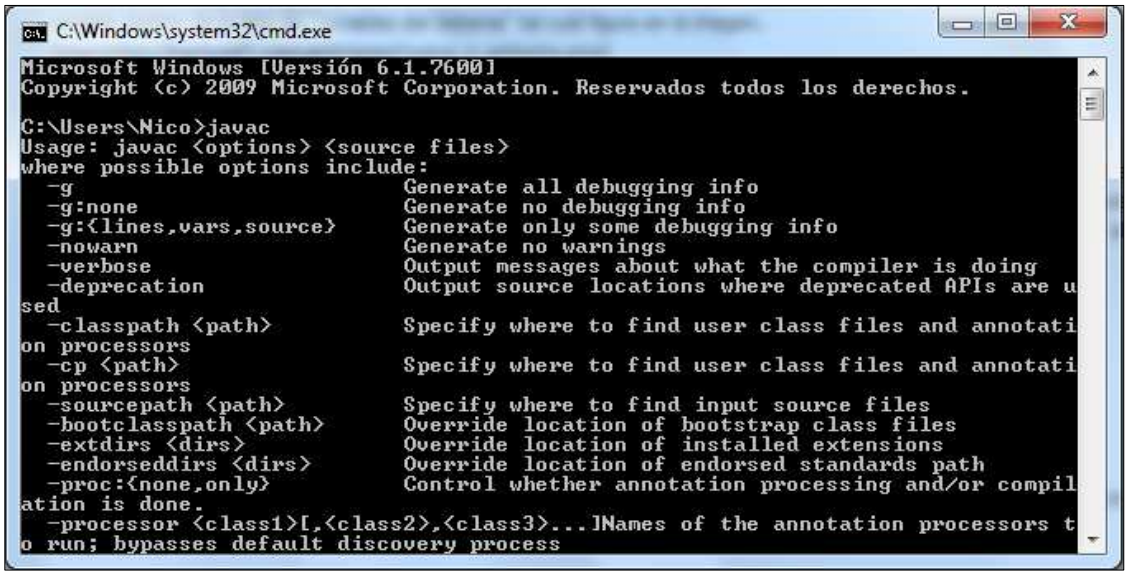
Para probar que la variable de entorno haya sido creada satisfactoriamente vamos a realizar lo siguiente:

Abrimos la consola DOS de windows (es importante abrirla después de cambiar las variables para que el cambio se vea reflejado en la instancia abierta de la consola)
Presionamos **Tecla Windows + R** escribimos "**cmd**" y **enter**



Una vez que tenemos la consola abierta sin importar en que directorio estemos parados escribimos el siguiente comando "**javac**" (o "**java**") y presionamos Enter.

Si te aparece la siguiente pantalla, indica que quedó bien. Sino hay que repetir los pasos anteriores nuevamente, observando los detalles de los mismos.



Tema 4: Primer Programa en Java

Este programa mostrará aspectos básicos de la programación en Java como son la creación de una clase, el método main, y la salida por consola.

```
1.  // primer programa en java
2.  public class GuiaJava {
3.          public static void main(String args []){
4.                  System.out.println("Hola Mundo desde Java");
5.          }
6.  }
```

La línea 1 de este ejemplo muestra el uso de comentarios de una sola línea.
Los comentarios no se compilan, y su función consiste en documentar lo que se está haciendo para futuras consultas.
También es posible el uso de comentarios que abarquen más de una línea, la forma de realizarlo es la siguiente:

```
/* esto es un comentario
de mas de una linea*/
```

La línea 2: public class GuiaJava
Comienza la declaración de la clase GuiaJava. Todo programa en Java contiene al menos una declaración de clases.
La palabra reservada class es utilizada para indicar la creación de la clase con el nombre que se indique a continuación. En este caso el nombre de la clase será GuiaJava.
La palabra public también es reservada e indica que la clase podrá ser accedida e instanciada desde cualquier otra clase. Más adelante se analizará con mayor profundidad.
El carácter "{" indica el comienzo de la implementación de la clase.

La línea 3: public static void main(String args [])
Los paréntesis luego de la palabra main indican la declaración de un método. Una clase puede contener uno o más métodos. Por otra parte una aplicación Java debería de contener un único método denominado main y su declaración se realiza como se indica en la linea 3. Al igual que en la implementación de una clase el carácter utilizado para indicar el comienzo de la misma es "{"

La linea 4: System.out.println("Hola Mundo desde java");
Acá se esta invocando el método necesario para mostrar un mensaje por consola. El texto entre paréntesis es el parámetro del método, y se mostrará tal cual se especifique entre comillas.
Al igual que en C se puede indicar un salto de línea con el carácter de escape "\n".

Las líneas 5 y 6:
Con el carácter "}" se indica el fin de la implementación de la clase y del método.

Ejercicio 5
Escriba y pruebe el programa Hola Mundo

Ejercicio 6

Modifique el ejercicio anterior usando el salto de línea, para mostrar el nombre del alumno en una línea y el apellido en otra.

Recuerde:

1. Use System.out.println() para escribir un texto en pantalla
2. Todo método de Java lleva paréntesis
3. Toda sentencia termina en punto y coma
4. \n hace que el cursor salte al siguiente renglón.
5. Java es CASE SENSITIVE. Distingue mayúsculas de minúsculas.

Solución al Ejercicio 7:

```
public class Guia_Java {
        public static void main(String args []){
                System.out.println("Juan \nPérez");
        }

}
```

Ejercicio 8

Modifique el programa anterior para que escriba el nombre de 2 personas en dos líneas diferentes de la pantalla.

Tema 5: Variables

Conceptos básicos

Una variable es un lugar de memoria, destinado a almacenar información, al cual se le asigna un nombre.
Una variable solo puede guardar un dato por vez, el cual puede ser usado en el programa tantas veces como se desee.
Una variable se puede representar como un casillero en la memoria que guarda un valor
Toda variable tiene un nombre, un tipo y un valor

Nombre de la variable

El nombre de la variable debe de cumplir ciertas reglas:

- Deben empezar con una letra del alfabeto ingles o guión bajo.
- Los siguientes caracteres pueden ser dígitos, letras o guión bajo.
- No se deben usar palabras reservadas, estas son aquellas propias del lenguaje Java como:
    - public, void, static, boolean, float, int, class, true, false, null, for, if, else, extends …
- El estándar de java sugiere utilizar comenzar en minúscula y indicar cambio de palabra con mayúscula ej:
    - nuevaVariable
    - cantPersonas
    - unNombreDeVariableMuyLargo

Ejercicio 9

Copie los siguientes nombres de variables e indique al lado si son válidos o no.

Nombre	S/N	Nombre	S/N	Nombre	S/N
costoItem		costo_Item		costoitem	
costoItem2		_costoitem		costo#item	
costoItemFacturado		costoItemfacturado		Class	
2012año		Año2012		anio2012	

Tipos de datos

Es el tipo de información que puede almacenar una variable.
Java dispone nativamente de 8 tipos de datos los cuales los podemos agrupar en 4 categorías:

Datos enteros:
Se usan para representar números enteros con signo. Hay cuatro tipos: byte, short, int y long.

Tipo	Tamaño	Valores
Byte	1Byte (8 bits)	-128 a 127
Short	2 Bytes (16 bits)	-32,768 a 32,767
Int	4 Bytes (32 bits)	-2,147,483,648 a 2,147,483,647
Long	8 Bytes (64 bits)	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807

Datos en coma flotante:
Se utilizan para representar números reales. Existen dos tipos float y double el primero utilizado para representar los números en precisión simple y el segundo en precisión doble.

Tipo	Tamaño	Valores
float	4 Bytes (32 bits)	+/- 3.4E+38F (6-7 dígitos importantes)
double	8 Bytes (64 bits)	+/- 1.8E+308 (15 dígitos importantes)

Datos caracteres
El tipo de dato char se usa para almacenar caracteres Unicode simples. Debido a que el conjunto de caracteres Unicode se compone de valores de 16 bits, el tipo de datos char se almacena en un entero sin signo de 16 bits.
Además de los tipos de datos nativos existe una gran variedad de clases destinadas a la representación de diversos tipos de información. Una de los más utilizados es el tipo de dato String que representa las cadenas de caracteres.

Datos booleanos
El tipo de datos boolean sólo acepta dos posibles valores: true o false. Es utilizado para la creación de variables de control.

Declaración de Variables

Para poder utilizar una variable debemos declararla.
Es necesario "avisarle" a Java qué variables vamos a usar, para este proceso se requiere especificar tipo de dato, nombre y opcionalmente un valor inicial.
Ej:

- int i;
- char letra='a', opc;
- float num, res=7.5;
- boolean existe= false;

Asignación

Es el proceso de darle un valor a una variable, el mismo puede ser asignado al momento de la declaración o posteriormente usando el signo de igual. Es importante aclarar que a la hora de asignar un carácter el mismo debe estar entre comillas simples (‘z‘) y al asignar un float debemos terminar el numero con una f (3.1416f), si la omitimos el compilador entenderá que estamos asignando un double
Ej:

- i = 7;
- numDouble = 3.1416;
- numFloat =3.1416f;
- opc= ‘z’;

Listar

La sentencia System.out.println(), se puede usar para desplegar el contenido de las variables.

```
1.  public class listarVariables {
2.          public static void main(String args []){
3.                  int a;
4.                  float pi;
5.                  char letra;
6.                  a= 7;
7.                  pi= 3.1416f;
8.                  letra = 'z';
9.                  System.out.println(" a vale " + a + ", pi vale " + pi + " y letra vale
    " + letra);
10.         }
11. }
```

A diferencia de C en Java no es necesario la conversión de tipos de datos a la hora de mostrarlos. Como se puede apreciar en la linea 9, al invocar el método, se debe pasar como parámetro el texto entre comillas (“ ”), mas las variables a mostrar.
El símbolo de más (+) es utilizado para concatenar cadenas y variables.

Carga

Es el proceso de asignación de información a una variable mediante el uso del teclado. En C se utilizaba la función Scanf() a la cual se le debía pasar una variable para almacenar la información ingresada, a su vez se debía especificar con un parámetro de conversión el tipo de dato de la variable a utilizar.

El lenguaje C comparte con Java la clase IN, pero se sugiere no utilizarla por ser más complejo su uso, particularmente para el alumno.

En Java esta tarea se puede realizar con la clase Scanner que pose una serie de métodos que harán más sencilla el manejo de la entrada. Para utilizar esta clase se debe incluir el paquete java.util específicamente java.util.Scanner.
La forma de incluir un paquete o clase es la siguiente: import java.util.Scanner;

A continuación un ejemplo que muestra su funcionamiento.

```
1.  import java.util.Scanner;
2.  public class usarScanner {
3.          public static void main(String args []){
4.                  int edad;
```

```
5.                  float altura;
6.                  String nombre;
7.                  Scanner input = new Scanner( System.in);
8.                  System.out.println("Ingrese su nombre");
9.                  nombre=input.next();
10.                 System.out.println("Ingrese su edad");
11.                 edad=input.nextInt();
12.                 System.out.println("Ingrese su altura");
13.                 altura=input.nextFloat();
14.                 System.out.println("\n Los datos ingresados fueron: \n Nombre:
    " + nombre + "\n                    Edad: " + edad + "\n Altura: " + altura );
15.         }
16. }
```

En la linea 7 del ejemplo se esta instanciando un objeto de la clase Scanner, esto se limita a definir una variable con tipo de dato Scanner y con el uso del operador new (destinado a la reserva dinámica de memoria), se invoca al constructor de la clase. En este caso se le pasa como parámetro el archivo de entrada del sistema, (podría ser por ejemplo cualquier otro archivo con extensión txt)
Como se puede apreciar en la linea 12, 14 y 16 existen métodos destinados a la obtención de información, definidos para todos los tipos primitivos y el tipo String. Estos métodos retornan el primer elemento de lo ingresado que cumpla con las características de los tipos de datos solicitados.

- input.next()-retorna un elemento de tipo String.
- input.nextInt()-retorna un elemento de tipo int.
- input.nextFloat()-retorna un elemento de tipo Float.
- input.nextBoolean()-retorna un elemento booleano.

Estos son los métodos más comunes pero como se menciono recientemente hay un método para cada tipo de dato primitivo entre otros.

Ejercicio 10
Crear un programa que solicite un número y calcule el doble del mismo.

Construya el pseudocódigo con los alumnos en el pizarrón y pídales que lo codifiquen en Java

Pseudocodigo
Inicio
    Mostrar “Ingrese un número”
    Leer numero
    doble=numero*2
    Mostrar “el doble es” doble
Fin

Solución:

```
import java.util.Scanner;

public class doble {
    public static void main(String args []){
        int num,resultado;
        Scanner input = new Scanner( System.in);
        System.out.println("Ingrese un número");
        num=input.nextInt();
        resultado=num*2;
        System.out.println("El doble del numero ingresado es:" + resultado  );
    }
```

```
}
```

Ejercicio 11
Crear un programa que solicite dos números y retorne su suma.
Ejemplo:

Ingrese un número: 8
Ingrese otro número:4
La suma es 12

Solución:

```
import java.util.Scanner;


public class suma {
	public static void main(String args []){

		int num,num2;

		Scanner input = new Scanner( System.in);
		System.out.println("Ingrese un número");
		num=input.nextInt();
		System.out.println("Ingrese otro numero");
		num2=input.nextInt();
		System.out.println(num + "+" + num2 + "=" + (num+num2) );

	}
}
```

Ejercicio 12
Crear un programa que solicite dos números enteros y retorne la división entre estos.

Trabaje con los alumnos el concepto de división real (exacta) y entera (donde necesita resto)

Solución:

```
import java.util.Scanner;


public class division {
     public static void main(String args []){
          int num,num2;
          float resultado;

          Scanner input = new Scanner( System.in);
          System.out.println("Ingrese un número");
          num=input.nextInt();
          System.out.println("Ingrese otro numero");
          num2=input.nextInt();
          resultado=(float)num/num2;
          System.out.println("La división entera entre " + num + " y " + num2 + " es " +
num/num2);
```

```
            System.out.println("El resto de la división entre " + num + " y " + num2 + " es
" + num%num2);
            System.out.println("La división real entre " + num + " y " + num2 + " es " +
resultado);

      }
}
```

CAST

Es importante destacar que a la hora de hacer una división real uno de los elementos de la misma debe ser de tipo float o doble. Como en este caso ninguno de los dos lo era se realizo una conversión de tipo (cast). El operador cast realiza este proceso, es decir convierte datos, variables o expresiones a un nuevo tipo de dato, su formato es:

- (nuevo tipo) dato;

Ejercicio 13

Ingrese un valor en grados Celsius y convertirlo a Fahrenheit
ºF = ºC*9/5+32

Aproveche la ocasión para aclarar que ambos son grados centígrados, ya que usan una escala divisible entre 100. Sugiera a los alumnos no llamar grados centígrados a los grados Celsius.

Tema 6: Sentencia IF

Instrucciones de control

Controlar el flujo es determinar el orden en el que se ejecutarán las instrucciones en nuestros programas.
Si no existiesen las sentencias de control entonces los programas se ejecutarían de forma secuencial, empezarían por la primera instrucción e irían una a una hasta llegar a la última.
Pero, obviamente este panorama sería muy malo para el programador. Por un lado, en sus programas no existiría la posibilidad de elegir uno de entre varios caminos en función de ciertas condiciones (sentencias de selección). Y por el otro, no podrían ejecutar algo repetidas veces, sin tener que escribir el código para cada una (sentencias repetitivas).
Para estos dos problemas tenemos dos soluciones: las sentencias de control selectivas y las repetitivas.
Estos dos conjuntos de sentencias se les llama estructuradas porque a diferencia de las simples pueden contener en su cuerpo otras sentencias.
Las sentencias selectivas se llaman así porque permiten elegir uno de entre varios caminos por donde seguirá la ejecución del programa. Esta selección viene determinada por la evaluación de una expresión lógica. Este tipo de sentencias se dividen en dos:

- La sentencia if
- La sentencia case
-

A las sentencias repetitivas se les conoce también como sentencias iterativas ya que permiten realizar algo varias veces (repetir, iterar).
Estudiaremos las siguientes:

- La sentencia while
- La sentencia do..while
- La sentencia for

Sentencia if..else

La idea básica de esta sentencia es la de encontrarnos ante una bifurcación de un camino, en la que seguiremos por uno u otro camino dependiendo de la respuesta a una pregunta que se halla escrita en la bifurcación.
Una variante de esto es que en lugar de dos posibilidades, se nos presenten más caminos por los que poder seguir.
Esta sentencia evalúa una condición: si la misma es cierta ejecuta un código, sino es cierta ejecuta el código a continuación del ELSE.
Ejemplo:

Pseudocódigo	**Java**
IF N > 2	if (n>2)
Mostrar “Es mayor que 2”	system.out.println (“Es mayor que 2”);
SINO	else
Mostrar “NO Es mayor que 2”	system.out. println (“No es mayor que 2”);
FINSI	

Una sentencia if evaluará primero su condición, y si el resultado es true ejecutará la sentencia o bloque de sentencias que se encuentre justo después

de la condición. Si la condición se evalúa como false y si existe una cláusula else, entonces se ejecutará la sentencia o bloque de sentencias que siguen al else. Esta parte es opcional.
La única restricción que se impone a la expresión condicional es que sea una expresión válida de tipo booleano que se evalúe como true o como false.

Un ejemplo de código con un if-else es el siguiente, en el que se elige el mayor de dos enteros:

```
int a=3, b=5, m;

if (a<b)
   m=b;
else
   m=a;
```

Un ejemplo de condicional sin parte else es el siguiente, donde se imprime si un número es par o impar:

```
int a=3;

String mensaje="El número " + a + " es ";

if (a%2 != 0)
   mensaje+="im";

mensaje+="par.";
System.out.println(mensaje);
```

Produciendo el resultado :

El número 3 es impar.
El número 4 es par.
dependiendo de si a es inicializada a 3 o a 4.

Reglas para el if

1. La condición va entre paréntesis
2. La línea del if y else no lleva punto y coma
3. El IF y el ELSE se ejecuta para una sola sentencia.
4. Si se necesita más de una sentencia se encierra entre llaves. Esto se llama bloque.

Símbolos para las condiciones
Los símbolos para hacer operaciones son las siguientes

Matemática	Java
=	==
<	<
>	>
≤	<=
≥	>=
≠	!=

Ejercicio 14

Modificar y probar el programa anterior para ingresar un número y decir si es par o impar
Hacer pseudocódigo y programa en C

Estructura else if

Es posible construir una serie de comprobaciones uniendo un if a la cláusula else de un if anterior. Por ejemplo, el siguiente trozo de código muestra una calificación dada una nota numérica:

```
float nota=6.7F;
String calificacion;

if(nota>=11.0)
  calificacion="Sobresaliente";
else if(nota>=9.0)
  calificacion="Muy Bueno";
else if(nota>=7.0)
  calificacion="Aceptable";
else
  calificacion="Insuficiente";

System.out.println("La calificación obtenida es: " + calificacion);
```

En este tipo de secuencias de sentencias if-else anidadas, las cláusulas else siempre se emparejan con el if más cercano sin else. Para variar este comportamiento se deben utilizar llaves para crear bloques de sentencias

Ejercicio 15

- Ingresar 2 números y calcular cociente y resto entero.
- Avisar que no es posible cuando el divisor es cero

Ejercicio 16

- Ingresar 2 números
- Decir si son iguales
- Si no son iguales decir si el primero es mayor o el segundo es mayor

Ejercicio 17

- Ingresar el año de nacimiento de una persona
- Mostrar la edad que tendrá a fin de año (o sea que cumplirá durante el año lectivo)
- Decir si es mayor o menor de edad
- Decir si falleció o si aún no ha nacido
- Cuando esté pronto testearlo con los años:
    - 1900
    - 1950
    - 2000
    - 2050

Tema 7: Estructuras repetitivas

La estructura while

Estructura destinada a iterar un bloque de código.
Un bloque puede ser una sola sentencia o un conjunto de código delimitado por llaves ({ }).
La estructura del *while se puede dividir* en tres grandes componentes

- La palabra reservada *while*
- La condición expresada entre paréntesis, mientras sea verdadera se repetirá el bloque especificado
- El bloque de código a repetir.

Ej:

```
while (a!=0){
        System.out.println("ingrese un numero (0 para salir)");
        a=input.nextInt();
}
```

Mientras no se ingrese el numero cero la condición es verdadera y repetirá el bloque de código.

Tener en cuenta:

- La condición va entre paréntesis
- La línea del while no lleva punto y coma
- El while se termina cuando la condición se hace falsa
- Si la condición es falsa de entrada, nunca se ejecuta el código dentro del while

Ejercicio 18
Crear un programa que solicite números hasta que se ingrese un 0 y retorne la suma de los mismos.

```
Inicio
suma=0
num=1
Mientras num!=0
        Mostrar “Ingrese un número”
        Leer n
        suma = suma + num
Fin Mientras
Mostrar “La suma es”, suma
Fin
```

Posible solución:

```
public class sumarNnumeros {
        public static void main(String args []){
                int suma=0, num=1;
                Scanner input= new Scanner(System.in);
                System.out.println("Ingrese un número");
                num=input.nextInt();
                while(num!=0){
                        suma=suma+num;
                        System.out.println("Ingrese otro número");
```

```
                num=input.nextInt();
            }
            System.out.println("La suma de los numeros ingresados es: " + suma);

        }
}
```

Contadores y acumuladores

Los contadores al igual que los acumuladores son variables a las cuales se las trata de manera particular.
Se les denomina *contador* a aquellas variables que son incrementadas o decrementadas con un valor constante para todas las iteraciones. Sin embargo, a aquellas que son modificadas con valores diversos se les denomina *acumuladores*.

Ejemplo contadores	Ejemplo acumuladores
contador=contador+1;	acumulador=acumulador+valor;
contador2=contador2+10;	acumulador=valor+acumulador;
contador++;	acumulador+=valor
contador=contador-1;	acumulador=acumulador-valor;
contador--	acumulador-=valor

Analice con los alumnos las características de los símbolos ++, --, +=, -=

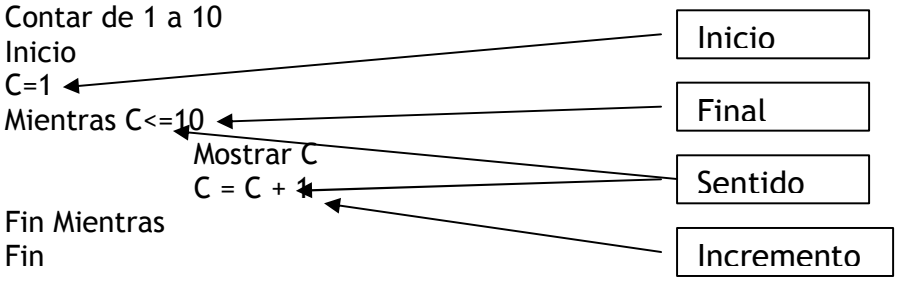
Ejercicio 19
Modificar el ejercicio anterior para que despliegue también la cantidad de números ingresados.

Solucion:

```
import java.util.Scanner;

public class sumarNnumerosContarlos {

        public static void main(String args []){
            int suma=0, num=1, cont=0;
            Scanner input= new Scanner(System.in);
            System.out.println("Ingrese un número");
            num=input.nextInt();
            while(num!=0){
                cont++;
                suma=suma+num;
                System.out.println("Ingrese otro número");
                num=input.nextInt();
            }
            System.out.println("Se ingresaron " + cont + " numeros y su suma es: " +
suma);

        }
}
```

Ejercicio 20

Hacer un programa que muestre todos los números del 1 al 10.



Trabaje con los alumnos la función que cumple cada elemento del contador

Solución:

```
import java.util.Scanner;

public class conteo1_10 {
	public static void main(String args []){
		int cont=1;
		while(cont<=10){
			System.out.println(cont);
			cont++;
		}
	}
}
```

Ejercicio 21

Modificar el programa anterior para que muestre los números múltiplos de 3, del 1 al 100.

Ejercicio 22

Modificar el programa anterior para que comenzando en 1000 decremente el valor del mismo en 0,25 hasta alcanzar el numero 900.

Ejercicio 23

Crear un programa en java que solicite un carácter y un numero, para luego desplegar el carácter tantas veces como indique el numero

Solución:

```
public class nCaracteres {
	public static void main(String args []){
		int repetir,cont=0;
		char caracter;
		Scanner input= new Scanner(System.in);
		System.out.println("ingrese un carácter ");
		caracter=input.next().charAt(0);//Convierto de String to char.
		System.out.println("Ingrese la cantidad de veces que desea repetirlo");
		repetir=input.nextInt();
		while(cont<repetir){
			System.out.println(caracter);
			cont++;
```

```
            }

        }
}
```

Nota: en la linea 7 se utiliza el método charAt(0) perteneciente a la clase String que permite extraer un char de cualquier posición de un string (comenzando de 0), en este caso el string era de largo uno

La estructura do while

Es muy similar a la estructura while, sus diferencias radican en que la estructura do while evalúa la condición al final del bloque a ejecutar.
Esto garantiza que el bloque de código será ejecutado al menos una vez.

Ejemplo:

```
do{
        System.out.println("ingrese un numero (0 para salir)");
        a=input.nextInt();
}while(a!=0);
```

Tema 8: Menú

Es una estructura de repetición con opciones
De las opciones posibles solo se puede elegir una
Termina con una opción destinada para ello. Generalmente la opción 0.

Pseudocódigo
Menú
Inicio
    Hacer
        Mostrar "1-Opción 1"
        Mostrar "2-Opción 2"
        Mostrar "3-Opción 3"
        …
        Leer opcion
        Si opcion = 1
            …
        Fin Si
        Si opcion = 2
            …
        Fin Si
    Mientras opcion != 0
Fin

Ejercicio 24
Hacer un menú para que muestre el resultado de algunos partidos de fútbol
Codificarlo en Java y probarlo.

Sentencia switch

Esta sentencia examina los valores de una variable y en base a ello ejecuta sentencias independientes.
La Sintaxis empleada por un *switch* es la siguiente:

```
switch (variable)  {
    case <posible valor> : Instrucciones : break;
    case <posible valor> : Instrucciones : break;
    case <posible valor> : Instrucciones : break;
    case <posible valor> : Instrucciones : break;
    case <posible valor> : Instrucciones : break;
    default : Instrucciones ;
}
```

- Dada una variable de entrada esta se define seguido de la sentencia switch.
- Se abre una llave para iniciar los posibles valores que pueda tomar dicha variable.
- Los juegos de valores son iniciados con case seguido del posible valor de la variable, posteriormente es definido un juego de instrucciones que serán ejecutados en caso de corresponder con el valor de la

variable y finalmente (opcional) se utiliza vocablo break para salir de ciclo *case*.

- Un valor opcional es la definición de la linea default, cuyas instrucciones serán ejecutadas en caso que la variable del *switch* no coincida con los valores definidos.

```
public class usarSwitch {
   public static void main(String[] args) {
      int mes=3;
      switch (mes) {
         case 1:  System.out.println("Enero"); break;
         case 2:  System.out.println("Febrero"); break;
         case 3:  System.out.println("Marzo"); break;
         case 4:  System.out.println("Abril"); break;
         case 5:  System.out.println("Mayo"); break;
         case 6:  System.out.println("Junio"); break;
         case 7:  System.out.println("Julio"); break;
         case 8:  System.out.println("Agosto"); break;
         case 9:  System.out.println("Septiembre"); break;
         case 10: System.out.println("Octubre"); break;
         case 11: System.out.println("Noviembre"); break;
         case 12: System.out.println("Diciembre"); break;
         default: System.out.println("Este mes no existe"); break;
      }
   }
}
```

Tenga en cuenta:

- Dentro del CASE no se necesitan llaves
- El programa entra en el DEFAULT si no entró en otra opción
- El BREAK hace que el CASE se termine

Ejercicio 25

La sentencia DO..WHILE junto con la selectiva múltiple (switch) son muy utilizadas para formar "menúes" que permiten a los usuarios seleccionar funcionalidad de los programas.
Un menú es entonces una estructura de repetición con opciones, que solo finaliza con una de estas.

Ejercicio 26

Crear un menú que permita seleccionar los resultados de diversos partidos.

```
import java.util.Scanner;

public class menuFutbol{
    public static void main(String args []){
      int op;
      Scanner input= new Scanner(System.in);
      do{
        System.out.println("\n1- Naciona VS Liverpool");
        System.out.println("2- Danubio VS Cerro");
        System.out.println("3- Peñarol VS Defensor");
        System.out.println("0- Salir");
        System.out.println("Ingrese un partido par ver su resultado");
        op=input.nextInt();
        switch(op){
          case 1: System.out.println("El partido culmino con victoria de
Nacional por 1 a 0"); break;
         case 2: System.out.println("El partido culmino con victoria de la
franja por 2 tantos contra 0");break;
           case 3: System.out.println("El aurinegro se impuso ante la viola por
un marcador de 1 a 0");break;
         case 0: break;
         default:System.out.println("Opcion incorrecta");

        }

      }while(op!=0);

   }
}
```

Indentado

Indentación es un anglicismo (de la palabra inglesa *indentation*) de uso común en informática que significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores para separarlo del texto adyacente, lo que en el ámbito de la imprenta se ha denominado siempre como *sangrado* o *sangría (extraído de wikipedia)*.
La palabra indentación no es una palabra reconocida por la Real Academia Española (consultado en la vigésimosegunda edición). La Real Academia recomienda utilizar el término «sangrado».
En los lenguajes de programación de computadoras, la indentación es un tipo de notación secundaria utilizado para mejorar la legibilidad del código fuente por parte de los programadores, teniendo en cuenta que los compiladores o intérpretes raramente consideran los espacios en blanco entre las sentencias de un programa.
Son frecuentes discusiones entre programadores sobre cómo o dónde usar la indentación, si es mejor usar espacios en blanco o tabuladores, ya que cada programador tiene su propio estilo.
El objetivo fundamental de la indentación del código fuente es facilitar su lectura y comprensión. Hay dos tipos de posibles lectores del código fuente: programas y personas. A los programas les da igual la indentación, leen bien nuestro código siempre que cumpla la sintaxis del lenguaje. Luego la indentación debe centrarse en la lectura y comprensión del código por personas.

Para entender cómo hay que indentar un código primero hay que entender cómo lo lee una persona. Actualmente la mayor parte de las personas que leen código fuente lo hacen con editores de texto simples o con la ayuda de los editores de los entornos de desarrollo. Por tanto nos vamos a centrar en este tipo de lectores.

Cuando una persona lee un código fuente, lo hace siguiendo una serie de patrones de lectura. Si queremos facilitar la lectura del código lo primero que hay que entender son estos patrones. La mejor bibliografía sobre patrones de lectura son los libros que enseñan sistemas de lectura rápida; dos puntos fundamentales de estos sistemas son el estudio de los patrones de movimiento de la vista y el reconocimiento de la estructura del texto.

Los patrones de lectura nos indican el movimiento que siguen los ojos al leer un texto. Cuando leemos un texto, la vista se desliza a saltos, no se desplaza de una manera continua. Nuestra vista capta la información y que haya alrededor del punto en que se detiene la vista. Un buen lector hace pocas paradas para leer una línea, aprovechando más su campo visual.

Por otra parte, al leer también utilizamos la visión periférica, aprovechando la estructura y disposición que tienen las letras para extraer información adicional del texto. Así, por ejemplo, instintivamente nos fijamos en el principio del párrafo para buscar la información relevante, y nos ayudamos de elementos como las listas, los tabuladores, tablas y otros elementos tipográficos al leer.

De estos puntos deducimos que la posición absoluta y relativa de los elementos que componen el texto del código fuente juegan un papel importante a la hora de facilitar la lectura y comprensión del mismo.

Requerimientos para definir un estilo de indentación

Vamos a enumerar los principales requisitos para poder definir un buen estilo de indentación. Algunos de los más importantes serán:

- Que el posicionamiento de los elementos dentro del texto sea predecible. Cuanto más predecible es la posición, el movimiento de los ojos llega antes a los puntos relevantes del código fuente. El punto más relevante de información es siempre el principio de la línea, por lo que hay que tener especial atención en él.
- Que la relevancia de la agrupación de los elementos refleje la relevancia de éstos dentro del programa.
- Priorizar la identificación del flujo del programa frente al detalle de los elementos que lo componen. Por ejemplo es más importante siempre tener una visión global de lo que hace una función que del detalle de cómo se declaran sus variables.
- Que los comentarios respeten la indentación de los elementos a los que van asociados, posicionándose de manera relativa a ellos según su relevancia. Si el lenguaje lo permite, hay que distinguir la relevancia de un comentario posicionándolo sobre el elemento (más importante) o al lado de él (menos importante). Es mejor situar los comentarios antes del elemento, ya que nos sirven de introducción a lo que va a continuación, y muchas veces nos permitirá saltarnos un bloque de código que no nos interesa.
- Utilizar la indentación basada en tabuladores. La indentación basada en el uso de caracteres de espacio en blanco es menos flexible que la basada en tabuladores, ya que los editores y entornos de desarrollo modernos permiten adapatar con facilidad el ancho de los tabuladores al gusto de cada programador.
- Mantener el tamaño de las líneas dentro de los márgenes de la pantalla, siempre que sea posible. Debemos tener en cuenta al lector potencial del código fuente. No se debe pensar que el lector tiene una pantalla demasiado pequeña o grande, sino asumir que el lector tiene una pantalla típica dentro del entorno de desarrollo habitual. Por ejemplo actualmente los desarrolladores suelen tener pantallas capaces de mostrar 40 líneas y 120 columnas de caracteres con facilidad. En este caso, utilizar el antiguo margen de 80 columnas por línea y 25 líneas por pantalla es más bien escaso.
- Utilizar el menor número de normas de indentación requerido. Indentar elementos en exceso complica visualmente la lectura del código, si no son estrictamente necesarios.

NOTA:
Si tiene dudas si la palabra a usar es “identar” o “indentar” tome en cuenta que el sufijo “i” no se usa en Idioma Español. El sufijo correcto es “in”.
La palabra original en inglés es “indent”.
Sugerencia:

- Se indenta siempre dentro de llaves
- Siempre se indenta el IF, el SWITCH, el CASE, y el WHILE
- En estas sentencias se abre llave en la misma línea y se cierra frente a dicha sentencia
- Se deja una línea en blanco antes de la clase, SWITCH y WHILE,

En Java existen tres tipos de comentarios:

• Comentario de párrafo.

```
/* Esto es un comentario
que puede tener más de una línea */
```

• Comentario de línea.

```
// Esto sirve para comentar una única línea
```

• Comentario de documentación.

```
/** Esto además de ser un comentario
sirve para que el programa javadoc documente
nuestro programa automáticamente */
```

Se puede leer más en http://www.upct.es/~orientap/ConvencionesCodigoJava.pdf

Ejercicio 27:
Indentar correctamente el menú realizado en el Ejercicio anterior

Tema 9: Operadores Lógicos

Los operadores relacionales

Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa. Por ejemplo, 8>4 (ocho mayor que cuatro) es verdadera, se representa por el valor **true** del tipo básico **boolean**, en cambio, 8<4 (ocho menor que cuatro) es falsa, **false**.
En la siguiente tabla se muestra en la primera columna de la tabla los símbolos de los operadores relacionales, el la segunda, el nombre de dichos operadores, y a continuación su significado mediante un ejemplo.

Operador	nombre	ejemplo	Significado
<	menor que	*a<b*	*a* es menor que *b*
>	mayor que	*a>b*	*a* es mayor que *b*
==	igual a	*a==b*	*a* es igual a *b*
!=	no igual a	*a!=b*	*a* no es igual a *b*
<=	menor que o igual a	*a<=5*	*a* es menor que o igual a *b*
>=	mayor que o igual a	*a>=b*	*a* es menor que o igual a *b*

Se debe tener especial cuidado en no confundir el operador asignación con el operador relacional igual a. Las asignaciones se realizan con el símbolo =, las comparaciones con ==.
En el programa *Relacionales*, se compara la variable *i* que guarda un 8, con un conjunto de valores, el resultado de la comparación es verdadero (**true**), o falso (**false**).

```
public class Relacionales{
   public static void main(String[] args) {
      int x=8;
      int y=5;
      boolean compara=(x<y);
      System.out.println("x<y es "+compara);
      compara=(x>y);
      System.out.println("x>y es "+compara);
      compara=(x==y);
      System.out.println("x==y es "+compara);
      compara=(x!=y);
      System.out.println("x!=y es "+compara);
      compara=(x<=y);
      System.out.println("x<=y es "+compara);
      compara=(x>=y);
      System.out.println("x>=y es "+compara);
   }
}
```

Los operadores lógicos

Los operadores lógicos son:

- && AND (el resultado es verdadero si ambas expresiones son verdaderas)
- || OR (el resultado es verdadero si alguna expresión es verdadera)
- ! NOT (el resultado invierte la condición de la expresión)

AND y OR trabajan son dos operandos y retornan un valor lógico basadas en las denominadas tablas de verdad.
El operador NOT actúa sobre un operando.
Estas tablas de verdad son conocidas y usadas en el contexto de la vida diaria, por ejemplo: "si hace sol Y tengo tiempo, iré a la playa", "si NO hace sol, me quedaré en casa", "si llueve O hace viento, iré al cine".
Las tablas de verdad de los operadores AND, OR y NOT se muestran en las tablas siguientes

El operador lógico AND

x	**y**	**resultado**
true	true	true
true	false	false
false	true	false
false	false	false

El operador lógico OR

x	**y**	**resultado**
true	true	true
true	false	true
false	true	true
false	false	false

El operador lógico NOT

x	**resultado**
true	false
false	true

Los operadores AND y OR combinan expresiones relacionales cuyo resultado viene dado por la última columna de sus tablas de verdad. Por ejemplo:

(a<b) && (b<c)

es verdadero (**true**), si ambas son verdaderas. Si alguna o ambas son falsas el resultado es falso (**false**).

En cambio, la expresión

(a<b) ||(b<c)

es verdadera si una de las dos comparaciones lo es. Si ambas, son falsas, el resultado es falso.

La expresión " NO *a* es menor que *b*"

!(a<b)

es falsa si *(a<b)* es verdadero, y es verdadera si la comparación es falsa. Por tanto, el operador NOT actuando sobre *(a<b)* es equivalente a

(a>=b)

La expresión "NO *a* es igual a *b*"

!(a==b)

es verdadera si *a* es distinto de *b*, y es falsa si *a* es igual a *b*. Esta expresión es equivalente a

(a!=b)

Ejercicio 28

- Ingresar N notas de alumnos:
- Validar cada nota
- Devolver la suma
- La cantidad
- El promedio

Ejercicio 29

Modificar el programa anterior para que muestre la menor y la mayor nota

```
Inicio
        Max=0
        N=1
        Mientras n != 0
                Leer n
                Si n>max
                        max=n
                Fin si
        Fin mientras
        Mostrar “El mayor fue”, max
Fin
```

Ejercicio 30

- Ingresar N notas
- Indicar cuántas notas son suficientes (de 7 a 12) y cuántas insuficientes (de 1 a 6)
- Indicar cuántos alumnos se van a examen en Febrero

Ejercicio 31

- Ingresar un número de 0 a 1.999 y expresarlo con palabras
- Inicio
    Ingresar un número
    Verificar el número
    Analizar casos especiales (0, 100, 1000)
    Tomar la primera cifra del número
    Buscar el nombre de esa cifra con un switch
    Repetir:
        Con los millares
        Con las centenas
        Con los números mayores a 20
        Con los números del 1 a 20
  Fin

Tema 10: Control de datos ingresados

Banderas

Una bandera es una variable boolean que controla el flujo del programa.
Se usan para controlar procesos
Por ejemplo:

- Pronto=false; //Aun no esta pronto
- Pronto=true; //Ya está pronto
- If (pronto) //Si está pronto hago…
  o
  If (!pronto) //Si no está pronto hago…

WHILE(TRUE)

Hace que el bucle siempre se repita indefinidamente
Se sale del mismo con break
Sirve para obligar al usuario a ingresar un dato correctamente.
En este caso da lo mismo usar while o do..while

Ejercicio 32

Ingresar una fecha del siglo XX o XXI y verificar si es correcta

```
Inicio
        bisiesto = falso
        Mientras true
                Mostrar “Ingrese una fecha”
                Leer dia, mes, anio
                bisiesto = anio%4==0 y anio…
                SI dia < 1 o dia> 31 o mes <1 o mes > 12 o
                        anio < 1900 o anio > 2099
                        Mostrar “Esa fecha no existe”
                Si no
                        Si (mes=4 o mes = 6 o mes = 9…) y dia=31
                                Mostrar “Ese mes no tiene 31 dias”
                        sino
…
```

Algoritmo del año bisiesto

Un año es **bisiesto** si es divisible entre 4, excepto si es divisible entre 100 pero no entre 400.
En programación, el algoritmo para calcular si un año es bisiesto es útil para la realización de calendarios.

Considérense las proposiciones o enunciados lógicos siguientes:
p: Es divisible entre 4
q: Es divisible entre 100
r: Es divisible entre 400

La fórmula lógica que se suele usar para establecer si un año es bisiesto es cuando $(p \wedge \neg q) \vee r$ es verdadera, pero esta otra $p \wedge (\neg q \vee r)$ sería más eficiente.

Tema 11: La estructura for

Al igual que el while o do while es una estructura destinada a iterar un bloque de codigo.
En particular la estructura for esta especialmente diseñada para la iteración en las que conocemos la cantidad de veces que se repetirá el código.
La sintaxis del for podemos subdividirlo en 3 partes: expresión inicio, condición e incremento:

```
for(inicial; condicion; incremento){
        sentencia;
}
```

- La expresión inicial se ejecuta antes de entrar en el bucle.
- Si la condición es cierta, se ejecuta sentencia y después la expresión final.
- Luego se vuelve a evaluar la condición, y así se ejecuta la sentencia una y otra vez hasta que la condición sea falsa.

Contadores

Una de sus mayores utilidades es usarlo para contar ya que en una línea se resuelve todo el contador
Observe la equivalencia for/while

```
for (int i=1;i<=10;i++)
  System.out.println (i);
```

```
int i=1;
while(i<=10){
  System.out.println (i);
  i++;
}
```

Ejemplos:

```
int i;
...
for ( i=0; i<10; i++ )
        System.out.println (i );
for (int c=1; c<=10; c++)
        System.out.println (c);
```

Pregunta: ¿Qué números muestran estos ejemplos?

NOTA:

Las tres expresiones del bucle for se pueden omitir, con el siguiente resultado.

SI se omite	**Resultado**
expresión_inicial	no se hace nada antes del bucle
condición	la condición es siempre cierta
final	no se hace nada tras cada iteración

No es una buena práctica omitir elementos del for, por lo tanto hay que tener cuidado que su estructura esté completa.

Ejercicio 33:

Mostrar un conteo en columnas del 0 al 200 de 5 en 5 (usando for)

Simplificación de operadores:

a = a +1	a++
b = b – 1	b--
c = c + 4	c+=4
d = d * 5	d*=5
e = e – 6	e-=6
f = f / 7	f/=7

Ejercicio 34

Hacer un conteo, de 10 a 0 de 0.5 en 0.5.
Use algún operador simplificado.

Ejercicio 35: Hacer un programa para generar tablas de multiplicar

Se debe solicitar el número de la tabla y luego generarla

For Anidado

Cuando un for contiene otro for dentro se dice que están anidados
El primer for repite n veces al otro for
Ejemplo de un producto cartesiano:

```
public class prodCartesiano {
   public static void main(String[] args) {
        for(char a='a';a<='z';a++)
        for(int b=1;b<=8;b++)
                        System.out.print("{"+a+","+b+"}");

   }
}
```

ASCII

El código ASCII (siglas en ingles para American Standard Code for Information Interchange, es decir Código Americano (estadounidense) Estándar para el intercambio de Información) (se pronuncia Aski).

Fue creado en 1963 por el Comité Estadounidense de Estándares o "ASA", este organismo cambio su nombre en 1969 por "Instituto Estadounidense de Estándares Nacionales" o "ANSI" como se lo conoce desde entonces.

Este código nació a partir de reordenar y expandir el conjunto de símbolos y caracteres ya utilizados por ese entonces en telegrafía por la compañía Bell.

En un primer momento solo incluía las letras mayúsculas, pero en 1967 se agregaron las letras minúsculas y algunos caracteres de control, formando así lo que se conoce como US-ASCII, es decir los códigos del 0 al 127.
Así con este conjunto de solo 128 caracteres fue publicado en 1967 como estándar, conteniendo todos lo necesario para escribir en idioma ingles.

En 1981, la empresa IBM desarrolló una extensión de 8 bits del código ASCII, llamada "pagina de código 437", en esta versión se reemplazaron algunos caracteres de control obsoletos, por caracteres gráficos. Además se incorporaron 128 caracteres nuevos, con símbolos, signos, gráficos adicionales y letras latinas, necesarias para la escrituras de textos en otros idiomas, como por ejemplo el español. Así fue como se agregaron los caracteres que van del ASCII 128 al 255.
IBM incluyó soporte a esta página de código en el hardware de su modelo 5150, conocido como "IBM-PC", considerada la primera computadora personal. El sistema operativo de este modelo, el "MS-DOS" también utilizaba el código ASCII extendido.

Casi todos los sistemas informáticos de la actualidad utilizan el código ASCII para representar caracteres y textos (44) .

Ejercicio 36

Imprimir en pantalla la tabla ASCII

```
public class tablaASCII {
   public static void main(String[] args) {
        int letra=32;
        for(int f=1;f<=6;f++){
           System.out.print(letra+"-");
           for(int c=1;c<=16;c++)
                System.out.print((char)letra++ + " ");
        System.out.println();
        }
    }
}
```

Números al azar

La clase *Random*

La clase *Random* proporciona un generador de números aleatorios que es más flexible que la función estática random de la clase *Math*.
Para crear una secuencia de números aleatorios tenemos que seguir los siguientes pasos:

1. Proporcionar a nuestro programa información acerca de la clase *Random*. Al principio del programa escribiremos la siguiente sentencia.
   import java.util.Random;
2. Crear un objeto de la clase *Random*
3. Llamar a una de las funciones miembro que generan un número aleatorio
4. Usar el número aleatorio.

Constructores

La clase dispone de dos constructores, el primero crea un generador de números aleatorios cuya semilla es inicializada en base al instante de tiempo actual.

```
Random rnd = new Random();
```

El segundo, inicializa la semilla con un número del tipo **long**.

```
Random rnd = new Random(3816L);
```

El sufijo L no es necesario, ya que aunque 3816 es un número **int** por defecto, es promocionado automáticamente a **long**.
Aunque no podemos predecir que números se generarán con una semilla particular, podemos sin embargo, duplicar una serie de números aleatorios usando la misma semilla. Es decir, cada vez que creamos un objeto de la clase *Random* con la misma semilla obtendremos la misma secuencia de números aleatorios.
Esto no es útil en el caso de loterias, pero puede ser útil en el caso de juegos, exámenes basados en una secuencia de preguntas aleatorias, las mismas para cada uno de los estudiantes, simulaciones que se repitan de la misma forma una y otra vez, etc.

Funciones miembro

Podemos cambiar la semilla de los números aleatorios en cualquier momento, llamando a la función miembro *setSeed*.

```
rnd.setSeed(3816);
```

Podemos generar números aleatorios en cuatro formas diferentes:

```
rnd.nextInt();
```

genera un número aleatorio entero de tipo **int**

```
rnd.nextLong();
```

genera un número aleatorio entero de tipo **long**

```
rnd.nextFloat();
```

genera un número aleatorio de tipo **float** entre 0.0 y 1.0, aunque siempre menor que 1.0

```
rnd.nextDouble();
```

genera un número aleatorio de tipo **double** entre 0.0 y 1.0, aunque siempre menor que 1.0

Casi siempre usaremos esta última versión. Por ejemplo, para generar una secuencia de 10 números aleatorios entre 0.0 y 1.0 escribimos

```
for (int i = 0; i < 10; i++) {
   System.out.println(rnd.nextDouble());;
}
```

Para crear una secuencia de 10 números aleatorios enteros comprendidos entre 0 y 9 ambos incluídos escribimos

```
int x;
for (int i = 0; i < 10; i++) {
      x = (int)(rnd.nextDouble() * 10.0);
      System.out.println(x);
}
```

(int) transforma un número decimal **double** en entero **int** eliminando la parte decimal.

Secuencias de números aleatorios

En el siguiente código, se imprimen dos secuencias de cinco números aleatorios uniformemente distribuídos entre [0, 1), separando los números de cada una de las secuencias por un carácter tabulador.

```
System.out.println("Primera secuencia");
for (int i = 0; i < 5; i++) {
   System.out.print("\t"+rnd.nextDouble());
}
System.out.println("");

System.out.println("Segunda secuencia");
for (int i = 0; i < 5; i++) {
   System.out.print("\t"+rnd.nextDouble());
}
System.out.println("");
```

Comprobaremos que los números que aparecen en las dos secuencias son distintos.

En la siguiente porción de código, se imprime dos secuencias iguales de números aleatorios uniformemente distribuídos entre [0, 1). Se establece la semilla de los números aleatorios con la función miembro *setSeed*.

```
   rnd.setSeed(3816);
   System.out.println("Primera secuencia");
   for (int i = 0; i < 5; i++) {
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");

   rnd.setSeed(3816);
   System.out.println("Segunda secuencia");
   for (int i = 0; i < 5; i++) {
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");
```

Ejemplo completo:

```
import java.util.Random;

public class Azar {
  public static void main (String[] args) {
    int[] ndigitos = new int[10];
    int n;

    Random rnd = new Random();

// Inicializar el array
    for (int i = 0; i < 10; i++) {
       ndigitos[i] = 0;
    }

// verificar que los números aleatorios están uniformemente distribuidos
for (long i=0; i < 100000L; i++) {
// genera un número aleatorio entre 0 y 9
       n = (int)(rnd.nextDouble() * 10.0);
//Cuenta las veces que aparece un número
       ndigitos[n]++;
    }

// imprime los resultados
    for (int i = 0; i < 10; i++) {
       System.out.println(i+": " + ndigitos[i]);
    }

//Dos secuencias de 5 número (distinta semilla)
    System.out.println("Primera secuencia");
    for (int i = 0; i < 5; i++) {
```

```
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");

   System.out.println("Segunda secuencia");
   for (int i = 0; i < 5; i++) {
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");

//Dos secuencias de 5 número (misma semilla)
   rnd.setSeed(3816L);
   System.out.println("Primera secuencia");
   for (int i = 0; i < 5; i++) {
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");

   rnd.setSeed(3816);
   System.out.println("Segunda secuencia");
   for (int i = 0; i < 5; i++) {
      System.out.print("\t"+rnd.nextDouble());
   }
   System.out.println("");
  }
}
```

Ejercicio 37:
Elegir un número al azar del 1 al 100 y adivinarlo
Decir si es menor o mayor cuando no se acierta
Decir en cuantos aciertos lo hizo

- ¿En cuántos intentos seguro que acierto un número del 1 al 100?

R: 7 intentos máximo usando bipartición

- ¿Puedo matematizar este concepto?

R: $Log_2(100)$
Logaritmo de 100 en base 2

- ¿Y en cuántos intentos un número del 1 al 1000?
- ¿Cómo puedo hacer para acertar el número en 1 intento?

R: Haciendo trampa e imprimiendo el numero elegido o usando una misma semilla de la cual conocemos la secuencia

Tema 12: Métodos (de la clase)

¿Qué es un método?

Un método es una parte del código que puede ser llamado y que ejecutará una serie de instrucciones para que nuestro programa sea ejecutado. En Java la mayoría de instrucciones siempre estarán dentro de un método (excepto en algunos casos que no se analizarán por el momento).

Un método es:

- Un bloque de código que tiene un nombre,
- recibe unos parámetros o argumentos (opcionalmente),
- contiene sentencias o instrucciones para realizar algo (opcionalmente) y
- devuelve un valor de algún Tipo conocido (opcionalmente).

En nuestro programa HolaMundo el método Principal fue:

```
public static void main(String[] args) {
 System.out.println("Hola Mundo!");
}
```

Los métodos en Java siempre estarán escritos con esa estructura:

```
<modificadores> <tipo> <nombre> (  <parámetros> ) {
<cuerpo>
}
```

1. MODIFICADORES: Son opciones que indicarán el comportamiento del método al momento de la ejecución (por el momento solo usaremos public static).

2. TIPO DE RETORNO: Es el tipo de datos que serán obtenidos una vez que la ejecución del método termine (los 8 tipos de datos más void).

3. NOMBRE: Es el identificador y nombre del método.

4. PARÁMETROS: La información que le enviaremos al método para que este haga lo que tiene que hacer!

5. CUERPO: Las instrucciones que nuestro método va a ejecutar.

En el ejemplo anterior:

MODIFICADORES: public static
TIPO DE RETORNO: void (Significa que no regresa ninguna información)
NOMBRE: main
PARÁMETROS: String [] args (Un vector de String que veremos más adelante)

CUERPO: System.out.println("Hola Mundo!");

Siguiendo esa misma lógica podemos definir un método para sumar dos números decimales (El tipo de datos decimal en Java es *double*) a y b.

MODIFICADORES: public static
TIPO DE RETORNO: double (Porque al sumar dos números decimales obtenemos otro número decimal).
NOMBRE: suma
PARÁMETROS: double a, double b (El método recibe dos parámetros, para realizar una función que sume dos números debemos recibir esos dos números)

Nuestro método queda escrito de la siguiente forma y sólo nos falta definir el cuerpo, es decir, cómo es que nuestro método realizará la suma de esos números:

```
public static double suma( double a, double b ) {
}
```

El cuerpo de este método es muy simple ya que sólo tenemos que sumar a + b y regresar el valor de esa suma.

```
public static double suma(double a, double b) {
 return a + b;
}
```

Tenemos ahora una función de nombre suma, que recibe dos parámetros (a,b) y que una vez que se ejecute obtendremos un valor decimal (*double*).

Entonces llamemos a nuestro método obtendremos algo más o menos así

suma (5 , 6) -> REGRESA: 11
suma (11,12) -> REGRESA: 23

Ahora, para realizar algunas pruebas lo único que tenemos que hacer, es poner todo dentro de una clase, y colocar un método principal, el cuál es el punto de inicio del programa.

```
public class Metodos {

 public static void main(String[] args) {
  System.out.println( suma(5,6) );
  System.out.println( suma(11,12) );
 }


 public static double suma(double a, double b) {
  return a + b;
```

```
 }
}
```

Guarda el archivo con el nombre de Metodos.java, compílalo y al correrlo verás que el resultado es
11.0
23.0

Ejercicio 38: Ingresar dos números y devolver la suma, mediante el método suma de la clase Calculadora

```
import java.util.Scanner;

public class Calculadora {
  public static void main (String[] args){
        suma();
  }

  public static void suma(){
          Scanner dato = new Scanner( System.in);
          int a, b, c;

          System.out.print("Ingrese un numero ");
          a=dato.nextInt();
          System.out.print("Ingrese un numero ");
          b=dato.nextInt();
          c=a+b;
          System.out.print(a+" + "+b+" es "+c);
  }
}
```

VARIABLES LOCALES Y DE CLASE

En JAVA el lugar donde sea declarada una variable afectará el uso que el programa quiera hacer de esa variable.
En Java las reglas básicas que determinan como una variable puede ser usada depende de 3 lugares donde se puede declarar una variable.

1. El primer lugar es dentro de cualquier método incluyendo main, a estas se les llama variables locales y solo pueden ser usadas por instrucciones que estén dentro de esa función o procedimiento.
2. El segundo lugar es como parámetro de un método, donde después de haber recibido el valor, podrá actuar como variable local en dicho método. En esencia una variable local solo es visible por el código de ese método donde fue definido y desconocida por el resto del programa.
3. El tercer lugar es fuera de todas las funciones incluyendo main(). A este tipo de variables se les llama variables de clase y podrán ser usadas por cualquier función o procedimiento del programa.

- Las variables de clase en Java deben ser inicializadas

Ejercicio 39: Realizar una calculadora
Pautas:

- Crear un método para opción de la calculadora (suma, resta, multiplicación, división, raíz cuadrada)
- Excepto la raíz las demás se realizan con 2 números
- En el caso de la raíz y la división se deben controlar las excepciones: casos donde se puede realizar (divisor cero o raíz negativa)
- Crear un método para el menú de la calculadora que será llamado desde el main

Tema 13: Array

También se les conoce como Vectores o Arreglos.

Debido a la existencia de la clase Vector de Java, que se parece a un array, pero con ciertas diferencias, usaremos en esta guía el término inglés array, para evitar confusiones o inconsistencias.

Se pueden definir un array como un tipo de variable que puede guardar muchos valores a la vez
Una variable sólo puede guardar un único valor a la vez: los array pueden almacenar muchos valores a la vez.

Un array llamado v de 8 enteros se podría dibujar así:

v	**4**	**5**	**1**	**4**	**3**	**7**	**8**	**1**
	0	1	2	3	4	5	6	7

Podemos decir que:

- V es el nombre del array
- Este array contiene 8 valores
- En negrita están los valores que contiene el array
- Los números de abajo son los índices del array
- Los índices son números naturales empezando siempre en cero
- Los índices se indican entre corchetes

Un array se declara:

tipo nombre [tamaño];

Este array se declaró:

int v[8];

Si hago: System.out.print("%i",v[5]); ¿qué devuelve?

Devuelve: 7

¿Cómo se colocó el 5 en la posición 1?

Haciendo: v[1]=5;

¿Qué muestra? a=6; System.out.print ("%i",v[a]);

Devuelve: 8

Nota:

La propiedad **length** devuelve el largo del array. Por lo tanto v.length en el caso anterior devuelve 8

Declaración

1. Se puede declarar de forma indistinta un array de las siguientes formas:

```
tipo[] nombre;
tipo nombre [];
```

Ej:

```
int [] m;
float [] temperatura;
```

o

```
int m[];
float temperatura[];
```

2. Después de haber declarado un array, el siguiente paso es crearlo o construirlo.
Crear un array significa reservar la cantidad de memoria necesaria para contener todos sus elementos y asignar al nombre del array una referencia a este bloque de memoria.
Esto se realiza:

```
nombre = new tipo [tamaño];
```

Ej:

```
m = new int[10];
temperatura = new float[30];
```

o todo de una sola vez:

```
int m[] = new int[10];
float temperatura = new float[30];
```

Cuando un array es declarado automáticamente se carga.
Si es numérico: con valor 0 en cada posición,
si es carácter con "" (valor "\u0000"),
si es booleano con false.

Inicializar

Podemos inicializar un array directamente con valores en el momento de crearlo de la siguiente manera:

```
float[] temperatura = {10.2F, 12.3F, 3.4F, 14.5F, 15.6F, 16.7F};
```

Recorrer

Consiste en ir de la primera posición a la última del array
Para ello se acostumbra usar un for.
Si fuera a listar el contenido de todo el array: lo recorro y lo muestro

```
for (int i=0;i< v.length;i++)
        System.out.print (v[i]);
```

Preste atención a i<v.length, ya que el signo de menor es indispensable para no ir hasta la posición 8 (del ejemplo anterior) ya que la última posición es 7 ya que la primera es la número 0 en un array de 8 posiciones.

Cargado Total

Consiste en cargar información en todo el array.
Para ello se recorre y se carga desde teclado.
Ej:

```
//se carga el array por teclado
Scanner dato=new Scanner(System.in);
for(int i=0;i<v.length;i++){
    System.out.println("Ingrese un numero en la posición "+i+"->");
    v[i]=dato.nextInt();
```

Ejercicio 40

Pruebe el cargado y el listado de este array

Asignación Directa

Puede asignar valores a un array con el signo de igual

Ej:

```
v[2]=123;
```

Puede vaciar el array por ejemplo con:

```
for (int i=0;i<v.length;i++)
    v[i]=0;
```

```
import java.util.*;

public class probandoArray {

   public static void main (String[] args) {

        // se declara el array
        int[] v=new int[8];

        // se carga manualmente
        v[0]=4;
        v[1]=5;
        v[2]=1;
        v[3]=4;
        v[4]=3;
        v[5]=7;
        v[6]=8;
        v[7]=1;

        //se recorre e imprime en pantalla
        for(int i=0;i<v.length;i++)
            System.out.println(v[i]);

        //se carga el array por teclado
        Scanner dato=new Scanner(System.in);
        for(int i=0;i<v.length;i++){
            System.out.println("Ingrese un numero en la posición "+i+"->");
            v[i]=dato.nextInt();
        }

        //se lista nuevamente el array
        for(int i=0;i<v.length;i++)
            System.out.println("En la posicion "+i+" esta el numero "+v[i]);

        }
}
```

Ejercicio 41

Cargue este array mediante asignación directa con números al azar del 1 al 100 y después lístelo.

```
INICIO
        Declarar el array
        Para i de 0 a 7 de 1 en 1
            v[i]=azar(100)
        Fin Para
        Para i de 0 a 7 de 1 en 1
            Mostrar v[i]
        Fin Para
    FIN
```

Constantes

Una constante es una variable del sistema que mantiene un valor inmutable a lo largo de toda la vida del programa. Las constantes en Java se definen mediante el modificador **final**.

La estructura sería:
static final nombreConstante = valor;

De esta forma si queremos definir las constantes DIAS_SEMANA ó DIAS_LABORABLES, que sabemos que son variables que no cambiarán su valor a lo largo del programa, generaremos el siguiente código:

```
static final int DIAS_SEMANA = 7;
static final int DIAS_LABORABLES = 5;
```

Se recomienda el uso de mayúsculas para las constantes, para visualizarlas mejor dentro del programa.

Si queremos utilizar una constante Java, simplemente deberemos de utilizar su nombre. Así, si queremos utilizar las anteriores constantes, lo haremos de la siguiente forma:

```
System.out.println("El número de días de la semana son " + DIAS_SEMANA);
System.out.println("El número de días laborables de la semana son " +
DIAS_LABORABLES);
```

El código completo del programa de constantes en Java será el siguiente:

```
public class ConstanteEnJava {
  static final int DIAS_SEMANA = 7;
  static final int DIAS_LABORABLES = 5;

  public static void main(String[] args) {

    System.out.println("El número de días de la semana son " + DIAS_SEMANA);
    System.out.println("El número de días laborables de la semana son " +
DIAS_LABORABLES);
```

```
  }
}
```

En este caso las hemos declarado static en la clase. Si bien podrían ir dentro del método main sin ser static:

```
public class ConstanteEnJava {

  public static void main(String[] args) {

    final int DIAS_SEMANA = 7;
    final int DIAS_LABORABLES = 5;

    System.out.println("El número de días de la semana son " + DIAS_SEMANA);
    System.out.println("El número de días laborables de la semana son " +
DIAS_LABORABLES);

  }
}
```

Tema 14: String

Un string es una cadena de caracteres o sea es un texto de 0 o más caracteres.
En Java se puede convertir el String en un array de caracteres de una forma similar a como lo hace el lenguaje C. Un array de caracteres y un String no es lo mismo. Se pueden convertir los datos de String a array de caracteres y viceversa.

Los strings u objetos de la clase String se pueden crear explícitamente o implícitamente. Para crear un string implícitamente basta poner una cadena de caracteres entre comillas dobles. Por ejemplo, cuando se escribe

```
System.out.println("El primer programa");
```

Java crea un objeto de la clase String automáticamente.

Para crear un string explícitamente escribimos

```
String str=new String("El primer programa");
```

También se puede escribir, alternativamente

```
String str="El primer programa";
```

Para crear un string nulo se puede hacer de estas dos formas

```
String str="";
String str=new String();
```

Un string nulo es aquél que no contiene caracteres, pero es un objeto de la clase String. Sin embargo,

```
String str;
```

está declarando un objeto str de la clase String, pero aún no se ha creado ningún objeto de esta clase.

Existe una clase llamada StringBuffer, que es similar a String.
Se usa String para cadenas estables y StringBuffer para cadenas variables.
En esta guía se usa mayoritariamente la clase String.

Cómo se obtiene información acerca del string

Una vez creado un objeto de la clase String podemos obtener información relevante acerca del objeto a través de las funciones miembro.

Para obtener la longitud, número de caracteres que guarda un string se llama a la función miembro length.

```
String str="El primer programa";
int longitud=str.length();
```

Podemos conocer si un string comienza con un determinado prefijo, llamando al método startsWith, que devuelve true o false, según que el string comience o no por dicho prefijo

```
String str="El primer programa";
boolean resultado=str.startsWith("El");
```

En este ejemplo la variable resultado tomará el valor true.

De modo similar, podemos saber si un string finaliza con un conjunto dado de caracteres, mediante la función miembro endsWith.

```
String str="El primer programa";
boolean resultado=str.endsWith("programa");
```

Si se quiere obtener la posición de la primera ocurrencia de la letra p, se usa la función indexOf.

```
String str="El primer programa";
int pos=str.indexOf('p');
```

Para obtener las sucesivas posiciones de la letra p, se llama a otra versión de la misma función

```
pos=str.indexOf('p', pos+1);
```

El segundo argumento le dice a la función indexOf que empiece a buscar la primera ocurrencia de la letra p a partir de la posición pos+1.

Otra versión de indexOf busca la primera ocurrencia de un substring dentro del string.

```
String str="El primer programa";
int pos=str.indexOf("pro");
```

Vemos que una clase puede definir varias funciones miembro con el mismo nombre pero que tienen distinto número de parámetros o de distinto tipo.

Comparación de strings

La comparación de strings nos da la oportunidad de distinguir entre el operador lógico == y la función miembro equals de la clase String. En el siguiente código

```
String str1="El lenguaje Java";
String str2=new String("El lenguaje Java");
 if(str1==str2)
        System.out.println("Los mismos objetos");
 else
        System.out.println("Distintos objetos");
 if(str1.equals(str2))
        System.out.println("El mismo contenido");
 else
        System.out.println("Distinto contenido");
```

Esta porción de código devolverá que str1 y str2 son distintos objetos pero con el mismo contenido. str1 y str2 ocupan posiciones distintas en memoria pero guradan los mismos datos.

Cambiemos la segunda sentencia y escribamos

```
String str1="El lenguaje Java";
String str2=str1;
 System.out.prinln("Son el mismo objeto "+(str1==str2);
```

Los objetos str1 y str2 guardan la misma referencia al objeto de la clase String creado. La expresión (str1==str2) devolverá true.

Asñi pues, el método equals compara un string con un objeto cualquiera que puede ser otro string, y devuelve true cuando dos strings son iguales o false si son distintos.

```
String str="El lenguaje Java";
 boolean resultado=str.equals("El lenguaje Java");
```

La variable resultado tomará el valor true.

La función miembro comapareTo devuelve un entero menor que cero si el objeto string es menor (en orden alfabético) que el string dado, cero si son iguales, y mayor que cero si el objeto string es mayor que el string dado.

```
 String str="Tomás";
 int resultado=str.compareTo("Alberto");
```

La variable entera resultado tomará un valor mayor que cero, ya que Tomás está después de Alberto en orden alfabético.

```
 String str="Alberto";
 int resultado=str.compareTo("Tomás");
```

La variable entera resultado tomará un valor menor que cero, ya que Alberto está antes que Tomás en orden alfabético.

Extraer un substring de un string

En muchas ocasiones es necesario extraer una porción o substring de un string dado. Para este propósito hay una función miembro de la clase String denominada substring.

Para estraer un substring desde una posición determinada hasta el final del string escribimos

```
String str="El lenguaje Java";
  String subStr=str.substring(12);
```

Se obtendrá el substring "Java".

Una segunda versión de la función miembro substring, nos permite extraer un substring especificando la posición de comienzo y la el final.

```
String str="El lenguaje Java";
  String subStr=str.substring(3, 11);
```

Se obtendrá el substring "lenguaje". Recuérdese, que las posiciones se empiezan a contar desde cero.

Convertir un número a string

Para convertir un número en string se emplea la función miembro estática valueOf (más adelante explicaremos este tipo de funciones).

```
int valor=10;
String str=String.valueOf(valor);
```

La clase String proporciona versiones de valueOf para convertir los datos primitivos: int, long, float, double.

Esta función se emplea mucho cuando programamos applets, por ejemplo, cuando queremos mostrar el resultado de un cálculo en el área de trabajo de la ventana o en un control de edición.

Convertir un string en número

Cuando introducimos caracteres en un control de edición a veces es inevitable que aparezcan espacios ya sea al comienzo o al final. Para eliminar estos espacios tenemos la función miembro trim

```
String str="  12 ";
String str1=str.trim();
```

Para convertir un string en número entero, primero quitamos los espacios en blanco al principio y al final y luego, llamamos a la función miembro estática

parseInt de la clase Integer (clase envolvente que describe los números enteros)

```
String str="  12 ";
int numero=Integer.parseInt(str.trim());
```

Para convertir un string en número decimal (double) se requieren dos pasos: convertir el string en un objeto de la clase envolvente Double, mediante la función miembro estática valueOf, y a continuación convertir el objeto de la clase Double en un tipo primitivo double mediante la función doubleValue

```
String str="12.35 ";
double numero=Double.valueOf(str).doubleValue();
```

Se puede hacer el mismo procedimiento para convertir un string a número entero

```
String str="12";
int numero=Integer.valueOf(str).intValue();
```

Scanner y String

La primera salvedad que tenemos que hacer cuando utilizamos el método next() es que solo nos permite ingresar una cadena de caracteres con la excepción del espacio en blanco (es decir debemos ingresar un nombre de persona y no su nombre y apellido separado por un espacio en blanco)
existe otro método llamado nextLine() que nos permite cargar espacios en blanco pero para su uso se complica cuando cargamos otras valores de tipo distinto a String (por ejemplo int, float etc.)
Si deseamos separar el dato ingresado por algún carácter existe el método useDelimiter("delimitador") para indicar el carácter límite del ingreso de datos.

Ejercicio 42:
Solicitar el ingreso del apellido, nombre y edad de dos personas. Mostrar el nombre de la persona con mayor edad. Realizar la carga del apellido y nombre en una variable de tipo String.

```
import java.util.Scanner;

public class CadenaDeCaracteres2 {
   public static void main(String[] ar) {
      Scanner teclado=new Scanner(System.in);
      String apenom1,apenom2;
      int edad1,edad2;
      System.out.print("Ingrese el apellido y el nombre:");
      apenom1=teclado.nextLine();
      System.out.print("Ingrese edad:");
      edad1=teclado.nextInt();
      System.out.print("Ingrese el apellido y el nombre:");
      teclado.nextLine();
      apenom2=teclado.nextLine();
      System.out.print("Ingrese edad:");
      edad2=teclado.nextInt();
```

```
        System.out.print("La persona de mayor edad es:");
        if (edad1>edad2) {
           System.out.print(apenom1);
        } else {
           System.out.print(apenom2);
        }
    }
}
```

Cuando se ingresa una cadena con caracteres en blanco debemos tener en cuenta en llamar al método nextLine()
Una dificultad se presenta si llamamos al método nextLine() y previamente hemos llamado al método nextInt(), esto debido a que luego de ejecutar el método nextInt() queda almacenado en el objeto de la clase Scanner el caracter "Enter" y si llamamos inmediatamente al método nextLine() este almacena dicho valor de tecla y continúa con el flujo del programa. Para solucionar este problema debemos generar un código similar a:

```
System.out.print("Ingrese edad:");
edad1=teclado.nextInt();
System.out.print("Ingrese el apellido y el nombre:");
teclado.nextLine();
apenom2=teclado.nextLine();
```

Como vemos llamamos al método nextLine() dos veces, la primera retorna la tecla "Enter" y la segunda se queda esperando que ingresemos el apellido y nombre (tener en cuenta que esto es necesario solo si previamente se llamó al método nextInt() o nextFloat().

Ejercicio 43: Haga un programa de saludo

Se debe pedir el nombre de la persona y saludarla:
Ej:

```
Ingrese su nombre: Juan
Bienvenido Juan
```

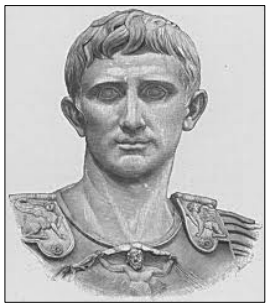
```
import java.util.*;
public class saludo {
  public static void main(String args[]){
          String nombre=new String();
          Scanner s=new Scanner(System.in);
          System.out.print("Ingrese su nombre ");
          nombre=s.next();
          System.out.println("Bienvenido "+nombre);

  }
}
```

Ejercicio 44: Modifique el programa anterior de manera que a una persona la salude de manera diferente.

Esto requiere consultar si el nombre es algún valor en particular.
También trabaje concatenar el mensaje con alguna frase

```
import java.util.*;
public class saludo {
  public static void main(String args[]){
     String nombre=new String();
      Scanner s=new Scanner(System.in);
      System.out.print("Ingrese su nombre ");
      nombre=s.next();
      if(nombre.equals("juan"))
        System.out.println("No te olvides "+nombre+" de pasar por Bedelía");
     else
        System.out.println("Bienvenido "+nombre);

  }
}
```

Ejercicio 45: Hacer el código del César



El César usaba un método que consistía agregar n caracteres más a su mensaje codificado.
Ejemplo:

A n a
+2
C p c

El ejercicio consiste en ingresar una palabra y luego mostrarla codificada mediante este método.
Aproveche la ocasión para hablarles a los alumnos de criptografía, sus utilidades y beneficios.

```
import java.util.Scanner;

public class cesar {
  public static void main(String args[]){
    //FRASE: variable tipo string que va a contener la frase a codificar
    //no necesito construirla, ya que Scanner le asignará memoria
    String frase;
    //se construye la variable S del tipo scanner
    Scanner s=new Scanner(System.in);

    System.out.print("Ingrese una frase ");
    //se carga en frase una linea de texto (puede contener espacios)
    frase=s.nextLine();
    //cargo la FRASE en un array de caracteres
    char[] codigo = frase.toCharArray();
    //se recorre el array
    for (int i=0;i<codigo.length;i++)
      //se le carga un caracter que esta 2 lugares mas adelante
      //es necesario CASTear a CHAR ya que el resultado es un int
      codigo[i]=(char)(codigo[i]+2);
      //se muestra el array de CHAR usando VALUEOF de la clase STRING
      //sino hay que mostrar el array caracter a caracter
```

```
    System.out.println("La palabra codificada es: "+String.valueOf(codigo));
  }
}
```

Reverse

Es un método que permite invertir un string.
Este método pertenece a la clase StringBuffer que es similar a String; StringBuffer está pensado para Strings variables

```
import java.util.*;

public class reverse {
  public static void main(String args[]){
        String frase;
        Scanner dato=new Scanner(System.in);
        System.out.print("Ingrese una frase");
        frase=dato.nextLine();
        StringBuffer inversa=new StringBuffer(frase).reverse();
        System.out.println("al reves se lee:"+inversa);
  }
}
```

Ejercicio 46: Realice el programa dec2bin. Debe ingresar un número en decimal y convertirlo a binario
El número ingresado es un int.
La conversión se carga en un array de String.

Trabaje con los alumnos el método de conversión de decimal a binario. Guarde la conversión en una variable String para no perder el cero a la izquierda, cuando sea necesario. Use el método REVERSE visto en el ejercicio anterior.

```
import java.util.*;

public class dec2bin {
  public static void main(String args[]){
        int num;
        String bin="";
        Scanner dato=new Scanner(System.in);
        System.out.print("Ingrese un numero :");
        num=dato.nextInt();
        while(num>0){
              int resto=num%2;
              bin=bin+String.valueOf(resto);
              num/=2;
        }
        StringBuffer inversa=new StringBuffer(bin).reverse();
        System.out.println("al reves se lee:"+inversa);
```

```
  }
}
```

Ejercicio 47: Realice el programa bin2dec para convertir de binario a decimal

Trabaje con los alumnos el algoritmo de conversión binario-decimal.
Analice cada método usado en este programa y su aplicación en dicho algoritmo.

```
import java.util.*;

public class bin2dec {
  public static void main(String args[]){
        int num=0,c=0;
        String sBin;
        char [] cBin;
        Scanner dato=new Scanner(System.in);
        System.out.print("Ingrese un numero binario:");
        sBin =dato.next();
        cBin=sBin.toCharArray();

        for(int i=cBin.length-1;i>=0;i--)
              num+=Math.pow(2,c++)*(int)(cBin[i]-48);

        System.out.println("El equivalente decimal es:"+num);
  }
}
```

Tratamiento de Excepciones

En Java, como en cualquier otro lenguaje de programación, pueden existir situaciones en la que el programa falle. Java llama a estos errores excepciones.
Para tratar excepciones deberemos usar la instrucción **try** (que significa intentar en español).

```
try {
   //declaración que causa la excepción
}
```

Entre las llaves de try escribiremos el código que hará funcional nuestro programa. Para capturar la excepción que puede generar este código necesitaremos otra instrucción llamada **catch** (capturar).

```
catch(NombredeExcepcion obj){
  //código para tratar el error
}
```

Entre las llaves de catch escribiremos el código que queramos para tratar el error.

Ejemplo:

```
class test{
   public static void main(String args[]){
   int numero;
   String cadena="  1";
   try{
              numero = Integer.parseInt(cadena);
   }
   catch(NumberFormatException ex){
   System.out.println("No es un número, es una cadena de texto.");
  }
 }
}
```

En este ejemplo tratamos de convertir una cadena de un número a un entero. Pero la variable cadena contiene un espacio en blanco, lo que provocará un error de ejecución que trataremos mostrando por pantalla el mensaje de error.

Tema 15: Objetos dentro de array

Calificadores: Public, Private, Protected, Static y Final.

El uso de calificadores de acceso en Java tiene sus bases en el uso de librerías ("packages"). Al ser diseñado un programa existen diversas funciones/métodos y variables dentro de éste, algunas de estas requerirán ser modificadas conforme incrementen las necesidades del programa, mientras otras permanecerán inmóviles, la importancia de estos cambios requiere que sean utilizados calificadores para permitir/negar el acceso a ciertos segmentos del programa.

Por ejemplo:

> Usted diseñó 300 clases que están siendo re-utilizadas por otros programas, sin embargo, se le ha solicitado una modificación radical a estos objetos base, ¿cuales métodos/campos de estas 300 Clases puede modificar sin quebrantar los otros programas que hacen uso de estas? Aquí puede surgir un serio problema sino han sido utilizados los calificadores de acceso adecuadamente.

Además de los calificadores de acceso que permiten restringir el uso de métodos/campos a determinadas situaciones, también existen otros calificadores que afectan directamente la creación y uso de instancias por clase estos calificadores son **static** y **final**.

public: Acceso libre .

El uso del calificador *public* significa que toda definición será accesible de cualquier punto, ya sea un método, campo o clase. Su uso implica un acceso global. Se sugiere no usar en elementos que se modifican frecuentemente.

private: Solo en la misma Clase .

El calificador *private* indica que dicho componente será accesible **únicamente dentro de la Clase en cuestión**, si se intenta acceder cualquier elemento de este tipo dentro de otra Clase será generado un error de compilación.

El calificador *private* suele utilizarse en Clases que serán modificadas continuamente, esto permite evitar futuros quebrantos en otras Clases.

protected: Clases Heredadas y misma Clase.

El uso de *protected* es utilizado bajo los conceptos de Herencias de Clase (tema a tratar en segundo año). Mediante *protected* es posible acceder elementos de la Clase Hereditaria , aunque no aquellos que utilicen el calificador *private*.

En otras palabras, si determinada Clase *hijo* hereda el comportamiento de una Clase *padre*, la Clase *hijo* tendrá acceso a todos aquellos campos/métodos definidos como protected en *padre*, pero no aquellos declarados como *private* en *padre*.

Ningún Calificador: Clase en Librería y misma Clase.

Finalmente, cuando no es empleado ninguno de los calificadores de acceso mencionados anteriormente los elementos son considerados *amigables*, esto

implica que todo campo/método carente de calificador será accesible dentro de todas Clases pertenecientes a su misma librería ("package").

***static : Una sola* instancia .**

El uso de *static* ha sido utilizado en los métodos principales (main) de los programas escritos anteriormente, su uso esta relacionado directamente al uso de **instancias** en Clases; en ocasiones es necesario o conveniente generar elementos que tomen un mismo valor para cualquier número de **instancias** generadas o bien invocar/llamar métodos sin la necesidad de generar **instancias**, y es bajo estas dos circunstancias que es empleado el calificador *static*.

En el caso del método main Java, éste puede ser llamado automáticamente al invocarse la respectiva Clase debido a que no se encuentra asociado con ningún tipo de instancia, esto implica que su comportamiento siempre será el mismo independientemente de la instancia que realza su llamada.

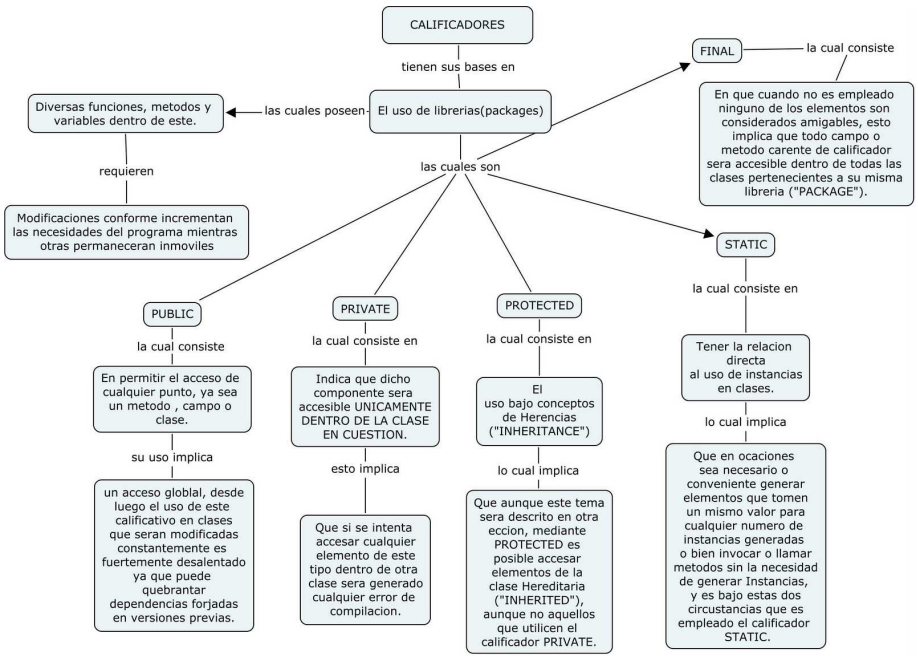
Dos aspectos característicos de utilizar el calificador static en un elemento Java son los siguientes :

- No puede ser *generada* ninguna instancia (Uso de new) de un elemento static puesto que solo existe una instancia.
- Todos los elementos definidos dentro de una estructura static deben ser static ellos mismos, o bien, poseer una **instancia** ya definida para poder ser *invocados*.
- NOTA: Lo anterior no implica que no puedan ser *generadas* instancias dentro de un elemento static; no es lo mismo *llamar/invocar* que *crear/generar*.

***final : Una sola* instancia *y* definitiva.**

El calificador final implica una asignación única y definitiva al elemento de una clase. A diferencia de static que implica una sola instancia, el término final lleva dicha instancia a una definición única. Se usa para la definición de constantes.

Generalmente es utilizado en aquellos elementos que no serán modificados al momento de ejecutarse ("Run-Time") un programa o clase logrando un nivel de *eficiencia* en ejecución.



Arrays con clases

Ejercicio 48
Programa actualizaciones en un array de 10 posiciones.
Cada posición del array debe contener el nombre y nota de un alumno.

Posible solución:
1. Primero defina la clase y sus variables.

Esto se hace antes del main() y sin calificadores para la clase.

```
class cAlumnos{
        String nombre;
        int nota;
```

Esto quiere decir que de cada alumno va a almacenar su **nombre** y **nota**.

Defina métodos (acciones) que necesite para estos datos.
Analícelos como datos aislados (1 solo alumno), aún no piense en el conjunto.
Se incluye el objeto Scanner porque es necesario al cargar un dato.

```
Scanner in=new Scanner(System.in);
//el método cargar() no puede ser declarado static
void cargar(){
        System.out.print("Ingrese nombre del alumno ");
        nombre=in.nextLine();
        System.out.print("Ingrese la nota del alumno ");
        nota=in.nextInt();
}
void mostrar(){
        System.out.println("Nombre del alumno "+nombre);
        System.out.println("Nota del alumno ");
}
```

Dentro del método principal declarar el array con n elementos de la clase definida.

```
static final int CANTIDAD=10;
//cAlumnos es la claseAlumnos
public static void main(String a[]){
        cAlumnos[] alumnado=new cAlumnos[CANTIDAD];
```

En este ejemplo se usa la constante CANTIDAD para definir el largo del array.
Se considera que genera una mejor lectura del programa.
Además cuando se hace el testing se cambiar el valor de la constante para probarlo rápidamente con más o menos posiciones.
Recorrer el array e invocar a la clase y sus métodos.

```
        //Cargar todo el array
        for (int i=0;i<alumnado.length;i++){
                alumnado[i]= new cAlumnos();
                alumnado[i].cargar();
        }
        //listar el vector
        for (int i=0;i<alumnado.length;i++){
                System.out.print("Posición : "+i+"->");
                alumnado[i].mostrar();
```

```
            }
        }
```

Ejemplo completo:

```
import java.util.*;
//defina la clase alumno fuera de la clase principal sin ningun modificador

class cAlumnos{
        String nombre;
        int nota;
        Scanner in=new Scanner(System.in);
        //el método cargar() no puede ser declarado static
        void cargar(){
                System.out.print("Ingrese nombre del alumno ");
                nombre=in.nextLine();
                System.out.print("Ingrese la nota del alumno ");
                nota=in.nextInt();
        }
        void mostrar(){
                System.out.println("Nombre del alumno "+nombre);
                System.out.println("Nota del alumno ");
}
}


public class alumnosynotas{
        //la constante debe ser static para ser vista desde todo el programa
        static final int CANTIDAD=10;
        //cAlumnos es la claseAlumnos
        public static void main(String a[]){
                cAlumnos[] alumnado=new cAlumnos[CANTIDAD];

                //Cargar todo el array
                for (int i=0;i<alumnado.length;i++){
                        alumnado[i]= new cAlumnos();
                        alumnado[i].cargar();
                }

                //listar el vector
                for (int i=0;i<alumnado.length;i++){
                        System.out.print("Posición : "+i+"->");
                        alumnado[i].mostrar();
                }
        }
}
```

Ejercicio 49:
Modificar el programa anterior, para que desde un menú se actualice el vector que contiene 10 nombres y notas de alumnos.
Dicho menú tendrá las opciones:
Cargado Total
Listado
Cargar 1 solo alumno
Mostrar 1 solo alumno
Modificar 1 alumno
Vaciado
Cada opción del menú debe tener un método asociado.

Posible Solución:

```
import java.util.*;
//Usada en la pausa
import java.io.IOException;

//defina la clase alumno fuera de la clase principal sin ningun modificador
class claseAlumnos{
	String nombre;
	int nota;
	Scanner in=new Scanner(System.in);
	//el método cargar() no puede ser declarado static
	void cargar(){
		System.out.print("Ingrese nombre  ");
		nombre=in.nextLine();
		while(true){
		System.out.print("Ingrese nota    ");
		nota=in.nextInt();
		if (nota >=1 && nota <=12)
			break;
		else
			System.out.println("La nota debe ser entre 1 y 12");
		}
	}

	void mostrar(){
		System.out.println("Nombre del alumno "+nombre+"\t Nota "+nota);
	}
	String sacarNombre(){
		return nombre;
	}
	int sacarNota(){
		return nota;
	}
	void ponerNombre(String n){
		nombre=n;
	}
```

```
	void ponerNota(int n){
		nota=n;
	}
	void vaciar(){
		nota=0;
		nombre="";
	}
}


public class menuAlumnosYnotas{
	//la constante debe ser static para que se pueda ver desde todo el
programa
	static final int CANTIDAD=5;
	//Declarar el array. Se hace antes del main para que sea visible
	//desde los demás métodos
	static claseAlumnos[] alumnado=new claseAlumnos[CANTIDAD];
	static Scanner in=new Scanner(System.in);
	public static void main(String a[]){
		//Construir el array
		for (int i=0;i<alumnado.length;i++)
			alumnado[i]= new claseAlumnos();
		//llamar al menu
		menu();
	}

	static void menu(){
		int opcion=0;
		do {
			System.out.println("1 - Cargado Total");
			System.out.println("2 - Listado");
			System.out.println("3 - Cargar 1 solo alumno");
			System.out.println("4 - Mostrar 1 solo alumno");
			System.out.println("5 - Modificar 1 alumno");
			System.out.println("6 - Vaciado");
			System.out.println("0 - Terminar");
			System.out.print("ingrese una opcion :");
			opcion=in.nextInt();
			switch(opcion){
				case 1: cargadoTotal();break;
				case 2:	listado();break;
				case 3:	cargar1();break;
				case 4:	mostrar1();break;
				case 5:	modificar1();break;
				case 6:	vaciado();break;
			}
		} while (opcion !=0);
	}
```

```
	static void cargadoTotal(){
		//Cargar todo el array
		for (int i=0;i<alumnado.length;i++){
			System.out.print("Alumno : "+i+"->");
			alumnado[i].cargar();
		}
	}

	static void listado(){
		//listar el vector
		for (int i=0;i<alumnado.length;i++){
			System.out.print("Posición : "+i+"->");
			alumnado[i].mostrar();
		}
		pausa();
	}

	static void cargar1(){
		System.out.print("Ingrese la posición a cargar (0-
"+(alumnado.length-1)+") ");
		int pos=in.nextInt();
		if (pos>=0 && pos<alumnado.length)
			alumnado[pos].cargar();
		else
			System.out.println("Posición inválida");
		pausa();
	}

	static void mostrar1(){
		String nom;
		System.out.print("Ingrese el nombre del alumno :");
		nom=in.next();
		boolean esta=false;
		for (int i=0;i<alumnado.length;i++)
			if (nom.equals(alumnado[i].sacarNombre())){
				System.out.println("Esta en la posicion :"+i);
				System.out.println("Su nota es
:"+alumnado[i].sacarNota());
				esta=true;
		}
		if (!esta)
			System.out.println("Alumno no encontrado");
		pausa();
	}

	static void modificar1(){
		System.out.print("Ingrese la posición a modificar :");
		int pos=in.nextInt();
		if (pos>=0 && pos <alumnado.length){
```

```
                System.out.println("Nombre actual
:"+alumnado[pos].sacarNombre());
                System.out.print("Ingrese nuevo nombre :");
                String nom=in.next();
                alumnado[pos].ponerNombre(nom);
                System.out.println("Nota actual:"+alumnado[pos].sacarNota());
                System.out.print("Ingrese nueva nota :");
                int nota=in.nextInt();
                alumnado[pos].ponerNota(nota);
            }else
                System.out.println("Posicion invalida...");
            pausa();
        }
        static void vaciado(){
            System.out.println("De ENTER si no desea vaciar:");
            System.out.print("Escriba VACIAR para confirmar :");
            String clave=in.next();
            if (clave.equals("VACIAR")){
                for (int i=0;i<alumnado.length;i++)
                    alumnado[i].vaciar();
                System.out.println("Array vacio");}
            else
                System.out.println("Vaciado cancelado");
            pausa();
        }
        static void pausa(){
            System.out.print("Pulse ENTER para continuar");
            try {
                System.in.read(); // Lee del teclado
                } catch (IOException  error) {

                }

        }
}
```

En la solución antes se hace un uso básico de las clases. A continuación se hace un uso más profundo.
Tal vez por el nivel de dificultad y abstracción, el siguiente ejemplo sea más aplicable al curso de Programación 2:

Solución Mejorada:

```
import java.util.*;
//Usada en la pausa
import java.io.IOException;

//defina la clase alumno fuera de la clase principal
class alumno{
	private String nombre;
	private int nota;
	alumno(String nom, int not ){
		nombre=nom;
		nota=not;
	}
	String obtenerNombre(){
		return nombre;
	}
	int obtenerNota(){
		return nota;
	}
	void asignarNombre(String n){
		nombre=n;
	}
	void asignarNota(int n){
		nota=n;
	}
	void vaciar(){
		nota=0;
		nombre="";
	}
}

//Esta clase representa la coleccion de alumnos
class clase{
	alumno[] alumnado;
	int cont;
	clase(int cantidad){
		alumnado=new alumno[cantidad];
		cont=0;
	}
	void agregarAlumno(alumno a){
		alumnado[cont]=a;
		cont++;
	}
```

```
	//precondicion: el alumno debe existir
	alumno obtenerAlumno(String nombre){
		alumno aux=null;
		boolean encontrado=false;
		int i=0;
		while (i<cont && !encontrado){
			if(nombre.equals(alumnado[i].obtenerNombre())){
				aux=alumnado[i];
				encontrado=true;
			}
			i++;
		}
		return aux;
	}
	// La posicion debe existir
	alumno obtenerPorPosicion(int pos){
		return alumnado[pos];
	}
	boolean existeAlumno(String nombre ){
		boolean encontrado=false;
		int i=0;
		while (i<cont && !encontrado){
			if(nombre.equals(alumnado[i].obtenerNombre())){
				encontrado=true;
			}
		}
		return encontrado;

	}
	int obtenerTope(){
		return cont;
	}
	void vaciarClase(){
		for(int i=0; i<cont; i++)
			alumnado[i].vaciar();
	}
}

//**La clase Principal**
public class menuAlumnosYnotasMejorado{
//la constante debe ser static para que se pueda ver desde todo el programa
	static final int CANTIDAD=5;
	static clase cl;
	static Scanner in;
	public static void main(String a[]){
		cl= new clase(CANTIDAD);
		in=new Scanner(System.in);
		menu();
	}
```

```
	static void menu(){
		int opcion=0;
		do {
			System.out.println("MENU DE NOTAS Y ALUMNOS MEJORADO");
			System.out.println("1 - Cargado Total");
			System.out.println("2 - Listado");
			System.out.println("3 - Cargar 1 solo alumno");
			System.out.println("4 - Mostrar 1 solo alumno");
			System.out.println("5 - Modificar 1 alumno");
			System.out.println("6 - Vaciado");
			System.out.println("0 - Terminar");
			System.out.print("ingrese una opcion :");
			opcion=in.nextInt();
			switch(opcion){
				case 1: cargadoTotal();break;
				case 2:	listado();break;
				case 3:	cargarUno();break;
				case 4:	mostrarUno();break;
				case 5:	modificarUno();break;
				case 6:	vaciado();break;
			}
		} while (opcion !=0);
	}

	static void cargadoTotal(){
		//Cargar todo el array
		for (int i=cl.obtenerTope();i<CANTIDAD;i++){
			cargarUno();
		}
	}

	static boolean validarNota(int n){
		if (n >=1 && n <=12)
			return true;
		else
			return false;
	}

	static void listado(){
		//listar el vector
		for (int i=0;i<cl.obtenerTope();i++){
			System.out.print("Posición : "+i+"->"+ "\nNombre: " +
cl.obtenerPorPosicion(i).obtenerNombre() + "\nNota: "+
cl.obtenerPorPosicion(i).obtenerNota() +"\n\n");
		}
		pausa();
	}

	static void cargarUno(){
```

```
            alumno aux;
            int nota;
            boolean correcta;
            System.out.println("Ingrese el nombre del alumno :");
            String nombre=in.next();

            do{
                  System.out.println("Ingrese la nota del mismo");
                  nota=in.nextInt();
                  if(!validarNota(nota)){
                        System.out.println("La nota debe estar comprendida
entre 1 y 12");
                        correcta=false;
                  }
                  else{
                        correcta=true;
                  }

            }while(!correcta);
            aux=new alumno(nombre,nota);
            cl.agregarAlumno(aux);

            pausa();
      }

      static void mostrarUno(){
            alumno aux;
            System.out.print("Ingrese el nombre del alumno :");
            String nom=in.next();
            if(cl.existeAlumno(nom)){
                  aux=cl.obtenerAlumno(nom);
                  System.out.println("La nota del alumno es:
"+aux.obtenerNota());
            }
            else
                  System.out.println("Alumno no encontrado");
            pausa();
      }

      static void modificarUno(){
            System.out.print("Ingrese la posición del alumno a modificar :");
            int pos=in.nextInt();
            if (pos>=0 && pos <cl.obtenerTope()){
                  System.out.println("Nombre actual :"+
cl.obtenerPorPosicion(pos).obtenerNombre());
                  System.out.print("Ingrese nuevo nombre :");
                  String nom=in.next();
                  cl.obtenerPorPosicion(pos).asignarNombre(nom);
```

```
                System.out.println("Nota actual :" +
cl.obtenerPorPosicion(pos).obtenerNota());
                System.out.print("Ingrese nueva nota :");
                int nota=in.nextInt();
                cl.obtenerPorPosicion(pos).asignarNota(nota);
            }else
                System.out.println("Posicion invalida...");
            pausa();
        }

        static void vaciado(){
            System.out.println("De ENTER si no desea vaciar:");
            System.out.print("Escriba VACIAR para confirmar :");
            String clave=in.next();
            if (clave.equals("VACIAR")){
                cl.vaciarClase();
                System.out.println("Array vacio");
            }
            else
                System.out.println("Vaciado cancelado");
            pausa();
        }

        static void pausa(){
            System.out.print("Pulse ENTER para continuar");
            try {
                System.in.read(); // Lee del teclado
                } catch (IOException  error) {

                }

        }
}
```

Tema 16: Obligatorio y defensa

Hasta este tema, existe una forma de enseñanza incremental, donde a cada tema le sigue uno de mayor complejidad, pero le falta algo indispensable para el curso de programación que es la necesidad de integrar los conocimientos adquiridos.
Para ello el docente debe proponer cerca de fin de año un trabajo Obligatorio.
Le llamamos Obligatorio a un programa que el estudiante debe realizar en equipo, en base a una propuesta, la cual debe entregar, documentar y defender a fin de año.
No le llamamos proyecto porque le faltan elementos para llegar a ello, en el diseño y avance del mismo. Dejamos el proyecto para años posteriores donde el estudiante está más maduro para estas tareas.
Equipo:
El estudiante debe formar un equipo de 1 a 3 estudiantes.
A pesar que un equipo de 1 está mal definido, en este primer año se aceptará esta opción.
Propuesta:
Para lograr resultados aceptables se sugiere proponer la letra del obligatorio después de vacaciones de primavera.
Esto requiere que antes de octubre esté casi terminado el programa, de manera de apoyar el mismo.
El apoyo consiste en dictar temas que permitan realizar eficientemente el obligatorio y atender en clase las consultas de los alumnos.

Entrega:
Se debe fijar una fecha de entrega del proyecto.
La misma generalmente es en la última clase de octubre.
Avance:
El alumno a medida que va avanzando en su obligatorio lo puede (y debe) ir llevando a clase para que el docente lo oriente.
Defensa:
Por lo general en noviembre se hace la defensa del obligatorio.
El docente hace pasar a la clase a los equipos, uno a uno, prueba el programa y les va preguntando a cada integrante ¿qué hace? ¿para qué sirve? ¿cómo solucionaría los errores del programa presentado? De manera de verificar el grado de participación en el mismo, el nivel de conocimientos de programación y si está preparado para cursar Programación 2.

Quien no presente el obligatorio debe ir a examen.

Reglas
Se acostumbra ajustar la entrega a ciertas reglas como ser:

1) No se aceptan variables en inglés. Esto está fundado en estudiantes que buscan soluciones en Internet que estan en ingles, la copian tal cual y la presentan.

2) Los alumnos deben presentar el código del programa impreso ya que eso es una documentación del trabajo. Debe figurar los nombres de los integrantes
3) Si se detecta copia en los trabajos, quienes copian y quienes se dejaron copiar van a examen.
4) Presentar el obligatorio y presentarse a defenderlo son condiciones para no ir a examen.

Ejemplo de Obligatorio

Passwords Aceptables

Un password seguro es algo delicado. Los usuarios prefieren passwords que sean fáciles de recordar (como amigo), pero este password puede ser inseguro. Algunos lugares usan un generador randómico de passwords (como xvtpzyo), pero los usuarios toman demasiado tiempo recordándolos y algunas veces lo escriben en una nota pegada en su computador. Una solución potencial es generar password “pronunciables” que sean relativamente seguros pero fáciles de recordar.
PassUTU está desarrollando un generador de passwords.
Su trabajo en el departamento de control de calidad es probar el generador y asegurarse de que los passwords sean aceptables. Para ser aceptable, el password debe satisfacer estas tres reglas:
1. Debe contener al menos una vocal.
2. No debe tener tres vocales consecutivas o tres consonantes consecutivas.
3. No debe tener dos ocurrencias consecutivas de la misma letra, excepto por ‘ee’ o ‘oo’.
(Para el propósito de este problema, las vocales son 'a', 'e','i', 'o', y 'u'; todas las demás letras son consonantes.)
Note que estas reglas no son perfectas; habrán muchas palabras comunes/pronunciables que no son aceptables.
La entrada consiste en una o más potenciales passwords, uno por línea, seguidas por una línea conteniendo una palabra 'fin' que señala el fin de la entrada. Cada password tiene como mínimo una y como máximo veinte letras de largo y esta formado solo por letras en minúscula. Por cada password, despliegue si es o no aceptable, usando el formato mostrado en el ejemplo de salida.

Ejemplo de entrada
a
tv
ptoui
bontres
zoggax
wiinq
eep
houctuh
fin

Ejemplo de salida
<a> es aceptado.
<tv> no fue aceptado.
<ptoui> no fue aceptado.
<bontres> no fue aceptado.
<zoggax> no fue aceptado.
<wiinq> no fue aceptado.
<eep> es aceptado.
<houctuh> es aceptado.

Solución

Este problema presenta tres simples reglas que se deben cumplir para que un password sea aceptable.
Primero es necesario identificar si un carácter es vocal o no, para lo cual utilizo la siguiente función que pregunta si el carácter es una vocal, y devuelve true por verdad y false por falso.

```
int esVocal( char ch ){
  if( ch=='a' || ch=='e' || ch=='i'|| ch=='o' || ch=='u' )
  return true;
return false;
}
```

Ahora iremos validando cada una de las tres condiciones expuestas en el planteamiento del problema en su respectivo orden.

Debe contener al menos una vocal.- Basta con realizar un recorrido por toda la cadena (palabra), en cuanto encontremos una vocal se retorna **true** y si se ha terminado de hacer
el recorrido y no se ha encontrado ninguna vocal retornamos **false**.

```
int regla_1(){
        int i;
        for( i=0; i<len; i++ )
                if( esVocal( palabra[i] ) )
                return true;
        return false;
}
```

No debe tener tres vocales consecutivas o tres consonantes consecutivas.-

Otra vez un recorrido por toda la cadena, pero esta vez utilizando dos contadores **v** y **c**, que se incrementan en cuanto se encuentra una vocal o consonante respectivamente, en cuanto alguno llegue a tres, la función termina con falso, y si logra terminar el recorrido sin problemas se da por cumplida la segunda regla.

```
int regla_2(){
        int i, v=0, c=0;
        for( i=0; i<len; i++ ){
                if( esVocal( palabra[i] ) ){
                  v++; c=0;}
                else{
                  c++; v=0;
                }
                if( v==3 || c==3 )
                  return false;
        }
return true;
}
```

No debe tener dos ocurrencias consecutivas de la misma letra, excepto por "ee" o "oo". Otro recorrido más, esta función, como las anteriores es muy explícita, la única diferencia es que empieza en el segundo elemento de la cadena y no así en el primero.

```
int regla_3(){
        int i;
        for( i=1; i<len; i++ ){
                if( palabra[i]==palabra[i-1] && palabra[i]!='e' && palabra[i]!='o' )
                return false;
        }
        return true;
}
```

Bien, ahora solo resta leer cadenas, verificamos si cumplen las tres reglas e imprimir el resultado correspondiente, en caso de haber leído la cadena 'fin', termina el programa.

```
import java.util.Scanner;
public class passw{
   static char palabra[];
   static int largo;

   static boolean esVocal( char letra ){
      if( letra=='a' || letra=='e' || letra=='i'
            || letra=='o' || letra=='u' )
         return true;
      return false;
   }

   static boolean regla_1(){
      for( int i=0; i<largo; i++ )
         if( esVocal( palabra[i] ) )
            return true;
      return false;
   }
```

```
    static boolean regla_2(){
        int i, v=0, c=0;
        for( i=0; i<largo; i++ ){
            if( esVocal( palabra[i] ) ){
                v++; c=0;
            }else{
                c++; v=0;
            }
            if( v==3 || c==3 )
                return false;
        }
        return true;
    }

    static boolean regla_3(){
        for( int i=1; i<largo; i++ ){
            if( palabra[i]==palabra[i-1]
                    && palabra[i]!='e' && palabra[i]!='o' )
                return false;
        }
        return true;
    }
    public static void main( String args[] ){
        String linea;
        Scanner in = new Scanner( System.in );
        while( true ){
            System.out.print("Ingrese un password ('fin'-termina) ");
            linea = in.next();
            if( linea.equals("fin") )
                break;
            palabra = linea.toCharArray();
            largo = linea.length();
            if(regla_1()&& regla_2() && regla_3())
                System.out.println( "<" + linea + "> es aceptado." );
            else
                System.out.println( "<" + linea + "> no fue aceptado.." );
        }
    }
}
```

¿Es posible hacerlo todo con una única función?

Claro, existen muchas formas de hacer las cosas.

```
int reglas_1_2_3(){
        int i, v=0, c=0, vocales=0;
        for( i=0; i<len; i++ ){
                if( palabra[i]=='a' || palabra[i]=='e'
                || palabra[i]=='i' || palabra[i]=='o'
                || palabra[i]=='u' ){
                v++; c=0; vocales++;
                }else{
                c++; v=0;
        }
        if( v==3 || c==3 )
                return 0;
        if( palabra[i]==palabra[i+1]&& palabra[i]!='e' && palabra[i]!='o' )
                return 0;
        }
        return vocales;
}
```

Esta función es más complicada y más difícil de entender. Se sugiere siempre hacer códigos legibles y depurables.

BIBLIOGRAFÍA

- **[Arnold y Gosling, 1997] Ken Arnold y James Gosling. Addison-Wesley/Domo.** "*El lenguaje de Programación Java*". Wesley Iberoamericana. 1997. 334 páginas. (*Muy básico, escrito por el desarrollador del lenguaje*).
- **[Cortés et al.,1996] José Luis Cortés, Nuria González, Virginia Pérez, Paqui Villena y Ana Rosa Freire**. "*Java, el lenguaje de programación de Internet*". Data Becker 1996.
- **[Cuenca, 1996] Pedro Manuel Cuenca Jiménez.** "*Programación en Java para Internet*". Anaya Multimedia. 1996.
- **[Cuenca, 1997] Pedro Manuel Cuenca Jiménez,.** "*Programación en Java*". Ediciones Anaya Multimedia. 1997.
- **[Eckel, 1997] Bruce Eckel**. "*Hands -on Java Seminar*". Presindent MindView Inc. 1997. 577 páginas. (*Tutorial completo en Inglés en formato PDF*).
- **[Framiñán, 1997] José Manuel Framiñán Torres**. "*Manual Imprescindible de Java*". Ediciones Anaya Multimedia. 1997.
- **[Froufe, 1997] Agustín Froufe**. "*Tutorial de Java*". Facultad de Informática de Sevilla. 1997. http://www.fie.us.es/info/internet/JAVA/.
- **[García et al., 1999]. Javier Garciá de Jalón, José Ignacio Rodríguez, Iñigo Mingo, Aitor Imaz, Alfonso Brazález, Alberto Larzabal, Jesús Calleja y Jon García**. *"Aprenda Java como si estuviera en primero"*. Universidad de Navarra. 1999. 140 páginas. (*Tutorial muy básico, en el que se trata la potencia de Java, pero sin profundizar en ningún tema en particular*).
- **[García, 1997] Francisco José García Peñalvo.** "*Apuntes de teoría de la asignatura Programación Avanzada del tercer curso de Ingeniería Técnica en Informática de Gestión*". Universidad de Burgos. 1997. 216 páginas.
- **[Johnson, 1996] Jeff Johnson**. "*Coding Standards for C, C++, and Java*". Vision 2000 CCS Package and Application Team. 1996. 14 páginas. (*Consejos para el formato de los fuentes Java, C y C++*).
- **[Lalani, 1997] Suleiman 'Sam' Lalani**. "*Java, biblioteca del programador*". Ediciones McGraw Hill. 1997.
- **[Morgan, 1999] Mike Morgan**. "*Descubre Java 1.2*". Prentice Hall. 1999. 675 páginas. (*Sin duda muy interesante, sobre todo por su actualidad al tratar con Java 2, y por su extensión al tratar todas las bibliotecas de Java*).
- **[Naughton, 1996] Patrick Naughton**. *"Manual de Java"*. Mc. Graw Hill 1996. 395 páginas. (*Introduce todos los aspectos de la programación básica en Java*).
- **[Piattini et al., 1996] Mario G. Piattini, José A. Calvo-Manzano, Joaquín Cervera y Luis Fernández**. "*Análisis y diseño detallado de Aplicaciones informáticas de gestión*". Ra-Ma. 1996.
- **[Rambaugh et al., 1998] J. Rambaugh J., M. Blaha, W. Premerlani, F. Eddy y W. Lorensen**. "*Modelado y Diseño Orientados a Objetos. Metodología OMT*". Prentice Hall, 2º reimpresión. 1998.
- **[Rational, 1997] Rational Software Corporation**. "*The Unified Modeling Language Documentation Set 1.1*". www.rational.com. Setiembre de 1997.

- **[Rifflet, 1998] Jean-Marie Rifflet** "*Comunicaciones en UNIX*". McGraw Hill. 1998.
- **[Rojo, 1998] Ignacio Rojo Fraile**. "*Comparativa de herramientas Java*". Artículo de la revista Solo Programadores nº49. Tower. Octubre 1998. (*Compara 5 IDEs Java*).
- **[Sanz, 1998] Javier Sanz Alamillo**. "*Novedades y cambios con Java 2*". Artículo de la revista Solo Programadores nº55. Tower. Marzo 1999. (*Buen resumen de los cambios que han surgido en el JDK 1.2*).
- **[Schildt 2005] Schildt, Herbert** "LA BIBLIA DE JAVA 2 V5.0" ANAYA MULTIMEDIA 1152 páginas. Idioma: Español ISBN: 8441518653. 1ª edición
- **[Sun, 1998] Sun Microsystems Inc.** "*JDK 1.2 Documentation*". www.sun.com. 1997. (*Documentación de la API de Java del JDK*).
- **[van Hoff et al., 1996] Arthur van Hoff, Sami Shaioi y Orca Starbuck**. *"Hooked on Java"*. Addison-Weslet. 1996. *(Todo lo que hace falta saber para crear applets, con muchos ejemplos. En inglés)*.
- **[Wu 2005] Thomas C. Wu** "INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA" Editorial McGraw-Hill. 2005.Idioma: Español ISBN: 8448131940. 864 páginas; 25x20 cm
- **[Zolli, 1997] Andrew Zolli**. "*La biblia de Java*". Anaya multimedia. 1997. 814 páginas. (*Completo en lo a que biliotecas del JDK 1.1 se refiere*).