

R



CORPORACIÓN
UNIVERSITARIA
REMINGTON
RES. 2661 MEN JUNIO 21 DE 1996

**FACULTAD DE CIENCIAS BÁSICAS E
INGENIERÍA**
Ingeniería de Sistemas
**Asignatura: Lenguaje de programación
avanzado I**

Dirección de Educación a Distancia y Virtual

Este material es propiedad de la Corporación Universitaria Remington (CUR),
para los estudiantes de la CUR en todo el país.

2013

CRÉDITOS



El módulo de estudio de la asignatura Lenguaje de Programación Avanzado I del Programa Ingeniería de Sistemas es propiedad de la Corporación Universitaria Remington. Las imágenes fueron tomadas de diferentes fuentes que se relacionan en los derechos de autor y las citas en la bibliografía. El contenido del módulo está protegido por las leyes de derechos de autor que rigen al país.

Este material tiene fines educativos y no puede usarse con propósitos económicos o comerciales. El autor es Tutor del CAT de la ciudad Villavicencio y el trabajo se desarrollo con el apoyo del Operador Logístico CIBERCTEC



AUTOR

Leyder Hernán López Díaz

Ingeniero de Sistemas. Especialista en Gerencia Informática. Seminario Herramientas Didácticas para el diseño de Estrategias de Aprendizaje para Docentes

Ingsistemas.ciberctec@gmail.com

Nota: el autor certificó (de manera verbal o escrita) No haber incurrido en fraude científico, plagio o vicios de autoría; en caso contrario eximió de toda responsabilidad a la Corporación Universitaria Remington, y se declaró como el único responsable.

Actualizaciones: fueron creadas a través de animaciones de repaso, ejercicios de autoevaluación interactivos, mapas conceptuales y pruebas iniciales por:

César Augusto Jaramillo Henao

Ingeniero de Sistemas

Docente universitario desde 1996

Cesar.jaramillo@Remington.edu.co

RESPONSABLES

Director de la Facultad de Ciencias Básicas e Ingeniería

Dr. Jorge Mauricio Sepúlveda Castaño

jsepulveda@remington.edu.co

Tomás Vásquez Uribe

Director Educación a Distancia y Virtual

tvasquez@remington.edu.co

Angélica Ricaurte Avendaño

Coordinadora de Remington Virtual (CUR-Virtual)

rricaurte@remington.edu.co

GRUPO DE APOYO

Personal de la Unidad Remington Virtual (CUR-Virtual)

EDICIÓN Y MONTAJE

Primera versión. Febrero de 2011. Segunda versión Marzo 2012

Derechos Reservados



Esta obra es publicada bajo la licencia Creative Commons. Reconocimiento-No Comercial-Compartir Igual 2.5 Colombia.

TABLA DE CONTENIDO

1. MAPA DE LA ASIGNATURA.....	5
2. PARADIGMA ORIENTADO A OBJETOS POO Y CARACTERIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN.....	6
2.1. Relación de Conceptos	9
2.2. Fases para la resolución de problemas a través del computador.....	9
2.3. Paradigma orientado a objetos.....	16
2.4. Desarrollo de métodos para los objetos	18
2.5. Uso de funciones predefinidas.....	31
3. METODOS ESTRUCTURAS ENCAPSULAMIENTO Y ADMINISTRACIÓN DE ERRORES HERENCIA POLIMORFISMO E IMPLEMENTACIÓN DE METODOS HEREDADOS	43
3.1. Relación de Conceptos	45
3.2. Controlar el acceso a los miembros de la clase Private Protected Public Acceso de paquetes	45
3.3. Constructores	55
3.4. Subclases Superclases y Herencia	58
4. ALGORITMOS DE BUSQUEDA Y ORDENAMIENTO MODELO DE VENTANAS E INTERFAZ ..	67
4.1. Relación de Conceptos	68
4.2. Algoritmos de búsqueda de datos de vectores.....	69
4.3. Creación de interfaz Grafica y comparación con las aplicaciones de consola	77
4.4. Definición de controles gráficos.....	87
5. COMO CREAR DIALOGOS	93
5.1. Relación de Conceptos	95
5.2. Que es un Applet.....	96
5.3. Applet Viewer.....	111
5.4. Ciclo de vida de un Applet.....	113
6. PISTAS DE APRENDIZAJE	120
7. GLOSARIO	121
8. BIBLIOGRAFÍA.....	129

1. MAPA DE LA ASIGNATURA

LENGUAJE DE PROGRAMACIÓN AVANZADO I

PROPÓSITO GENERAL DEL MÓDULO

El objetivo del lenguaje de programación es acercar al estudiante al conocimiento de las herramientas fundamentales para trabajar en el entorno integrado de desarrollo.

OBJETIVO GENERAL

Crear Applets Java basados en el modelo de programación orientada a objetos, diseñando y sobrescribiendo los métodos necesarios para la construcción apropiada de este tipo de aplicaciones.

OBJETIVOS ESPECÍFICOS

- ✘ Retomar la esencia de la programación básica y estructurada y la tomara como base para la migración hacia el paradigma de objetos y las herramientas de su implementación
- ✘ Comprender cómo se implementa adecuadamente un método, parametrizarlo, definiendo su retorno y estructurando el algoritmo que da esencia a su funcionalidad.
- ✘ Entender qué es una GUI (Interfaz Gráfica de Usuario), y definiendo su importancia en la implementación de aplicaciones que interactúan gráficamente con el usuario final.
- ✘ Comprender que gracias a la interacción de ventanas es posible la construcción una solución que facilita el acceso y la comprensión del usuario de los datos y su importancia para la gestión de los procesos.

UNIDAD 1

PARADIGMA
ORIENTADO A
OBJETOS POO Y
CARACTERIZACION
DEL LENGUAJE DE
PROGRAMACIÓN

UNIDAD 2

METODOS, ESTRUCTURAS,
ENCAPSULAMIENTO Y
ADMINISTRACION DE
ERRORES – HERENCIA,
POLIMORFISMO E
IMPLEMENTACION DE
METODOS HEREDADOS

UNIDAD 3

ALGORITMOS DE
BUSQUEDA Y
ORDENAMIENTO
- MODELO DE
VENTAS E
INTERFAZ

UNIDAD 4

COMO CREAR
DIALOGOS

2. PARADIGMA ORIENTADO A OBJETOS POO Y CARACTERIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN

<http://www.youtube.com/watch?v=teyIHL1BJWA>

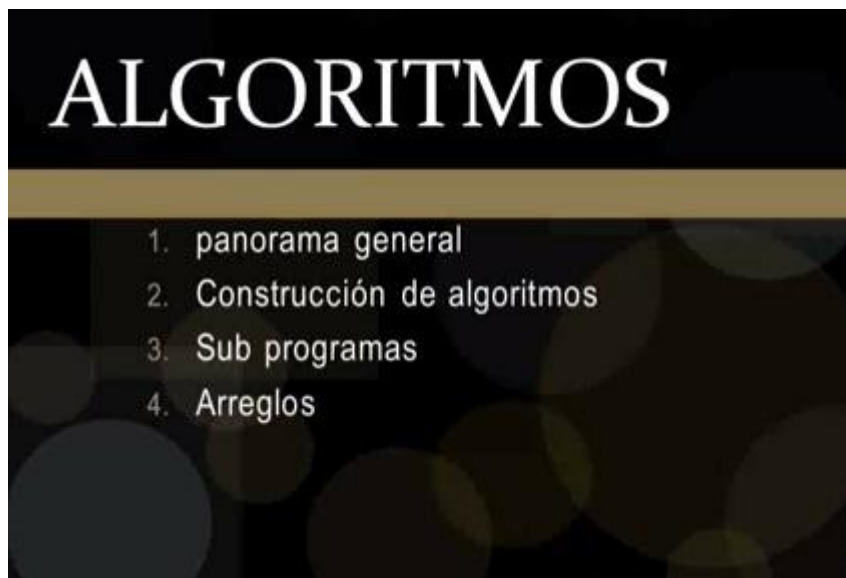


Imagen relacionada del video de youtube

OBJETIVO GENERAL

Retomar la esencia de la programación básica y estructurada y la tomara como base para la migración hacia el paradigma de objetos y las herramientas de su implementación

OBJETIVOS ESPECÍFICOS

- ❏ Desarrollar habilidades lógicas que permitan transmitirlos y procesarlos a través del pc, herramientas que pueden ser educativas, de entretenimiento o de producción, todo basado en procesos lógicos que permitirán plasmarlo y volverlo operativo.
- ❏ Conocer de manera detallada las etapas y los conceptos claves de la orientación a objetos, de dónde proviene, cuál es el propósito dentro de la programación y por qué es importante en esta metodología.

- ❑ Desarrollar habilidades que permitan una mayor interacción entre nuestra cotidianidad y el pc.
- ❑ Conocer de una manera firme los propósitos del lenguaje de programación, sus características más comunes y más importantes.

2.1. Prueba Inicial

Antes de iniciar con el estudio del mundo de la Programación Orientada a Objetos, debe recordar que la esencia de la programación está en la algorítmica, es decir, en la ciencia de diseñar estructuras de análisis lógicos, que fundamentadas en las condicionales, los ciclos, las asignaciones y la definición de variables, permitan resolver cualquier problema de procesamiento de información. Por lo tanto es necesario un repaso de conceptos vistos, así resuelva el siguiente cuestionario:

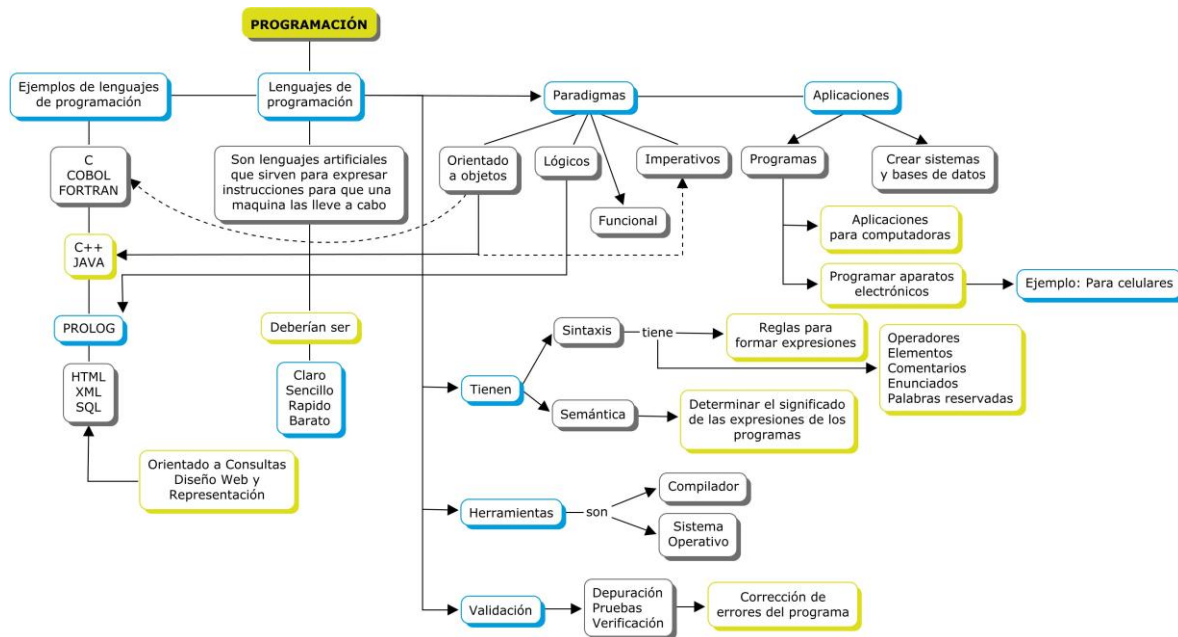
- ❑Cuál es la principal diferencia entre variables y constantes?
- ❑Porque es importante primero diseñar al algoritmo y luego proceder a la programación específica del mismo bajo el lenguaje?
- ❑Desarrolle un algoritmo básico para leer el precio de un conjunto de n productos y calcular el monto de la factura total de la venta de los mismos, calculando un IVA de 17% para todos aquellos productos cuyo costo sea mayor a \$5000. El resto que no cumpla con esta condición tienen un IVA aplicable del 16%.
- ❑Desarrolle un algoritmo para leer dos números enteros y determinar si la suma, la resta, el producto y el cociente entre ellos es un nuevo número que al dividirlo entre ambos operan dos genere como cociente u número par o impar.

NÚMERO 1	100
NÚMERO 2	30

SUMA	130
RESTA	70
PRODUCTO	3000
DIVISIÓN	3,33333333

	COCIENTE	TIPO
Comprobación suma con número 1	1	IMPAR
Comprobación suma con número 2	4	PAR
Comprobación resta con número 1	1	IMPAR
Comprobación resta con número 2	2	PAR
Comprobación producto con número 1	30	PAR
Comprobación producto con número 2	100	PAR
Comprobación cociente con número 1	0	PAR
Comprobación cociente con número 2	0	PAR

2.2. Relación de Conceptos



Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
mapa-conceptual-programación	http://alan-lenpro.blogspot.com/2010/08/mapa-conceptual-programacion.html	Alan Hernández broke	21/10/2011

2.3. Fases para la resolución de problemas a través del computador

La computadora personal es antes que nada, una máquina y como máquina que es, responde a las falencias de quienes interactúan y operan con ella. Pero igualmente, cuando se realiza una correcta estructuración algorítmica de una solución para computadora, esta finalmente ejecutará con precisión el conjunto global de instrucciones diseñadas por el especialista en la programación de dicha aplicación. En primer lugar, es importante tener en cuenta que un Software se construye bajo los siguientes principios:

- Recolección detallada de los requerimientos del usuario.
- Definición de los casos de uso o funcionalidades que el sistema proveerá a partir del análisis de los requerimientos.
- Especificación de una descripción muy detallada de los casos de uso a nivel de todos los pasos necesarios para llegar a su adecuada y correcta ejecución.

- d. Extracción de los objetos candidatos que potencialmente implementaran las funcionalidades descritas en las especificaciones o caracterización de los casos uso.
- e. Definición de los diagramas UML encargados de reflejar el cuerpo arquitectónico de la solución.

No obstante, como se observa, todos estos pasos, muy fácil a simple vista, requeriré de un proceso riguroso de definición de un problema, pero lo más importante, del análisis del mismo con el fin de lograr resultados concretos. En la programación estructurada, no obstante, el análisis, es mucho más simplista, por lo que al programar bajo este modelo se incurre en el riesgo de no tener en consideración todos los escenarios posibles y cometer posteriores errores en el funcionamiento de una aplicación. Donde si la programación orientada hacia los objetos, es en el poder definir bien una secuencia algorítmica de pasos que con orden, estructura y forma, permitan dar respuesta a un problema puntual de procesamiento de información.

Ejemplo

Una empresa de fabricación de compotas requiere tener un sistema para facturar sus ventas con base en las siguientes especificaciones: Toda compota marca Rostington cuya venta sea menor a 100 unidades tendrá un descuento del 20% sobre el valor de la venta; si entre las 101 y 200 unidades se hará un descuento del 25% y si está entre 201 unidades en adelante del 45%. Si las compotas de un lote marca Premiere cumplen con las siguientes condiciones, entonces los descuentos serán como los que se citan a continuación:

Cantidad de compotas	Descuentos sobre la venta
1- 100 unidades	15%
101 – 200 unidades	35%
201 unidades en adelante	50%

Las compotas Rostington se venden a \$5.000 la unidad, mientras que la Premiere a \$10.000. Como ingenieros de Sistemas se le requiere para diseñar una aplicación que se encargue de garantizar la venta de estos productos, siguiendo las reglas de negocio definidas por la compañía.

valor de compotas Rostington	\$ 5.000
valor de compotas Premiere	\$ 10.000

valor de compotas Rostington

Numero de compotas	50
Porcentaje de Descuento I	20%

Porcentaje de Descuento II	25%
Porcentaje de Descuento III	45%

Venta Bruta	\$ 250.000
Porcentaje Aplicado	20%
Descuento	\$ 50.000
Venta Neta	\$ 200.000

valor de compotas Premiere

Numero de compotas	180
Porcentaje de Descuento I	15%
Porcentaje de Descuento II	35%
Porcentaje de Descuento III	50%

Venta Bruta	\$ 1.800.000
Porcentaje Aplicado	35%
Descuento	\$ 630.000
Venta Neta	\$ 1.170.000

Tabla 2.1 Representación de la Solución de Venta en Excel

Ahora el reto es construir un algoritmo que soluciones este problema y posteriormente acudir a un lenguaje como C++, Java, Pascal u otro, y construir un aplicativo que haga esto una realidad. Ahora el paso es identificar las variables, tipo de datos y tipo de almacenamiento. Se tendrá algo así:

Variable	Tipo de Variable	Tipo Almacenamiento	Solución en Excel
TipoCompota	String	Variable	Valor compotas Rostington valor compotas premiere
numeroCompotas	int	Variable	50
precioCompota	double	Variable	10000
descuento	double	Variable	50000
ventaBruta	double	Variable	250000
ventaNeta	double	Variable	200000
porcentajeDescuento	double	Variable	25%

Tabla 2.2 variables del problema de venta compota

EJERCICIO DE AUTOEVALUACIÓN

A continuación se debe implementar el siguiente código para solucionar el problema anterior

Algoritmo VentaCompotas

Inicio

//1. Definición de las variables Cadena

TipoCompota = "";

Entero numeroCompotas = 0;

Real precioCompota = 0.0;

Real descuento = 0.0;

Real ventaBruta = 0.0; Real ventaNeta = 0.0;

Real porcentajeDescuento = 0.0;

//2. Lectura de los datos

Leer ("Ingrese la marca de la compota (1 para Rostington, 2 para Premiere", tipoCompota);

Leer ("Ingrese la cantidad de compotas para la venta:", número Compotas);

//3. Ejecución de los cálculos si (tipoCompota == "1")

Inicio Si

//Asignamos el precio precioCompota = 5000.00;

//Compotas tipo Rostington si (numeroCompotas < 100) Inicio Si

porcentajeDescuento = 0.20;

Fin Si

si (numeroCompotas >= 100 AND numeroCompotas < 200) Inicio Si

porcentajeDescuento = 0.25;

Fin Si

DLC

Inicio DLC

porcentajeDescuento = 0.45; Fin DLC

Fin Si

DLC

//Compotas Premiere

//Asignamos el precio precioCompota = 10000.00;

si (numeroCompotas < 100) Inicio Si

porcentajeDescuento = 0.15;

Fin Si

si (numeroCompotas >= 100 AND numeroCompotas < 200)

Inicio Si

porcentajeDescuento = 0.35;

Fin Si

//Calculamos la venta bruta

ventaBruta = numeroCompotas * precioCompota;

//Calculamos el descuento

descuento = ventaBruta * porcentajeDescuento;

//Calculamos la venta neta

ventaNeta = ventaBruta - descuento;

//4. Imprimimos los resultados si (tipoCompota == "1") Inicio Si

Imprimir("Se realizó una venta de compotas marca Rostington"); Fin Si

DLC

Inicio DLC

Imprimir("Se realizó una venta de compotas marca Premiere"); Fin DLC

//Número de compotas vendidas

Imprimir("La cantidad de compotas vendidas fue de:
",numeroCompotas);

//Precio de la compota

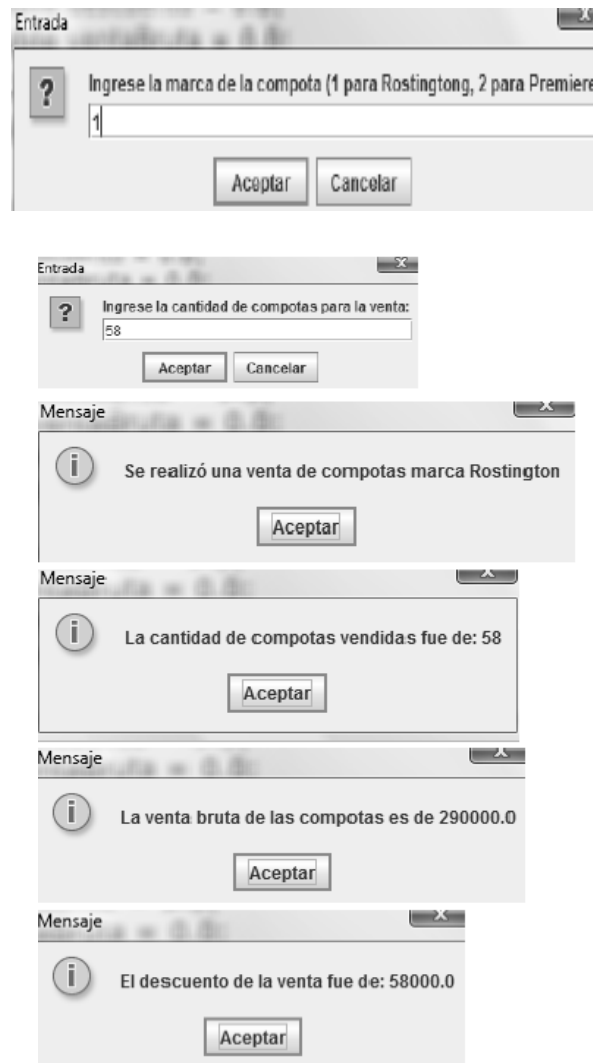
Imprimir("El precio de la compota es de: ",precioCompota);

//Venta bruta

Imprimir("La venta bruta de las compotas es de ",ventaBruta);

```
//El porcentaje de descuento aplicado
Imprimir("El porcentaje de descuento aplicado fue de
",(porcentajeDescuento * 100),"%");
//El descuento
Imprimir("El descuento de la venta fue de: ",descuento);
//La venta neta
Imprimir("La venta neta fue de: ",ventaNeta);
```

Algoritmo, que si es implementado y ejecutado en Java, generaría resultados como estos:



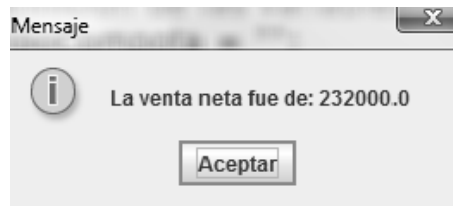


Imagen 2.1

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Algoritmo implementado	Ventanas de trabajo después de implementar el código anterior	Leyder Hernán López	21/10/2011

Componentes necesarios para la construcción de algoritmos

Los algoritmos, antes que nada, son bloques específicos o en términos más simples, “recetas de cocina”, que con precisión y minuciosidad se emplean para resolver secuencial y estructuralmente un problema. Gracias al avance de las matemáticas y a la resolución de los problemas de procesamiento matemático, la informática y la computación permitieron la estructuración de nuevos paradigmas para la generación de soluciones de automatización de procesos. Este fue el punto de partida para la generación de algoritmos cada vez más estructurados y que posibilitaron a lo largo del resto del siglo, organizar la esencia de la Ingeniería de Sistemas. Es importante recordar que la algoritmia brinda herramientas para construir aplicaciones, basadas en el uso de símbolos y su correspondientes elementos de notación en el pseudocódigo o Lenguaje intermedio, el cual finalmente se asemeja bastante a la sintaxis de los Lenguajes de programación tanto comerciales como no comerciales (como C, Visual Basic, Java, C#, entre otros), y que son los que finalmente se usan para programar las aplicaciones de forma concreta y tangible.

Pista de aprendizaje:

Tener en cuenta: la algoritmia y la lógica son fundamentales para interactuar entre la persona y el pc.

Tenga presente: el manejo adecuado de los recursos lógicos dará resultado dentro de las herramientas de desarrollo.

Traer a la memoria: existen metodologías como las pruebas de escritorio que nos hacen más independientes en el manejo de las herramientas.

2.4. Paradigma orientado a objetos

Durante los últimos años se ha hecho un especial énfasis en la forma de estructurar la programación de manera que se convierta en una herramienta que propenda por la modularidad, es decir, por la capacidad de hacer más eficientes las tareas de automatización, recurriendo a unidades funcionales denominadas módulos, que en la forma alegórica de piezas de rompecabezas, puedan ser utilizadas de múltiples formas para generar a su vez, múltiples soluciones informáticas.

La programación estructurada durante las décadas del 60, 70 y 80 fue una solución a muchos problemas de computación, pero a medida que los lenguajes de programación comenzaron a sofisticarse, la programación de este tipo comenzó a convertirse en una pesadilla para labores como el mantenimiento de aplicaciones grandes y complejas. De ahí la necesidad de trabajar sobre un nuevo paradigma que propendiese por hacer más modulares, acoplables y mantenibles las soluciones de Software.

En este sentido, la Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, Modularidad, Polimorfismo y Encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos. Los objetos son entidades que combinan estado, comportamiento e identidad:

- El estado está compuesto de datos, será uno o varios atributos a los que se habrán asignado unos valores concretos (datos).
- El comportamiento está definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él.
- La identidad es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador (concepto análogo al de identificador de una variable o una constante).

Los problemas y sus soluciones por medio de objetos

Los objetos son, ante todo, abstracciones de la realidad física y por lo tanto, se emplean para simular el comportamiento de los objetos físicos en el mundo real. Igualmente competen una gran funcionalidad al representar abstracciones no sólo de cosas del mundo tangencial, sino también para representar modelos de operación de tareas cotidianas e implementar funciones capaces de resolver tareas de procesamiento algorítmico, como por ejemplo, labores de lectura masiva de

datos, recolección de información sobre seres humanos o representar procesos de negocio de las organizaciones a través de operaciones y las relaciones entre dichos procesos.

Gracias a ellos, es muy fácil producir soluciones que independicen la presentación (Interfaz Gráfica de Usuario) de los métodos de procesamiento de información. Esto permite que se puedan reutilizar los objetos de forma transparente en otras aplicaciones, puesto que cada objeto se encarga de exponer sus métodos a los demás a través de la visibilidad.

Así mismo, es importante considerar que gracias a la parametrización de los métodos es posible establecer formas genéricas de capturar la información, para que dependiendo del planteamiento del problema, sea posible determinar formas más genéricas de procesar la información y no anclar la implementación del código de la clase que implementa el objeto a valores fijos que tal vez se deban posteriormente modificar para hacer que la aplicación se comporte de forma diferente. Un ejemplo claro se da en las aplicaciones financieras: Cuando una nueva legislación o una nueva regla financiera o legal es establecida, la aplicación que implementa la funcionalidad financiera debe estar en capacidad de operar con nuevos datos.

```
public double calcularImpuestoVentas (double valor)  
{  
    double impuestoVentas = 0.0;  
    impuestoVentas = valor * 0.15;  
    impuestoVentas = impuestoVentas + (2 * valor);  
    return (impuestoVentas);  
}
```

Como se observa, el valor del impuesto está anclado a multiplicar siempre el valor por el 15%. De ser necesario aplicar un nuevo porcentaje de impuesto, se tendría que modificar el código del método para que se viera reflejado este valor

Así pues, utilizando el concepto de modularidad y capacidad de moldear las operaciones, tal y como lo permite la programación orientada a objetos, tendríamos más bien una porción de código para el método expresada de la siguiente forma:

```
public double calcularImpuestoVentas (double valor, double porcentajeImpuesto)  
{  
    double impuestoVentas = 0.0;  
    impuestoVentas = valor * porcentajeImpuesto; impuestoVentas = impuestoVentas + (2 * valor);  
    return (impuestoVentas);  
}
```

Pista de aprendizaje:

Tener en cuenta: la POO es una herramienta más exigente que otras metodologías de desarrollo.

Tenga presente: que de un buen uso de las metodologías habrá un buen aprovechamiento del recurso del pc.

Traiga a la memoria: que es la única metodología que combina teoría y práctica para aprovecharla al 100%.

2.5. Desarrollo de métodos para los objetos

Análisis de las soluciones

Antes que nada, tenga en cuenta que en la programación el punto de partida de todo proceso se encuentra en el análisis. Entendiéndose éste como la capacidad de entender claramente los tres aspectos básicos que forman todo problema de programación: La especificación o requerimiento, el mundo o contexto y el requerimiento no funcional. Estos tres elementos son claves para realizar un correcto proceso de implementación de sistemas de Software informático.

Las soluciones de Software requieren por tanto, de la especificación, la cual se traduce como un mecanismo que permite detallar con claridad los requerimientos del usuario, es fundamentalmente cualquier cosa que el Sistema debe realizar para dar respuesta al usuario y a sus necesidades de computación y de procesamiento de información. Imaginemos el siguiente caso: Una Institución de Educación Superior (IES), como una Universidad o un Instituto de Formación Tecnológica está implementando su Sistema Software de Información y está desarrollando a su vez, un módulo para manejar los cursos de educación continuada. Se requiere que el Sistema ejecute las siguientes funcionalidades:

- ❏ Buscar registros de propuestas de capacitación
- ❏ Crear ficha académica de nuevos programas
- ❏ Crear registro de propuesta de Educación Continuada Abierta
- ❏ Crear registro de propuesta de Educación Continuada Cerrada
- ❏ Evaluar el estado de satisfacción del cliente

Donde cada funcionalidad hace alusión a un “pantallazo” o conjunto de pantallas que guiarán al usuario a realizar las tareas que mapean situaciones del mundo físico. Por ejemplo, “crear registro de propuesta de educación continuada abierta” haría alusión a un proceso en el cual el director del centro de educación continuada debe crear un documento en un procesador de

palabras en donde se condensen elementos tales como: el nombre del curso, el nombre del capacitador o proveedor que lo impartirá, el número de horas de duración, los contenidos de los módulos que lo forman, la metodología de trabajo y el esquema de evaluación del mismo.

Se busca ahora, por tanto, que el Sistema permita agilizar ese proceso, de manera que no tenga que hacerse en una herramienta externa (procesador de palabras), sino que sea el mismo Sistema quien al final produzca el documento de manera automática sin tener que intervenir mucho el usuario en su construcción, por medio, por ejemplo, de una plantilla estandarizada que genere el documento y lo almacene en la base de datos del sistema.

Un requerimiento como “Evaluar el estado de satisfacción del cliente”, haría alusión a la capacidad del sistema de permitir la presentación de un formulario de evaluación, que por criterios genere estadísticas rápidas y un consolidado promedio de los resultados de evaluación de todo el curso, para evitar al usuario del Sistema tener que llevar los datos a una hoja de cálculo electrónica y tener que realizar los análisis estadísticos de forma manual.

Para realizar este proceso de análisis, que finalmente lleve a la visualización de un producto Software puntual con pantallas, botones y herramientas de procesamiento, se hace necesario detallar de forma muy puntual y granular, cada interacción entre lo que el usuario humano (o no humano) va a hacer, versus lo que el Sistema debe generar para responder a sus respuestas.

Dicho proceso se realiza mediante la implementación de una serie de artefactos documentales, conocidos como Casos de Uso. Estos son elementos que permiten capturar los requisitos de forma detallada, indicando las fronteras de operación del Sistema Software, los usuarios que interactúan con las funcionalidades y finalmente los resultados generados por el procesamiento de las mismas, a partir del proceso de interacción usuario-sistema. Dichos artefactos, se acompañan a su vez de una serie de diagramas que permiten entender mejor la interacción del usuario con el Sistema y de la responsabilidad que cada actor tiene en el proceso. Dichos diagramas son conocidos igualmente como Diagramas de Casos de Uso.

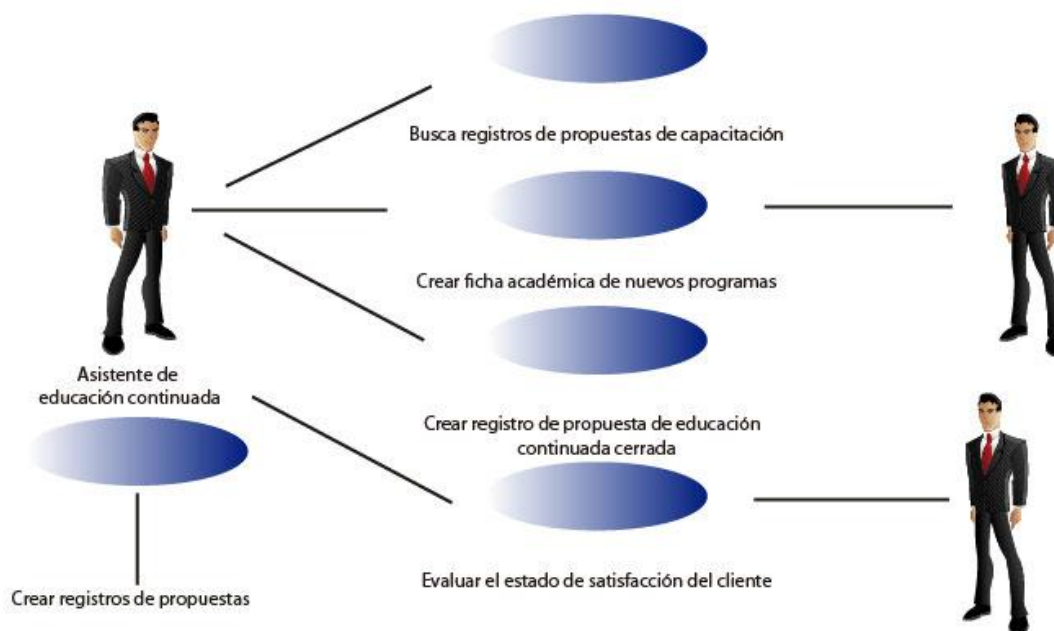


Imagen 2.2

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Diagramas de caso de uso	Diseñado en Word	Leyder Hernán López	21/10/2011

Como se observa en el ejemplo, cada óvalo representa la funcionalidad sistémica que se desea que el Software provea, mientras que las pequeñas figuras humanoides representan actores (en este caso de tipo humano) que pueden interactuar con el sistema. Esto se ve reflejado en lo siguiente:

- El asistente de educación continuada puede buscar en el Sistema registros de propuestas de capacitación, para verificar si ya han sido creadas o para identificar si no lo están. En caso de no existir, puede proceder a crearlas para incluirlas en el sistema. Siempre habrá necesidad de que existan nuevos cursos para vender por parte de la Universidad.
- Crear la ficha de un nuevo curso es responsabilidad del docente, tutor o proveedor que imparte el curso. Una vez este ha ingresado los datos del nuevo curso, el asistente puede entrar a verificar la propuesta para dar su aval o visto bueno a la misma y ofertar el curso a través de los medios de publicidad de la Universidad.
- El asistente debe poder crear nuevas propuestas de educación continuada tanto abiertas como cerradas (abiertas, son propuestas por ejemplo que se abren al público en

general, sin importar que tipo de formación tengan previamente (podrían ser contadores, comerciantes, ingenieros, etc.), por ejemplo, un curso de informática o de finanzas para no financieros; mientras que una propuesta cerrada, es aquella que se realiza sobre un tema en particular destinado a un público objetivo específico, por ejemplo, un curso de fabricación de empaques o un curso de sistemas de gestión de calidad para empresas del sector ambiental).

d) Finalmente, el estudiante es quien debe evaluar los cursos, pues es él quien realiza el curso una vez de abierto al público, ya sea la propuesta en modalidad abierta o cerrada. Una vez evaluado un curso por el compendio de todos sus estudiantes, el asistente debe poder entrar al sistema para revisar las evaluaciones y generar la respectiva retroalimentación, en aras de que el próximo curso sea mucho mejor. Una vez se han identificado las funcionalidades, el siguiente paso es pasar a documentar bien dicha interacción, para lo cual se emplea lo que se conoce como Formatos Expandidos del Caso de Uso, los cuales contienen los siguientes componentes:

1. Nombre detallado del Caso de Uso (se expresa mediante el uso de un verbo en infinitivo o regular. Por ejemplo: Buscar registros, Imprimir Documentos, Generar Listados, Crear usuarios, Eliminar cuentas bancarias, etc.).
2. Descripción Breve: Ilustra en forma de resumen, lo que busca la funcionalidad del sistema, el valor agregado que va a generar para el usuario y los resultados que se espera que el sistema produzca una vez ejecutada la función del Sistema.
3. Actores: Hace alusión a los actores humanos o no humanos (como por ejemplo, otro Software u otro sistema informático) que interactúan con la funcionalidad e instancian o despliegan el funcionamiento de la misma. Sin actor no hay caso de uso que se pueda modelar.
4. Precondiciones: Indican el estado del Sistema antes de la ejecución de la funcionalidad, es decir, cómo se encontrará el entorno de trabajo o funcionalidades antes de que el actor despliegue la ejecución del caso de uso. Por ejemplo, en un sistema de registro de pedidos, una precondición puede ser que el usuario esté autenticado frente al sistema o que existan registros de productos antes de que pueda generarse una orden de pedido.
5. Flujo Básico de Eventos: Representa el conjunto lineal o algorítmico secuencial de instrucciones e interacciones de procesamiento que se realizan por el proceso de interacción usuario sistema. El flujo básico de eventos describe el “caso ideal” o mejor de los casos donde toda la secuencia correcta de pasos permite llegar a la completa ejecución de toda la funcionalidad.

6. **Flujos Alternativos:** Permiten definir rutas alternas en la ejecución de los procesos. Es decir, establecen alternativas que debe seguir el flujo de interacción usuario-sistema a medida que se presentan otras opciones para la ejecución del proceso en general. Por ejemplo, si estuviéramos modelando un caso de uso relacionado con la compra de productos, un flujo básico sería el que sigue el sistema al registrar la compra en efectivo, mientras que flujos alternativos serían definidos en la compra del mismo producto a través de tarjeta de crédito o pago con cheque, opciones que harían que la interfaz gráfica de usuario con la cual el usuario interactúa sea completamente diferente. Debe hacerse la salvedad que el flujo alternativo lleva a una respuesta similar a la que se genera en el flujo básico, solo que cambiando la forma de ejecutar los procesos.
7. **Flujos Excepcionales:** Los flujos excepcionales hacen alusión a los puntos de detención del sistema, donde al presentarse un error producido ya sea por el usuario o por el Sistema, este debe reaccionar para manejar y administrar dicho error. El flujo excepcional muestra cómo debería validarse el error y qué mensaje deberá recibir el usuario para entender y comprender la dificultad aceptada.
8. **Puntos de Extensión:** Definen la relación del requerimiento actual con otros casos de uso, es decir, la vinculación que tiene la funcionalidad con otras necesarias para el buen funcionamiento de todo el Sistema. Por ejemplo, la funcionalidad de buscar registros de estudiantes está relacionada con las funcionalidades de crear registros nuevos, modificarlos y borrarlos también del Sistema.
9. **Postcondiciones:** Define cómo es el estado del sistema una vez ha finalizado la ejecución de la funcionalidad sistémica. Indica cómo se comportan tanto la aplicación como los datos, una vez han finalizado el caso de uso, a través de su flujo básico de eventos o sus flujos alternativos. Por ejemplo, si hablásemos de una aplicación de registro de clientes en un hotel, la pos condición del caso de uso Reservar Habitación, sería la de un registro de habitación separada, que fue asignada a un usuario en particular (cliente del Hotel) y que ya no puede ser dada a ningún otro cliente.
10. **Reglas de Negocio:** Define el conjunto de reglas matemáticas, lógicas, de relaciones humanas y sociales que influyen en el desarrollo de la lógica algorítmica del requerimiento funcional. Estas reglas son muy importantes, precisamente porque establecen las bases del mundo del problema que deben ser tenidas en cuenta para la posterior implementación programática de las interacciones del usuario con el sistema, a través de los objetos y las interfaces gráficas de usuario.

Metodología para la solución de problemas con el enfoque de objetos

En la vida cotidiana estamos rodeados de objetos. Elementos físicos como automóviles, aviones, computadores, teléfonos, celulares, televisores, y muchos otros que proveen servicios específicos para realizar tareas cotidianas. Qué comparten todos ellos en común? Pues el hecho de que todos son elementos tangenciales que podemos tomar y utilizar para realizar dichas tareas y que además tienen ciertas propiedades que se alteran en el momento de hacer uso de ellos. Por ejemplo, cuando usamos un televisor podemos cambiar de canal usando el control remoto o subir y bajar el volumen del audio de lo que estamos viendo. Identificamos, por tanto que el objeto televisor posee dos propiedades internas (canal de televisión y volumen) y que existen operaciones que podemos hacer sobre dicho objeto (cambiar el canal y subir o bajar el volumen).

Bajo este precepto, durante la década de los 80 se analizó que la programación requería de mayor nivel de granularidad, puesto que los programas de tipo estructurado estaban haciendo que el Software se hiciese complejo y sobretodo, poco mantenible, y en el peor de los casos, completamente descartable. Se observó por tanto, con base en la experiencia de lenguajes de programación como Simula 67 y posteriormente perfeccionados por SmallTalk, que era posible definir estructuras de programación modulares que mostraran su propio comportamiento y pudieran almacenar la información de persistencia de los objetos, entendiendo por persistencia, como la capacidad de almacenar de forma permanente los datos para su posterior visualización. Por lo tanto, definamos ahora con mayor precisión los objetos.

Imagine un concepto del mundo real, por ejemplo, la cuenta bancaria de un usuario. Esta es una entidad que como concepto hace alusión a un registro de información financiero de una persona natural o jurídica que permite almacenar el dinero de la persona como tal y realizar ciertas tareas como transacciones como depósitos o retiros.



Imagen 2.3

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Figura cuenta bancaria	http://www.pymecom.com/2010/02/03/lagestion-de-la-tesoreria-2/	Pymecom	27/10/2011

Pero qué la hace realmente un objeto? Fundamentalmente el hecho de que una cuenta bancaria no es una entidad que pueda ser asignada a una sola persona, sino que por el contrario, cada persona (cada usuario del banco) tiene su propia cuenta de manera individual e independiente. Cuando tenemos al Señor Jaime Pérez, este muy seguramente en su cuenta puede que no tenga saldo alguno (\$0), mientras que la señorita Martha Suárez podría tener \$1000.000.000 en la de ella. Como se observa, ambas cuentas son iguales en concepto, pero en su estado son diferentes (el elemento saldo), el cual ya no se puede considerar como un objeto, sino como una propiedad del mismo y que si observamos con profundidad hace alusión a un dato de tipo numérico, en este caso un valor de clase real (dado que el monto del saldo podría contener centavos. Ejemplo: \$50.500.35. Así encontramos que el objeto Cuenta Bancaria tendría propiedades como estas, que harían una cuenta diferente a otra:

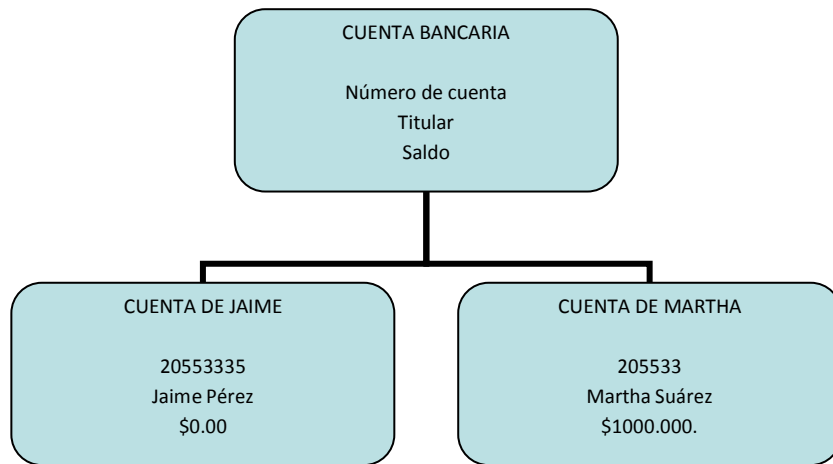


Imagen 2.4

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Concepto de la cuenta Bancaria y ejemplo de la cuenta en el mundo real.	Diseñado en Word	Leyder Hernán López	27/10/2011

Qué garantiza que cada cuenta sea individual? Muchas cosas. En primer lugar que cuando se ejecute una transacción se haga hacia una cuenta específica y que no afecte el saldo de los demás usuarios diferentes al del interesado sobre el que se desea aplicar la transacción. También que sea posible consultar el estado de cada uno o generar un reporte general del estado de todos los usuarios en el sistema independientemente de que sean 20 o 10000. Pero, cómo llegamos a esto? Pues precisamente, a través de la definición de una plantilla que permita especificar de forma genérica, datos que sean comunes a todos los usuarios, y ahí es cuando aparece el concepto de la Clase.

En el mundo real, normalmente se tiene muchos objetos del mismo tipo. Por ejemplo: Un teléfono celular es sólo uno de los miles que hay en el mundo. Si se habla en términos de la programación orientada a objetos, se puede decir que el objeto celular de cualquier tipo 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 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 se le llama Clase, y a los equipos que se sacan a partir de esta se le llaman objetos (Ejemplo: Celulares Motorola, celulares Nokia, Celulares Ericsson, etc).

Esto mismo se aplica a los objetos de Software, se puede tener muchos objetos del mismo tipo y mismas características. Por otro lado, una instancia de una clase es otra forma de llamar a un

objeto. 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.

Así, en nuestro ejemplo, tendríamos que la Cuenta Bancaria se convierte por tanto, en una Clase, una plantilla aplicable a cualquier usuario del banco:

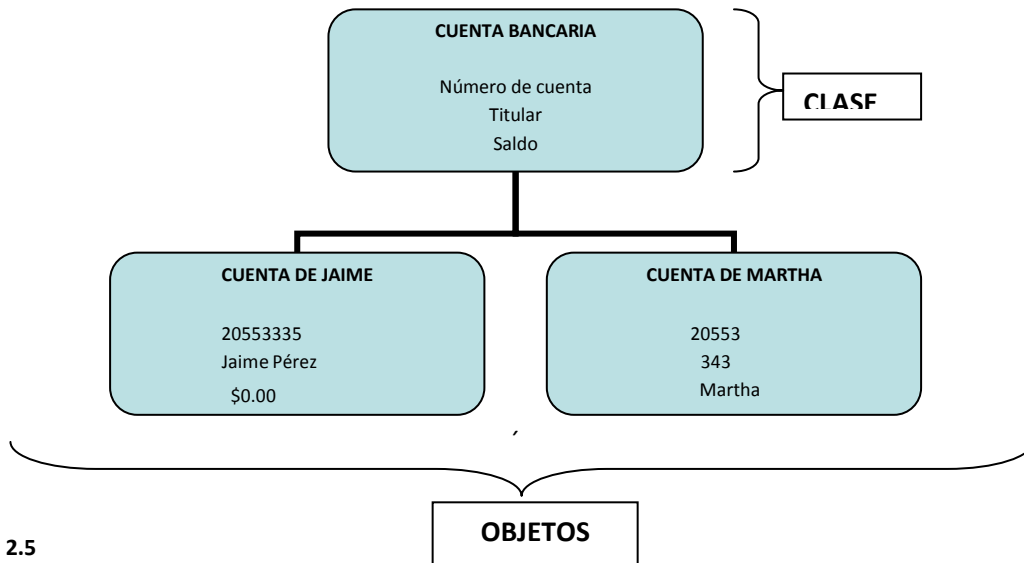


Imagen 2.5

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Representación de una clase y sus correspondientes objetos.	Diseñado en Word	Leyder Hernán López	27/10/2011

Así pues, en el momento en que se crea un nuevo objeto a partir de su plantilla, estamos creando una entidad individual de información con sus propias propiedades y comportamiento y dicho proceso se conoce como Instanciación. Se dicen entonces, que un Objeto es una Instancia o un nuevo tipo de dato generado a partir de una Clase que define su estructura y comportamiento.

Con el objeto de hacer un proceso más detallado de diagramación de los objetos, se ha desarrollado el lenguaje de modelado UML, que permite establecer a través de su notación, los elementos estructurales de diseño de una solución de Software basada en el paradigma orientado a objetos. Dichos diagramas de Clases, están formados por los siguientes elementos:

Propiedades: también llamados atributos o características, son valores que corresponden a un objeto, como color, material, cantidad, ubicación. Generalmente se conoce como la información detallada del objeto. Suponiendo que el objeto es una puerta sus propiedades serían: la marca, tamaño, color y peso.

Operaciones: son aquellas actividades o verbos que se pueden realizar con/para este objeto, como por ejemplo abrir, cerrar, buscar, cancelar, acreditar, cargar. De la misma manera que el nombre de un atributo, el nombre de una operación se escribe con minúsculas si consta de una sola palabra. Si el nombre contiene más de una palabra, cada palabra será unida a la anterior y comenzará con una letra mayúscula, a excepción de la primera palabra que comenzará en minúscula. Por ejemplo: abrirPuerta, cerrarPuerta, buscarPuerta, etc.

Los métodos tienen una estructura de la siguiente forma:

```
public double retirarDinero (String codigoCuenta, boolean
```

```
aceptaRetirar, double montoRetirar)
```

EJERCICIO DE AUTOEVALUACIÓN

A continuación se debe implementar el siguiente código para comprender como funcionan métodos en Java:

Retorno de un dato entero

```
public int calcularSuma (int n1, int n2)
{
//Tipo de dato del retorno int resultado = 0;
//Algoritmo de cálculo
resultado = n1 + n2;
//Retorno del método return (resultado);
}
```

Retorno de un dato real

```
public double calcularIVAProducto (double precio, double porcentaje)
{
double ivaProducto = 0.0; ivaProducto = precio * porcentaje; return (ivaProducto);
}
```

Retorno de un dato booleano

```
public boolean determinarSiSePagalImpuesto (double precio, double retencion)
{
boolean sePagalImpuesto = false;
double pago = 0.0;
pago = precio * 0.16;
```

```
if (pago > 5000)
{
    sePagalmpuesto = true;
}
return (sePagalmpuesto);
}
```

Retorno de un dato caracter (char)

```
public char obtenerInicialNombre (String nombre)
{
    char caracterInicial = '\0';
    caracterInicial = nombre.charAt(0); return (caracterInicial);
}
```

Retorno de un dato de cadena de caracteres (String)

```
public String obtenerNombreMes (int numeroMes)
{
    String nombreMes = "";
    switch (numeroMes)
    {
        case 1:
            nombreMes = "Enero";
            break;
        case 2:
            nombreMes = "Febrero";
            break;
        case 3:
            nombreMes = "Marzo";
            break;
        case 4:
            nombreMes = "Abril";
            break;
        case 5:
            nombreMes = "Mayo";
            break;
        case 6:
            nombreMes = "Junio";
            break;
        case 7:
            nombreMes = "Julio";
    }
}
```

```
break;
case 8:
nombreMes = "Agosto";
break;
case 9:
nombreMes = "Septiembre";
break;
case 10:
nombreMes = "Octubre";
break;
case 11:
nombreMes = "Noviembre";
break;
case 12:
nombreMes = "Diciembre";
break;
} // Fin del switch
return (nombreMes);
}

Método de retorno:
public Empleado obtenerDatosEmpleado (String codigo, String nombre, String apellido, int edad,
double salario)
{
Empleado datosEmpleado = null;
//Instanciamos el objeto a retornar datosEmpleado = new Empleado();
/*Establecemos los valores de los atributos mediante
los métodos de encapsulamiento de la clase y tomados de los valores de los parámetros*/
datosEmpleado.setCodigo(codigo);                datosEmpleado.setNombre(nombre);
datosEmpleado.setApellido(apellido);              datosEmpleado.setEdad(edad);
datosEmpleado.setSalario(salario);

return (datosEmpleado);
}
```

Los métodos de encapsulamiento son métodos que permiten definir la accesibilidad hacia el objeto, entendiendo por accesibilidad como la capacidad de los demás objetos de interactuar con el objeto diseñado. Un ejemplo claro lo tenemos por ejemplo con las propiedades de un objeto como teléfono celular, tales como el número de llamadas que puede manejar, el valor del saldo de la cuenta de pago y el listado de teléfonos de su agenda. Estos datos cuando están escondidos no pueden ser alterados por otros objetos. Otro teléfono celular no puede alterar la agenda del

teléfono actual ni alterar el saldo del mismo. No obstante las funciones como llamar, recibir llamadas, mandar mensajes de texto, que son funciones genéricas a todos los teléfonos, si se encuentran expuestas al uso de los usuarios. El cambio en el estado de las variables de instancia o atributos del objeto teléfono celular (saldo, número de llamadas, etc.) se da por el accionar de dichas operaciones que se exponen a los usuarios de los teléfonos como la capa de servicios por ellos implementados.

Por lo tanto, cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos definidos en la clase:

public: Cualquier clase desde cualquier lugar puede acceder a las variables y métodos de instancia públicos.

```
public void cualquieraPuedeAcceder()  
{  
}
```

protected: Sólo las subclases de la clase y nadie más puede acceder a las variables y métodos de instancia protegidos. Ya veremos este tema con más detalle en el fascículo de Herencia.

```
protected void SoloSubClases()  
{  
}
```

private: Las variables y métodos de instancia privados sólo pueden ser accedidos desde dentro de la clase. No son accesibles desde las subclases.

```
//Ejemplo de un atributo de encapsulamiento privado private String  
NumeroDelCarnetDeldentidad;
```

friendly (sin declaración específica): Por defecto, si no se especifica el control de acceso, las variables y métodos de instancia se declaran friendly (amigas), lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete. Es lo mismo que protected.

```
void MetodoDeMiPaquete(){} }
```

Pista de aprendizaje:

Tener en cuenta: el uso de métodos puede ahorrarnos trabajo, es más simple controlar un bloque de 40 líneas de un gran bloque de 10.000.

Tenga presente: de un buen nombramiento de los métodos y de su estructura puede generar nuevas experiencias dentro de la POO.

Traiga a la memoria: que estos métodos pueden ser recursos para otros desarrollos creados.

2.6. Uso de funciones predefinidas

Ejemplos de conversión de algoritmos a métodos de clases en Java Los datos en Java son el punto de partida fundamental para la construcción de los algoritmos. Gracia a ellos, es posible caracterizar la Información del mundo real, en términos de lo que deseamos que el programa procese a través de sus cálculos y funciones. Finalmente, gracias a estos tipos, podemos construir variables en nuestros programas, las cuales se catalogan en las siguientes:

a) Variables de instancia o atributos: en la esencia del paradigma orientado a objetos, constituyen la fuente de información primaria de procesamiento de cualquier clase. Dichas variables poseen una visibilidad privada, lo que indica que sólo la clase que las implementa puede manipular a través de sus métodos, la información contenida en ellas. Tienen alcance, es decir, se pueden usar por todo el código del programa, pero siempre dentro de los métodos de lógica de negocio implementados por la clase. Ejemplos:

```
private int numero1;
private int numero2;
private int resultado;

public int calcularSuma ()
{
    resultado = numero1 + numero2;
    return (resultado);
}

public void imprimirResultado (String mensaje, int valor)
{
    resultado = valor; JOptionPane.showMessageDialog(null,mensaje + resultado);
}
```

b) Variables locales a los métodos: Son variables que el compilador utiliza únicamente durante la ejecución de dichos métodos. Una vez se ha finalizado la ejecución del método y el retorno (si ha sido implementado) se genera, el garbage collector (recolector de basura de Java) libera las referencias de memoria de dichas variables. Ejemplo:

```
public int calcularProducto (int n1, int n2)
{
    int resultado = 0; resultado = n1 * n2; return (resultado);
}
```

```
public int calcularCociente (int n1, int n2)
{
    int resultado = 0;
    if (n2 != 0)
    {
        resultado = n1/n2;
    }
    return (resultado);
}
```

c) Variables locales a las instrucciones o sentencias: Son variables que únicamente se emplean dentro de instrucciones como ciclos para o en bloques de instrucciones definidos por llaves, como las que puede tener la parte verdadera o falsa de una decisión. Si durante la ejecución del programa, por ejemplo, la condición lógica hace que no se ejecute la parte (falsa o verdadera) en donde se encuentren esas variables, eso significa que la máquina virtual de Java jamás libera memoria para la instanciación de dichas variables. Ejemplos:

```
public void calcularPromedioNotas (int numero_notas)
{
    //Alcance de la variable solo dentro de un ciclo para double promedio = 0.0;
    double nota = 0.0;
    for (int i = 0; i < numero_notas; i++)
    {
        nota = Integer.parseInt(JOptionPane.showInputDialog(null,"Ingrese el valor de la nota: "));
        promedio = promedio + nota;
    }
    //Esta instrucción no podría ejecutarse, puesto que la variable i
    //tiene alcance solo en el for
    //JOptionPane.showMessageDialog(null,"Valor del contador i: " + i);
    //En cambio, promedio como tiene alcance en el método
    //si puede ser accesada fuera del for, pero no antes de
    //su declaración
    promedio = promedio/numero_notas; JOptionPane.showMessageDialog(null,"Promedio de notas:
    " + promedio);
}
```

EJERCICIOS DE AUTOEVALUACIÓN

Desarrollar un algoritmo para ingresar una cadena de caracteres y obtener su equivalente inverso (la cadena al revés). La implementación, por tanto sería:


```
public String invertirCadenaCaracteres (String cadenaOriginal)
{
//Variables locales
String cadenaInvertida = "";
    int i = 0;
char caracter = '\0';

//Recorremos la cadena de atrás hacia adelante
for (i = cadenaOriginal.length()-1; i >= 0; i--)
{
//Sacamos el caracter actual de la cadena caracter = cadenaOriginal.charAt(i);
//Lo agregamos a la cadena que va a tener la
    Palabra //invertida
    cadenaInvertida += caracter;
}

//Retornamos la cadena invertida
return (cadenaInvertida);
}
```

Definición de la clase para el problema del cajero bancario

```
public class CuentaBancaria
{//Inicio de la clase
//Atributos o características de la cuenta bancaria private String numeroCuenta;
private String titular;
private double saldo;
//Método Constructor public CuentaBancaria()
{
super();
// TODO Auto-generated constructor stub
}
public boolean consignarDinero (String numeroCuenta, double montoConsignar)
{
boolean seConsigno = false;
return (seConsigno);
}
public double retirarDinero (String codigoCuenta, boolean aceptaRetirar, double montoRetirar)
{
}
```

```
double valorRetirado = 0.0;
return (valorRetirado);
}
public double consultarSaldo (String codigoCuenta)
{
double valorSaldo = 0.0;
return (valorSaldo);
}
public boolean cerrarCuenta (String codigoCuenta, boolean aceptaCierre)
{
boolean seCerroCuenta = false;
return (seCerroCuenta);
}
} //Fin de la clase
```

Implementación del problema del cajero bancario

```
import javax.swing.JOptionPane;

public class Cuenta
{ //Inicio de la clase
//Atributos de la clase
private String numeroCuenta;
private double saldo;
private String [] cuentas = { "123456", "567890", "112233", "224455" };
    public Cuenta() { super(); }
public boolean consignarDinero (String codigoCuenta, double montoConsignar)
{
//Variables locales del método boolean seConsigno = false; boolean existeCuenta = false;
double bonoAporte = 0.0;
int i = 0;
numeroCuenta = codigoCuenta;
for (i = 0; i < cuentas.length; i++)
{
if (numeroCuenta.equals(cuentas[i]))
{
existeCuenta = true;
}
}
if (existeCuenta == true)
```

```
{
//Verificamos si el valor del saldo es mayor a
//10.000, para
//incrementar al saldo el bono de aporte bonoAporte = 0.0;
if (montoConsignar > 10000)
{
bonoAporte = montoConsignar * 0.20;
JOptionPane.showMessageDialog(null,
"Se otorgará un bono del 20% del valor a consignar en la cuenta");
}
saldo = saldo + montoConsignar;
if (bonoAporte > 0)
{
saldo = saldo + bonoAporte;
}
seConsigno = true;
} else
{
JOptionPane.showMessageDialog(null,"La
//cuenta no existe en el Sistema");
seConsigno = false;
}
return (seConsigno);
}

public double retirarDinero (String codigoCuenta, boolean aceptaRetirar, double montoRetirar)
{
//Variables locales al método
double valorRetirado = 0.0; double retencionFuente = 0.0; double totalRetirado = 0.0; boolean
existeCuenta = false; int i = 0;
String confirmarRetiro = "";
numeroCuenta = codigoCuenta;
for (i = 0; i < cuentas.length; i++)
{
if (numeroCuenta.equals(cuentas[i]))
{
existeCuenta = true;
}
}
if (existeCuenta == true)
{
```

```
if (saldo > 5000)
{
    confirmarRetiro = JOptionPane.showInputDialog(null, "Digite Si para confirmar el retiro o no para
cancelar la operación: ");
    if (confirmarRetiro.equals("Si"))
    {
        //Verificamos el monto a retirar para aplicar la //retención
        if (montoRetirar > 500000)
        {
            retencionFuente = montoRetirar * 0.05;
            JOptionPane.showMessageDialog(null, "Se hará una retención sobre el retiro del 5%");
        }
        totalRetirado = montoRetirar + retencionFuente;
        saldo = saldo - totalRetirado;
        //Informamos cuanto se retiró valorRetirado = otalRetirado;
    }
    else
    {
        JOptionPane.showMessageDialog(null, "La operación ha sido cancelada por el usuario"); }
    }
    else { JOptionPane.showMessageDialog(null, " No se puede retirar: Fondos insuficientes");
    valorRetirado = 0.0;
    }}
    else
    {
        JOptionPane.showMessageDialog(null, "La cuenta no existe en el Sistema");
        valorRetirado = 0.0;
    }
    return (valorRetirado);
}

public double consultarSaldo (String codigoCuenta)
{
    double valorSaldo = 0.0; boolean existeCuenta = false; int i = 0;
    numeroCuenta = codigoCuenta;
    for (i = 0; i < cuentas.length; i++)
    {
        if (numeroCuenta.equals(cuentas[i]))
        {
            existeCuenta = true;
        }
    }
}
```

```
}  
if (existeCuenta == true)  
{  
    valorSaldo = saldo;  
} else  
{  
    JOptionPane.showMessageDialog(null,"La cuenta no existe en el Sistema");  
}  
return (valorSaldo); }  
public static void main(String[] args)  
{  
    String codigoCuenta = "";  
    double valor = 0.0; double saldo = 0.0; double valorRetirar = 0.0;  
    boolean seConsigno = false;  
    codigoCuenta = JOptionPane.showInputDialog(null,"Ingrese el código de su cuenta: ");  
    valor=Double.parseDouble(JOptionPane.showInputDialog(null,"Ingrese el valor a consignar: "));  
    seConsigno = nuevaCuentaBancaria.consignarDinero (codigoCuenta,valor);  
    if (seConsigno == true)  
    { JOptionPane.showMessageDialog(null,"El dinero se consignó satisfactoriamente");  
    }  
    Double.parseDouble(JOptionPane.showInputDialog(null,"Ingr es el valor a retirar de la cuenta: "));  
    saldo      =      nuevaCuentaBancaria.retirarDinero      (codigoCuenta,true,valorRetirar);  
    JOptionPane.showMessageDialog(null,"El valor retirado fue de " + saldo);  
    saldo = nuevaCuentaBancaria.consultarSaldo (codigoCuenta);  
    JOptionPane.showMessageDialog(null,"El saldo actual en la cuenta es de: " + saldo);  
}  
} //Fin de la clase
```

EJERCICIOS DE AUTOEVALUACIÓN

Generar una aplicación que desarrolle un cajero electrónico que permita ingresar dinero y retirar del mismo un valor determinado por el usuario. El sistema debe tener en cuenta las siguientes características:

1. Cuando se realiza una consignación mayor a \$10.000.00, el Banco genera un bono de contribución para el cuenta habiente del 20% del valor depositado.
2. Cuando el usuario retira cualquier cantidad de dinero mayor a \$500.000.00, se genera una retención del 5% sobre el monto retirado.

3. El Sistema debe generar un mensaje de error que no permite retirar si el saldo actual del cuentahabiente es menor o igual a \$5000.00. Teniendo esto en cuenta, el Sistema implementaría los siguientes casos de uso:

- a) Consignar monto en cuenta
- b) Retirar monto de cuenta
- c) Consultar Saldo de la Cuenta

Lo que nos lleva a las siguientes especificaciones de los casos de uso y a generar, por supuesto, el siguiente diagrama de UML:

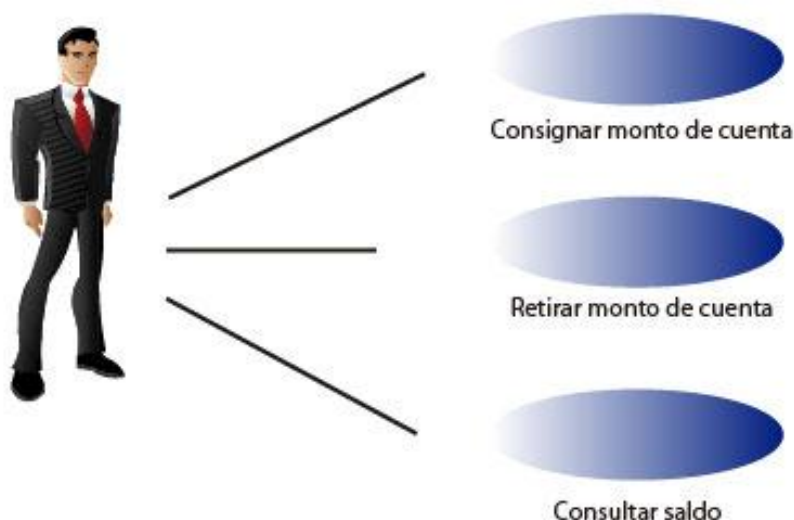


Imagen 2.6

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Diagramas de casos de uso.	Diseñado en Word	Leyder Hernán López	27/10/2011

Las especificaciones del caso de uso: Consignar monto de cuenta, rezarían bajo el siguiente tenor:

Caso de Uso: Consignar monto en cuenta
1. Descripción breve: Este caso de uso hace alusión a la capacidad del sistema de poder ingresar un valor específico de dinero a la cuenta bancaria y con ello incrementar el valor actual del saldo consignado en dicha cuenta. Para ello el usuario debe ingresar el código de la cuenta y el valor total que desea consignar
2. Actor: Cuentahabiente o usuario
3. Precondición: Contar con una cuenta habilitada en el Sistema y tener un saldo y un titular asignado a la misma.
4. Flujo Básico de Eventos El usuario ingresa el código de la cuenta El sistema solicita el valor total que va a ser consignado El usuario ingresa el valor del total a depositar El sistema incrementa el valor del saldo de la cuenta

El sistema informa al usuario que el valor ha sido consignado

El usuario recibe el desprendible de confirmación del monto consignado.

5. Flujos Excepcionales

- ✖ Primer flujo excepcional: Si en el paso 1 el usuario ingresa mal el código, el Sistema genera un mensaje de error informándole que dicha cuenta no existe. El Sistema regresa a pedir nuevamente el código.
- ✖ Segundo flujo excepcional: Si en el paso 2, el usuario presiona el botón Cancelar, el Sistema genera un mensaje de error indicando que la operación de consignación ha sido cancelada. El sistema se cierra por seguridad.

Pos condición: Una vez finalizado el ingreso del nuevo monto, el Sistema actualiza el saldo del cuentahabiente.

7. Lógica de Negocio: Cuando se realiza una consignación mayor a \$10.000.00, el Banco genera un bono de contribución para el cuentahabiente del 20% del valor depositado.

ACTIVIDAD GENERAL

1. Responda las siguientes preguntas:

- a. ¿Qué ventajas tiene la programación como una herramienta para ayudar a optimizar los procesos en las organizaciones y para facilitar el desarrollo de tareas complejas y repetitivas?
- b. ¿Es posible desarrollar una aplicación de computadora sin seguir un proceso riguroso de análisis de requisitos, sino simplemente optando por programar directamente lo que se supone el programa debe implementar de manera inmediata?
- c. ¿Cuáles son las grandes fallas que presentan actualmente las aplicaciones informáticas y porque cada vez se hace más necesario contar con metodologías rigurosas para el diseño de Sistemas de Software?
- d. ¿Puede la programación ayudar a fomentar el pensamiento racional y analítico en las personas del común? Si su respuesta es afirmativa o negativa, justifiquen la razón.

2. Desarrolle un algoritmo para calcular el salario mensual neto de un empleado que tiene las siguientes condiciones:

- ✖ Si el empleado trabaja entre las 8:00 am y las 10:00 pm se le paga un recargo de horas extra equivalente al 10% del total de su sueldo bruto.

- ✘ Si el empleado trabaja entre las 10:00 p.m y las 06:00 a.m. del siguiente día se le paga un recargo de horas extras más horas nocturnas equivalente al 20% del salario bruto (las horas extra representan el 10% y las nocturnas el otro 10%).
- ✘ Si el empleado trabaja entre las 06:00 a.m y las 8:00 a.m no se le da ningún recargo adicional.
- ✘ Si el empleado trabaja entre las 8:00 am y las 10:00 pm se le paga un recargo de horas extra equivalente al 10% del total de su sueldo bruto.
- ✘ Si el empleado trabaja entre las 10:00 p.m y las 06:00 a.m. del siguiente día se le paga un recargo de horas extras más horas nocturnas equivalente al 20% del salario bruto (las horas extra representan el 10% y las nocturnas el otro 10%).
- ✘ Si el empleado trabaja entre las 06:00 a.m y las 8:00 a.m no se le da ningún recargo adicional.
- ✘ A todos los sueldos brutos, independientes del recargo, deben cobrárseles las siguientes deducciones:

15% por salud

14.5% por pensiones y cesantías

1.25% por Riesgos profesionales si está en nivel I

2.25% por Riesgos profesionales si está en nivel II

3.25% por Riesgos profesionales si está en nivel III

4.25% por Riesgos profesionales si está en nivel IV

5.25% por Riesgos profesionales si está en nivel V

3 Desarrolle un algoritmo que sume todos los números de un rango dado de valores y calcule sobre ellos los siguientes valores:

$$X = (2 * \text{Suma})/5;$$

$$Y = (\text{Suma} \wedge 3) + 1/3 * (\text{Suma} * \text{sqrt}(\text{Suma}))$$

$$Z = 1/2 * (\text{Suma}) + \text{sqrt}(\text{LN}(\text{Suma} + 1/\text{Suma}))$$

Así, si los valores están entre 1 y 5, la suma debe dar 15 y el algoritmo debe dar los resultados de los cálculos arriba mencionados.

3. Desarrolle un algoritmo para calcular el número total de días que contiene un grupo de años dado. Así si se introducen 40 años, el algoritmo debe mostrar 14@10 días. Recuerde que algoritmo debe tener en cuenta los años que son bisiestos.

Pista de aprendizaje:

Tener en cuenta: las herramientas de desarrollo son muy amplias y muy completas para muchas de las tareas de la vida cotidiana.

Tenga presente: de no presentarse una función para nuestro propósito las podremos construir con los recursos que nos brinda el lenguaje.

Traer a la memoria: que la única limitante para el desarrollo de software somos nosotros mismos.

3. METODOS ESTRUCTURAS ENCAPSULAMIENTO Y ADMINISTRACIÓN DE ERRORES HERENCIA POLIMORFISMO E IMPLEMENTACIÓN DE METODOS HEREDADOS

<http://www.youtube.com/watch?v=nhn8995y5MI>



Imagen relacionada del video de youtube

OBJETIVO GENERAL

Comprender cómo se implementa adecuadamente un método, parametrizarlo, definiendo su retorno y estructurando el algoritmo que da esencia a su funcionalidad.

OBJETIVOS ESPECÍFICOS

- Estudiar de manera extensa el alcance de cada componente de la programación orientada a objetos.
- Analizar los componentes de las clases que permitan el control inicial, la instauración y el uso de los recursos propios y del sistema, el uso de los procesos creados y los recursos que podamos compartir entre aplicativos y componentes.

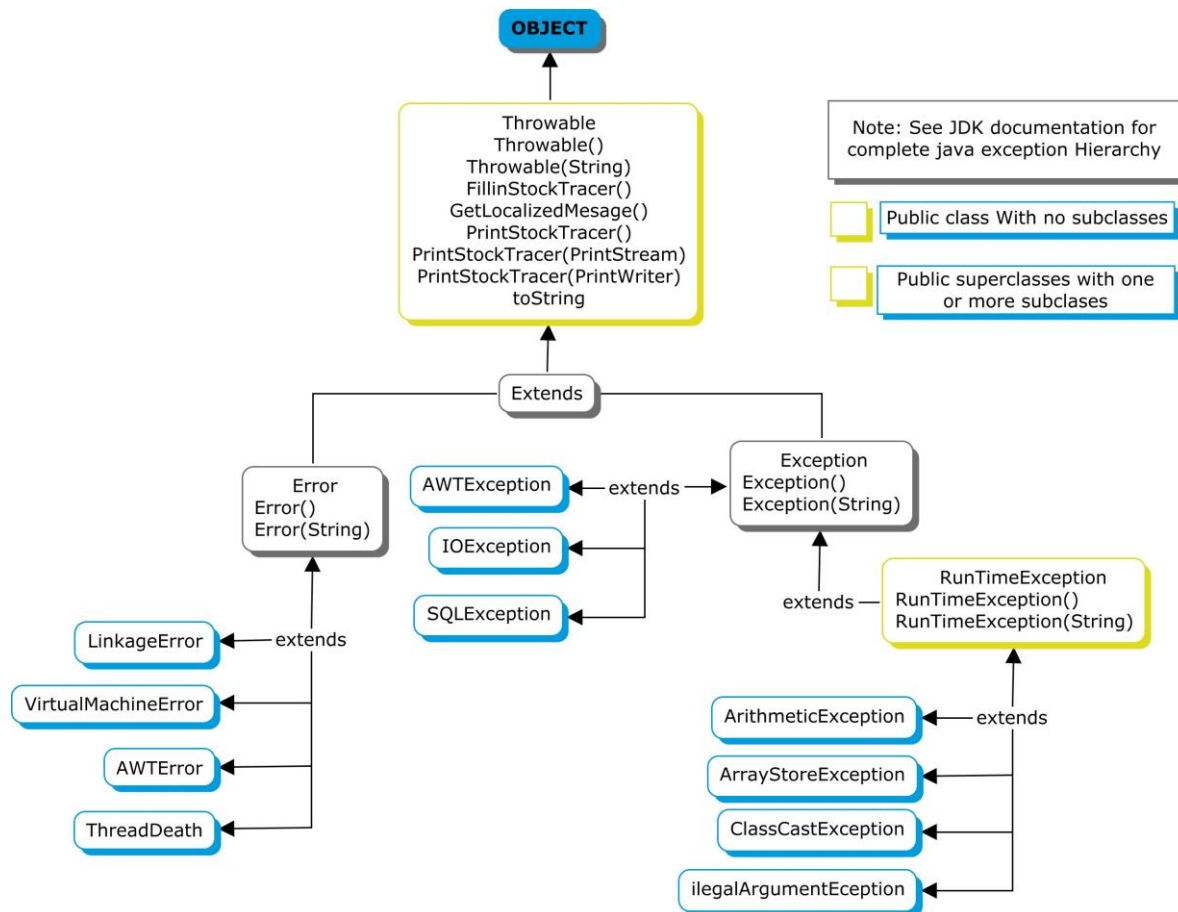
- Conocer nuestro entorno para aprovecharnos de las herramientas del lenguaje, entender apropiadamente las jerarquías de nuestro componente y clasificar si es subclase o superclase.

3.1. Prueba Inicial

Responda las preguntas planteadas a continuación:

- a) Qué importancia tiene programar una aplicación basada en el uso de clases, a diferencia de un programa construido con el modelo clásico?
- b) Qué partes forman una clase Java y cuál es el método que se encarga de dar vida a la aplicación?
- c)Cuál es la diferencia principal entre datos primitivos y datos avanzados (de tipo objeto)?
- d) Qué ventajas tiene programar de forma modular, a diferencia de programar en forma estructurada?
- e) Qué buena estrategia puede tenerse en cuenta para diseñar una Clase y para generar los atributos y métodos que forman la solución para el problema de programación?

3.2. Relación de Conceptos



Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Errores y excepciones en Java	http://victoralex-poo.blogspot.com/2011/04/eventos-excepciones-y-errores.html	Victor Alejandro	29/11/2011

3.3. Controlar el acceso a los miembros de la clase Private Protected Public Acceso de paquetes

Recuerde que el punto de partida para la construcción de aplicaciones está en la definición de su algoritmo como pieza lógica de ejecución de los procesos. El algoritmo de programación, independientemente de su función, contiene las siguientes estructuras básicas:

- a) Sentencias de Salto: tienen como objetivo permitir la estructuración lógica de decisiones en los puntos de control de un algoritmo, en donde es necesario realizar una pregunta para tomar un camino de bifurcación lógico.

If (Boolean)

```
{  
//Parte verdadera  
Sentencias;  
}  
else  
{  
//Parte verdadera  
Sentencias;  
}
```

Ejemplo

```
public void mostrarFuncionamientoSiLogico (int valor)  
{  
if (valor > 100)  
{  
JOptionPane.showMessageDialog (null,"El valor " + valor + " es mayor a 100");  
}  
else {  
  
JOptionPane.showMessageDialog (null,"El valor " + valor + " es menor o igual a 100");  
}  
}
```

En este orden de ideas, las decisiones tienen tres modalidades:

- ❑ Decisiones simples: como las vistas en el ejemplo anterior. Tienen una sola pregunta lógica y dos opciones posibles (su parte verdadera y su parte falsa).
- ❑ Decisiones anidadas: se denominan así porque una pregunta de la estructura lógica decisional conduce invariablemente hacia otra pregunta. Se puede considerar como una especie de filtros sucesivos que llevan hacia la ejecución de un proceso, con base en el cumplimiento de todas las condiciones planteadas en las preguntas que lo forman. No obstante, las partes falsas de dichos procesos de condicionalidad lógica también pueden conducir hacia la ejecución de instrucciones, que hacen que el filtro deje de operar en algún punto donde la condición llegase a hacerse falsa.

if (sentencia1)

```
{
```

```
if (sentencia2)
{
if (sentencia3)
{
instrucciones_parte_verdadera3;
}
else
{
instrucciones_parte_falsa3;
}
}

else
{
instrucciones_parte_falsa2;
}
}

else
{
instrucciones_parte_falsa1;
}
```

- ❏ Decisiones encadenadas: Las decisiones de tipo encadenado son decisiones de carácter excluyente, es decir, que una vez se ha evaluado una condición lógica y esta se hace verdadera, el resto de decisiones quedan automáticamente excluidas. A esto se le llama evaluación “en corto circuito”. Suponga que entra a un restaurante y pregunta sobre las opciones del menú. En el momento en que el mesero diga que un plato está presente, se respondería que lo traiga, lo que haría que el resto de las alternativas del menú (que seguramente también estarían presentes para su consumo) fueran descartadas a raíz de la escogencia de esta decisión.

Implementación de métodos en Java

En los esquemas de la programación estructurada, se seguía una metodología orientada por la ejecución lineal de los procedimientos. Es decir, se seguía el esquema del modelo entrada-proceso-salida, que aunque se sigue manejando en la programación orientada a objetos, cambia a un medio más estructurado de construir las aplicaciones. Imagine la siguiente situación: se solicita la construcción de una aplicación que permita calcular el número de meses que existen entre dos años dados por el usuario. Siguiendo el modelo estructurado, tendríamos un algoritmo más o menos de esta forma:

Programa NumeroMeses;

Inicio

```

/*****/

```

```

//Declaración de variables Entero numero_meses = 0;

```

```

Entero ano_inicial = 0; Entero ano_final = 0;

```

```

Entero diferencia = 0;

```

```

Leer("Ingrese el año inicial: ", ano_inicial);

```

```

Leer("Ingrese el año final: ", ano_final);

```

```

diferencia = ano_final - ano_inicial; numero_meses = diferencia * 12;

```

```

Imprimir("Entre el año " + ano_final + " y el año " + ano_inicial + " hay " + numero_meses + "
meses");

```

Fin.

Al analizar la estructura del algoritmo, se encuentra que éste contiene lo mencionado con anterioridad: Primero una declaración de variables, posteriormente un proceso de Lectura de datos, acto seguido procedemos a transformar los datos mediante las instrucciones matemáticas y lógicas del algoritmo y finalmente procedemos a la salida. No obstante, si corremos linealmente el programa, obtenemos la siguiente secuencia de eventos:

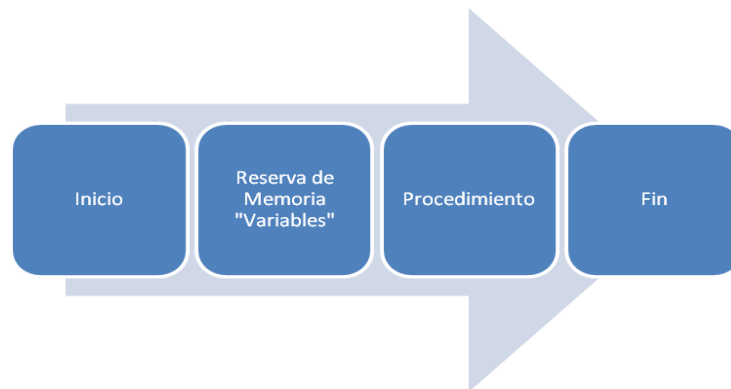


Imagen 3.1

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Esquema de ejecución de una aplicación programada estructuralmente.	Diseñado en Word	Leyder Hernán López	29/11/2011

Si llevamos esta implementación a Java, se obtendría la siguiente definición de instrucciones para hacer realidad el programa Y al ejecutar el programa en el IDE, tendríamos la siguiente respuesta:



Imagen 3.2

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ejecución del programa de obtención del número de meses de una diferencia de años.	Código implementado	Leyder Hernán López	29/11/2011

Hasta el momento no hay ningún problema. El programa está perfectamente implementado y a excepción de que no se ha validado el ingreso de los datos, tenemos una aplicación funcional. Pero qué ocurre si se desea hacer más modular esta aplicación e implementar más funcionalidades o incluso permitir a otras aplicaciones usar la funcionalidad del cálculo del número de meses? Como está implementada actualmente bajo un lenguaje de programación, la aplicación está completamente encapsulada y cerrada a los demás objetos que pudiesen llegar a interactuar con ella. Es en este momento cuando se retoma el concepto de los objetos como unidades funcionales, capaces de proveer servicios.

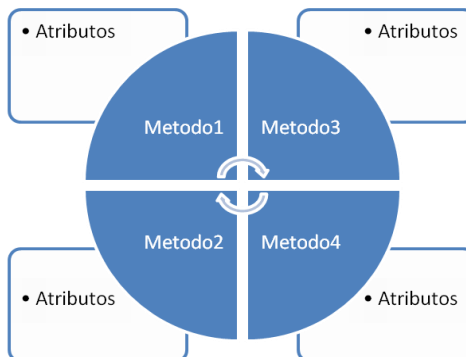


Imagen 3.3

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Interacción entre los objetos.	Diseñado en Word	Leyder Hernán López	29/11/2011

En primer lugar, hay que tener en cuenta que un conjunto de objetos aislados tiene escasa capacidad para resolver un problema. En una aplicación real los objetos colaboran e intercambian información, existiendo distintos tipos de relaciones entre ellos.

Pasar Información a un Método

La esencia clave de la construcción de métodos en Java, se fundamenta en gran medida en las especificaciones de los casos de uso. Estos nos permiten saber cuáles son las entradas (parámetros) y cuáles las salidas (retornos) que los métodos deben de tener. Recuerde que si el requerimiento dice qué debemos hacer, a partir de la lectura de la semántica de dicho requerimiento debemos obtener los parámetros, variables locales y finalmente retornos de los métodos. Por ejemplo, se nos pide crear un método para calcular el impuesto que debe ser aplicado a un producto, que cumpla las siguientes condiciones:

- ❏ Si su precio está entre 1000 pesos y 10000, el impuesto aplicable debe ser del 20% del valor del producto.
- ❏ Si su precio está entre 10000 pesos y 50000, el porcentaje del impuesto debe ser del 40%, y si
- ❏ Su precio es mayor a 50000, el valor del impuesto debe ser del 55%

```
public double calcularImpuestoPrecioProducto (double precioProducto, double
porcentajeRetencion)
{
double valorImpuesto = 0.0;
double impuesto = 0.0;
double retencion = 0.0;
if (precioProducto >= 1000 && precioProducto < 10000)
{
impuesto = precioProducto * 0.20;
}
else
if (precioProducto >= 10000 && precioProducto < 40000)
{
impuesto = precioProducto * 0.40;
}
else
{
```

```
impuesto = precioProducto * 0.55;
}
if (precioProducto >= 25000 && precioProducto <= 45000)
{
    retencion = precioProducto * porcentajeRetencion;
}
valorImpuesto = precioProducto - (impuesto + retencion);
//Retornamos el valor return (valorImpuesto); }
```

El concepto de visibilidad se convierte en una herramienta fundamental dentro del enfoque del paradigma orientado a objetos. La visibilidad, como la capacidad de los objetos de ser vistos entre sí y con ello facilitar la interacción de los mismos, a través de las relaciones de herencia, asociación, agregación o composición, es un recurso muy importante para la definición de la forma como los objetos colaboran entre sí para resolver los problemas de procesamiento de información. Como se ha visualizado en Eclipse, o en cualquier otro IDE, en el momento de desplegar la implementación de las clases y definir su visibilidad, esto permite que una clase pueda ser instanciada desde otra sin ninguna dificultad.

No obstante, cuando las clases se encuentran ubicadas en proyectos aparte o en recursos aparte, se hace necesario saber hasta donde es posible tener control de acceso a los miembros, es decir, poder acceder a los atributos y métodos implementados por las clases, de acuerdo a su accesibilidad.

EJERCICIOS DE AUTOEVALUACIÓN

IMPLEMENTACIÓN DEL PROGRAMA DE MANEJO DE FECHAS EN FORMA ESTRUCTURA

```
import javax.swing.JOptionPane;

public class NumeroMeses
{
    //Inicio de la clase
    public NumeroMeses()
    {
        // TODO Auto-generated constructor stub
    }
    public static void main(String[] args)
    {
        int numero_meses = 0;
        int ano_inicial = 0; int ano_final = 0; int diferencia = 0;
```

```
//Lectura de los datos ano_inicial = Integer.parseInt (JOptionPane.showInputDialog(null
,"Ingrese l año inicial: ")); ano_final = Integer.parseInt (JOptionPane.showInputDialog(null
,"Ingrese el año final: "));
diferencia = ano_final - ano_inicial;
numero_meses = diferencia * 12;
JOptionPane.showMessageDialog(null,"Entre el año " + ano_final + " y el año " + ano_inicial + " hay
" +
numero_meses + " meses"); }
} //Fin de la clase
```

IMPLEMENTACIÓN DE LA RELACIÓN DE DEPENDENCIA DE CLASES

```
import java.util.Date;
import javax.swing.JOptionPane;
public class Reloj
{
private int hora;
private int minuto;
private int segundo;
private Date fechaSistema;
public Reloj()
{
super();
fechaSistema = new Date();
//System.out.println("fechaSistema: " +fechaSistema);
}
public int obtenerHora ()
{
hora = fechaSistema.getHours();
return (hora);
}
public int obtenerMinuto ()
{
minuto = fechaSistema.getMinutes(); return (minuto);
}
public int obtenerSegundos()
{
segundo = fechaSistema.getSeconds();
return (segundo);
}
```

```
}
```

```
public void mostrarHora (int hora, int minuto, int segundo)
```

```
{
```

```
String horaSistema = "Hora del Sistema: ";
```

```
String cadenaHora = "";
```

```
String cadenaMinuto = "";
```

```
String cadenaSegundo = "";
```

```
String cadenaFormato = "";
```

```
if (hora >= 0 && hora <= 9)
```

```
{
```

```
cadenaHora = "0" + String.valueOf(hora);
```

```
}
```

```
else
```

```
{
```

```
if (hora >= 10 && hora <= 12)
```

```
{
```

```
cadenaHora = String.valueOf(hora);
```

```
}
```

```
else
```

```
{
```

```
cadenaHora = String.valueOf(hora-12);
```

```
}
```

```
}
```

```
if (minuto >= 0 && minuto <= 9)
```

```
{ cadenaMinuto = "0" + minuto; }
```

```
else
```

```
{
```

```
cadenaMinuto = String.valueOf(minuto);
```

```
}
```

```
if (segundo >= 0 && segundo <= 9)
```

```
{
```

```
cadenaSegundo = "0" + segundo;
```

```
}
```

```
else
```

```
{
```

```
cadenaSegundo = String.valueOf(segundo);
```

```
} if (hora >= 0 && hora <= 12)
```

```
{
```

```
cadenaFormato = "AM";
```

```
}  
else  
{  
cadenaFormato = "PM";  
}  
horaSistema = cadenaHora + ":" + cadenaMinuto + ":" + cadenaSegundo + " " + cadenaFormato;  
JOptionPane.showMessageDialog(null, horaSistema);  
}  
public static void main(String[] args)  
{  
int salir = 0;  
do  
{  
Reloj relojActual = new Reloj();  
relojActual.mostrarHora(relojActual.obtenerHora(),  
relojActual.obtenerMinuto(), relojActual.obtenerSegundos());  
do-while  
salir = JOptionPane.showConfirmDialog(null, "¿Desea salir del reloj (S/N): ");  
} while (salir == JOptionPane.NO_OPTION);  
}  
  
}
```

Pista de aprendizaje:

Tener en cuenta: los compones del lenguaje O.O son muy amplios y de mucha utilidad en futuros desarrollos.

Tengapresente: que se deben de conocer al detalle para aprovecharlos adecuadamente.

Traer a la memoria: que todos manejamos conceptos y lógica distinta, por esto no encontramos aplicaciones exactamente iguales así tengan el mismo propósito.

3.4. Constructores

Tenga en cuenta que cuando se va a ejecutar la aplicación principal o método main de cualquier clase, siempre se incluye la sentencia

Clase objeto = new Clase();

Esta sentencia hace alusión al método fundamental que el lenguaje ofrece para la instanciación de nuevos objetos o la “construcción” de nuevos objetos. De ahí, el nombre de Constructor para este método especial, cuyo objetivo es el de reservar memoria en una nueva instancia de la máquina virtual, con el objetivo de dar vida a un objeto y por ende, facilitar el acceso a sus atributos y los métodos que ya sabemos implementar.

El uso fundamental de los métodos constructores en Java, busca facilitar la instanciación de un objeto en tiempo de ejecución y a su vez, gracias al concepto de la sobrecarga, es decir, la capacidad de parametrizar de diferentes maneras un mismo método, facilitar la configuración o presetado de ciertos valores, instrucciones y condiciones lógicas que deben ser tenidas en cuenta el momento de crear un nuevo objeto. Por ejemplo, si se está creando una clase para crear un objeto de tipo Empleado y queremos que dicho empleado asuma una serie de valores en el momento de la inicialización, debemos poder sobrescribir el método constructor base que Java da a dicha Clase, mediante la configuración de valores que en determinando momento cumplan dicha tarea de configuración de datos a las variables de instancia de la clase.

```
//Constructor base o vacío public Empleado()
{
}
//Constructor sobrecargado de la clase que se encarga de
//asignar valores a las variables de instancia de la clase Empleado
public Empleado (String nombre, int edad, double salario)
{
//Asignamos a cada variable de instancia de la clase
//el valor entregado por el parámetro this.nombre = nombre;
this.edad = edad;
this.salario = salario;
}
```

Esto permite definir infinitud de constructores, todos especializados en tareas que deben ser ejecutadas en el momento de crear un nuevo objeto del tipo de la clase a la cual pertenece dicho constructor. Dado que el método constructor es otro método más de la clase, a dicho método se le pueden aplicar los indicadores de accesibilidad (`private`, `public` y `protected`), por lo que considerando este criterio, tendríamos las siguientes características:

- a) `private`: Ninguna otra clase puede crear un objeto de su clase. La clase puede contener métodos públicos y esos métodos pueden construir un objeto y devolverlo, pero nada más.
- b) `protected`: Sólo las subclases de la clase pueden crear ejemplares de ella.
- c) `public`: Cualquiera puede crear un ejemplar de la clase.
- d) `package-access`: Nadie externo al paquete puede construir un ejemplar de su clase. Esto es muy útil si se quiere que las clases que tenemos en un paquete puedan crear ejemplares de la clase pero no se quiere que lo haga nadie más.

Excepciones

En algunas ocasiones cuando una aplicación Java se está desplegando, se presentan en ella muchos tipos de errores, debidos en gran medida a datos mal ingresados por parte del usuario o a fallos en los recursos del sistema operativo que soportan la máquina virtual donde finalmente la aplicación está corriendo. Es por ello, que se hace muy necesario que el programador pueda controlar este tipo de errores en tiempo de ejecución, con el fin de poder dar una salida limpia a la ejecución del programa, en el momento de que dichas fallas puedan llegar a presentarse. Para ello, Java, como todos sus procesos, maneja los errores mediante la implementación de objetos propios, encargados de procesar las eventualidades potenciales, que de acuerdo al tipo de aplicación desarrollada y a sus objetivos de ejecución, pueden llegarse a presentar por múltiples factores. El siguiente esquema ilustra la jerarquía de errores manejada por la Plataforma Java de Sun:

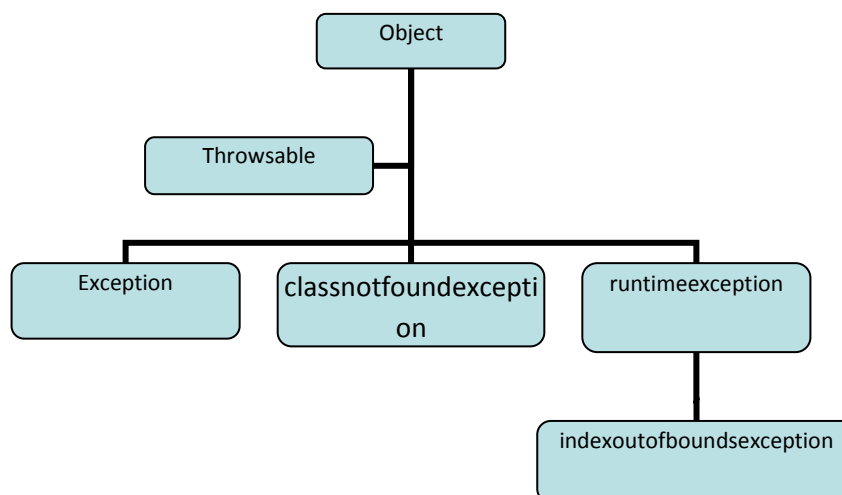


Imagen 3.4

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Jerarquía de Errores en Java.	Diseñado en Word	Leyder Hernán López	29/11/2011

Como se observa en el diagrama, así como todas las clases Java heredan de la clase Object, los errores en Java comienzan a heredar de una jerarquía a partir del objeto Throwable, por lo que gracias a esto, el desarrollador puede acceder a los siguientes métodos para depurar y controlar los errores presentados en el momento de la ejecución:

- ❏ getMessage(): Se usa para obtener un mensaje de error asociado con una excepción.
- ❏ printStackTrace(): Se utiliza para imprimir el registro de la pila de procesos de Java donde se ha iniciado la excepción.
- ❏ toString(): Se utiliza para mostrar el nombre de una excepción junto con el mensaje que devuelve getMessage().

Así pues, el objetivo de las excepciones es ofrecer la capacidad de permitir que la aplicación siga corriendo a pesar de haberse presentado el error e informar al usuario de cuál fue el error presentado en el momento de lanzarse la excepción capturada por la máquina virtual. Cuando una clase no implementa excepciones, los programas terminan por abortarse abruptamente, generando con ello que el ciclo de ejecución se detenga y que no se pueda rastrear con tanta facilidad la causa profunda y concreta del desarrollo de la falla. El manejo de excepciones utiliza para ello, el bloque de sentencias java try- catch-finally. Dicho bloque de sentencias cumple el siguiente propósito:

- ❏ try: Es un bloque que determina qué sentencias deben ejecutarse en el momento del disparo de un método, presuponiéndose que no se va a presentar ningún error ni de usuario ni de procesamiento de datos en el momento del desarrollo de dichas instrucciones.
- ❏ catch: Define la excepción que Java a través del método que implementa dicho bloque catch debe administrar. En este bloque se le indica al programa qué hacer cuando se

presente el error y cómo debe informar al usuario sobre la consola estándar de Java, con el fin de que este pueda mirar la trazabilidad del error ejecutado y detectar con facilidad la falla.

- ✖ **finally:** Es un bloque de sentencias alternativas que deben ejecutarse una vez la falla se ha presentado. En dicho bloque, el programador puede realizar tareas como cerrar objetos, eliminar referencias de memoria, limpiar procesos o detener la ejecución de tareas pesadas en Java que hayan sido abiertas en el momento previo al disparo de la excepción de ejecución.

Pista de aprendizaje:

Tener en cuenta: de un buen uso de la herramienta encontraremos aprovechamiento no solo lógico sino de los propios recursos del sistema como la memoria y el espacio en disco.

Tenga presente: los constructores los podemos especificar de lo contrario el sistema hará esto de forma transparente para el usuario.

Traer a la memoria: estos recursos se manejan en la memoria del computador y nos darán un alcance mayor según nuestro propósito.

3.5. Subclases Superclases y Herencia

La implementación de los métodos permite que un objeto que no ejecuta un comportamiento específico a través de las funcionalidades algorítmicas provistas en tales métodos, invoque a través de una instancia de ese segundo objeto, los métodos pasando los parámetros apropiados para que sea esa clase la encargada de procesar y devolver resultados a través de los retornos. En Java, como en otros lenguajes de programación orientados a objetos, las clases pueden derivar desde otras clases. La clase derivada (la clase que proviene de otra clase) se llama subclase. La clase de la que está derivada se denomina superclase. De hecho, en Java, todas las clases deben derivar de alguna clase, lo que nos lleva a la cuestión ¿Dónde empieza todo esto?. La clase más alta, la clase de la que todas las demás descienden, es la clase Object, definida en java.lang. Object es la raíz de la herencia de todas las clases.

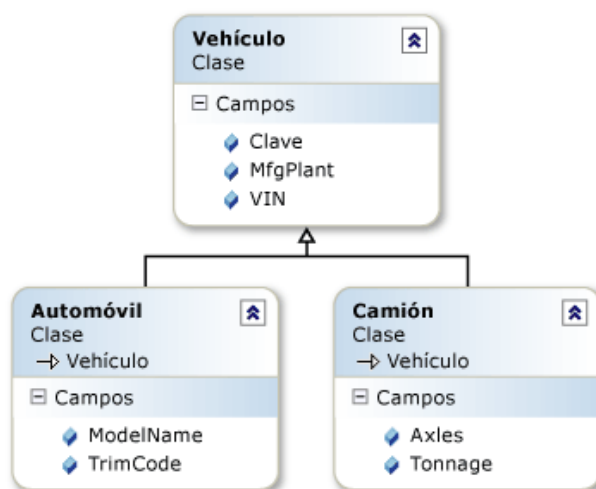


Imagen 3.5

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Herencia	http://cecilia-sanchez.blogspot.com/2010/10/herencia-y-polimorfismo.html	Imágenes de plantillas de Airyelf	29/10/2011

En el mundo natural, observamos que los hijos y los padres comparten características en común. Igualmente, se heredan ciertas capacidades. Nuestro padre o madre pueden ser consumados músicos de renombre, brillantes literatos o fantásticos expertos en pedagogía contemporánea y los hijos en algunas ocasiones, heredan igualmente estas grandes habilidades siendo incluso mucho mejores que sus padres en el desempeño de estas potencialidades.



Imagen 3.6

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
madre	Imágenes prediseñadas de Word	Microsoft word	29/10/2011

Con los objetos pasa algo similar, puesto que las clases en Java como característica clave del paradigma orientado a objetos pueden transmitir tanto sus atributos como sus métodos a otras clases que se definan como sucesoras o herederas de aquellas. Dentro de este modelo, se tiene que las clases heredan unas de otras, siendo la más importante de todas, la Clase Object o la clase genérica Objeto de la cual extienden o heredan todas las demás. Es por ello que una clase que hereda de otra debe llevar la siguiente sintaxis:

```
public class ClaseHija extends SuperClasePadre
```

Por ello, existen muchas clases del tipo `public class AppletGraficas extends Applet`, lo cual indica que la clase `AppletGraficas` es una clase que hereda todas las características y métodos de la clase general Java la cual permite implementar Applets o aplicaciones de tipo gráfico para navegadores HTML. Igual ocurre con otras clases, y por tanto todas comparten la capacidad de heredar y de usar los métodos de sus clases predecesoras, en aras de hacer más completo el proceso de uso de las funcionalidades y de programación de las aplicaciones que ellas pueden llegar a componer.

El objetivo de la clase `ClaseHija` es el de heredar de su padre ciertas funcionalidades, como el poder también realizar la suma de los números reales, función que ya está previamente implementada en su clase padre. No obstante, ella realiza una curiosa implementación, al igual que lo hace su clase hermana `ClaseHijaSobrescrita`. Finalmente, existe una última clase denominada `ClaseNieta`, que hereda de `ClaseHija`, convirtiéndose ella también en una heredera de la `SuperClasePadre`, la cual se convierte en el “abuelo” de esta. A continuación se muestra la implementación y lo que hace cada una de estas clases, para que se comprendan con mayor claridad los mecanismos de funcionamiento de la herencia.

```
public class Ventana {  
    protected int x;  
    protected int y;  
    protected int ancho;  
    protected int alto;  
    public Ventana(int x, int y, int ancho, int alto) {  
        this.x=x;  
        this.y=y;  
        this.ancho=ancho;  
        this.alto=alto;  
    }  
    public void mostrar(){  
        System.out.println("posición : x="+x+", y="+y);  
        System.out.println("dimensiones : w="+ancho+", h="+alto);  
    }  
    public void cambiarDimensiones(int dw, int dh){  
        ancho+=dw;
```

```
    alto+=dh;  
  }  
}
```

Todo método de encapsulamiento sigue siempre la misma filosofía:

```
private tipo_de_dato nombreAtributo;
```

Por lo tanto, la definición de sus métodos get (recuperación) y set (configuración) es la siguiente:

a) Método de recuperación del estado actual de valor de la variable de instancia o getter: Debe incluir su visibilidad, el tipo de retorno del método que debe ser del mismo tipo de dato del que esté constituido el atributo y el nombre `getNombreAtributo`. Dicho método no recibe parametrización. Así, si tuviéramos un atributo del tipo:

```
private int numeroEmpleados;  
Su método getter sería por lo tanto:  
private int getNumeroEmpleados ()  
{  
    return (numeroEmpleados);  
}
```

Y la responsabilidad de este método radicaría en devolver el valor actual de la variable `numeroEmpleados`, la cual previamente pudo haber sido modificada en alguna asignación o cálculo ejecutado por otro método de la clase.

b) Método de configuración o puesta de valor en el atributo actual o variable de instancia de la clase (setter): Como su nombre lo indica, el objetivo de este método es asignar un valor para la actualización o refresco del valor de la variable de instancia. Dado que es un método que se encarga de poner un valor a la variable, su signatura sería la siguiente:

```
public void setNumeroEmpleados (int numeroEmpleados)  
{  
    this.numeroEmpleados = numeroEmpleados  
}
```

Y se muestra que tanto el parámetro como el atributo se llaman de la misma forma. Esto aparentemente es un error de sintaxis, pero en realidad no lo es, puesto que como se observa en el código de la implementación tenemos que `this.numeroEmpleados = numeroEmpleados`. La palabra `this`, es conocida en Java como el operador de referenciación y hace parte fundamental de los elementos que permiten diferenciar variables de instancia de otras variables en la construcción de la clase. Este indicador u operador le dice al compilador que la variable `numeroEmpleados` es el atributo de la clase, mientras que la otra variable `numeroEmpleados` que se encuentra en el parámetro del método, se compila como un parámetro que en este caso, proporciona un valor de

actualización para el atributo o variable de instancia de la clase. Mediante el operador de referenciación podemos invocar dentro del código de otros métodos dentro de la misma clase, todos los métodos y atributos de esta, con el fin de referenciar que pertenecen a ella y no a otros objetos. De esta forma, si otra clase utiliza el mismo nombre de método, al anteponer el operador `this`, el compilador sabe diferenciar a qué clase corresponde qué método.

Finalmente, se encuentran funcionalidades que son implementadas y hechas por el padre, y que para nuestro caso, son funcionalidades que deberán ser heredadas por todas las clases que sean hijas de él dentro de la jerarquía de herencia.

```
public boolean calcularValorVerdad (boolean v1, boolean v2)
{
    boolean resultado = false;
        try
            {
                if (v1 == true && v2 == true)
                {
                    resultado = true;
                }
            }
        catch (Exception error)
        {
            error.toString(); error.printStackTrace(); return (false);
        }
    return (resultado);
}
```

Lo anterior indica que la clase se encarga de sobrescribir el comportamiento de la clase padre a nivel de este método.

¿Qué es una Interface?

(Definir un Interface, Sobrecarga de métodos)

Muchas veces se hace necesario establecer métodos y funcionalidades que no dependan de un esquema específico de implementación establecido en la jerarquía de la herencia. Tomemos como ejemplo las estructuras de datos. Estas son básicamente colecciones de registros que pueden ser de un tipo de dato específico o de un tipo de dato abstracto (como un objeto) y que cumplen funcionalidades comunes tales como poder ingresar datos a la estructura, retirarlos de la estructura o contabilizar la cantidad de elementos en ella almacenados. No obstante, existen muchas estructuras como las colas, las pilas o las listas, que aunque tienen estas funcionalidades

en común, su algoritmo de implementación interna es bien diferente. Si quisiéramos utilizar la herencia para crear diferentes estructuras de datos, tendríamos que sobrescribir los métodos en todas las diferentes estructuras, lo cual haría que la implementación global de todas estas fuera bastante compleja.

Por ello, Java ofrece el concepto de la Interfaz, la cual no debe confundirse con la Interfaz Gráfica de Usuario (GUI) de una aplicación. Cuando se habla de interfaz, se hace referencia a una colección de métodos específicos que no exponen ninguna implementación específica para el programador y de una colección de valores constantes, pero que si determinan qué parametrización y qué retornos deben establecerse para que otras clases si se encarguen de realizar la implementación de estas interfaces. Las interfaces se utilizan para definir un protocolo de comportamiento que puede ser implementado por cualquier clase del árbol de clases. Una interfaz se declara de la siguiente manera:

```
public interface NombreInterfaz
{
//Datos manejados por la interfaz tipoDato nombreValor;
//Métodos expuestos por la interfaz
visibilidad tipoRetorno nombreMetodo (parámetros) captura
Excepción;
}
```

Así, para instanciar una interfaz a partir de un objeto se hace necesario seguir estos pasos:

- Declarar una variable del tipo de la interfaz.
- Crear una instancia de la clase implementadora que implementa la interfaz.
- Hacer un casting (o moldeo) al tipo de la clase implementadora para poder instanciar la interfaz

Así, para probar el funcionamiento de la interfaz, tendríamos que ejecutar la siguiente secuencia de pasos.

```
ClaseImplementadoraInterfaz objetoClaseImplementadora = new
ClaseImplementadoraInterfaz();
Interfaz objetoInterfaz = (ClaseImplementadoraInterfaz)
objetoClaseImplementadora;
```

EJERCICIOS DE AUTOEVALUACIÓN

1. Revisar errores en los siguientes códigos

```
class A {  
    int i, j;  
    A(int a, int b) {  
        i = a;  
        j = b;  
    }  
    void show() {  
        System.out.println("i y j: " + i + " " + j);  
    }  
}  
  
class B extends A {  
    int k;  
    B(int a, int b, int c) {  
        super(a, b);  
        k = c;  
    }  
    // muestra k -- sobrescribe el metodo show() de A  
    void show() {  
        System.out.println("k: " + k);  
    }  
}  
  
class Override {  
    public static void main(String args[]) {  
        B subOb = new B(1, 2, 3);  
        subOb.show(); // llama al metodo show() de B  
    }  
}  
  
class Figure {  
    double dim1;  
    double dim2;  
    Figure(double a, double b) {  
        dim1 = a;  
        dim2 = b;  
    }  
}
```



```
double area() {
    System.out.println("Area para Figure es indefinida.");
    return 0;
}
}
class Rectangle extends Figure {
    Rectangle(double a, double b) {
        super(a, b);
    }
    double area() {
        System.out.println("Dentro de Area para Rectangle.");
        return dim1 * dim2;
    }
}
class Triangle extends Figure {
    Triangle(double a, double b) {
        super(a, b);
    }
    double area() {
        System.out.println("Dentro de Area para Triangle.");
        return dim1 * dim2 / 2;
    }
}
class FindAreas {
    public static void main(String args[]) {
        Figure f = new Figure(10, 10);
        Rectangle r = new Rectangle(9, 5);
        Triangle t = new Triangle(10, 8);
        Figure figref;
        figref = r;
        System.out.println("Area es " + figref.area());
        figref = t;
        System.out.println("Area es " + figref.area());
        figref = f;
        System.out.println("Area es " + figref.area());
    }
}
```

2. Desarrolle una Clase Java que implemente un objeto de tipo Avión. Defina los atributos necesarios para indicar el número de pasajeros, la cantidad de combustible a cargar en el avión y el trayecto que

el avión debe recorrer entre un punto origen y un punto destino. Utilizando los métodos de la clase genérica Object, cree dos instancias de aviones y desarrolle un método que indique si dos vuelos van al mismo destino. Si esto es cierto, cambie el destino al segundo vuelo y traslade la mitad de los pasajeros del primer vuelo al segundo vuelo. (Recuerde utilizar los métodos de encapsulamiento de los atributos para poder recuperar los datos de cada objeto y así poder realizar las operaciones solicitadas por este problema).

3. Elabore una clase Java que lea 2 números enteros y determine mediante la estructura de decisión simple, cuál de los dos números es el mayor y cuál es el menor.

4. Utilizando el esquema de decisiones de tipo anidado, construya una Clase Java que permita leer el valor de un producto y que calcule el valor de su IVA (16%). Si dicho valor del IVA se encuentra entre 1000 y 2000 pesos, debe asignarle un descuento al producto del 2% sobre el valor del IVA. Si el IVA está entre 2000 y 5000, se la aplica un descuento del 6% al valor de dicho IVA y finalmente si el producto vale más de 5000, se le hace un descuento sobre el valor del 6% del IVA calculado. El precio neto del producto será igual al precio neto más el IVA menos el descuento obtenido.

Pista de aprendizaje:

Tener en cuenta: es un tema que debemos aplicar solo cuando tengamos experiencia dentro de la herramienta.

Tenga presente: si utilizamos estos recursos muy temprano podrá generar confusiones y una experiencia no muy amena en la herramienta.

Traer a la memoria: que debemos de conocer a fondo los conceptos previos que nos lleven a profundizar en los procesos siguientes.

4. ALGORITMOS DE BUSQUEDA Y ORDENAMIENTO MODELO DE VENTANAS E INTERFAZ

http://www.youtube.com/watch?v=7URpcSZK_8o

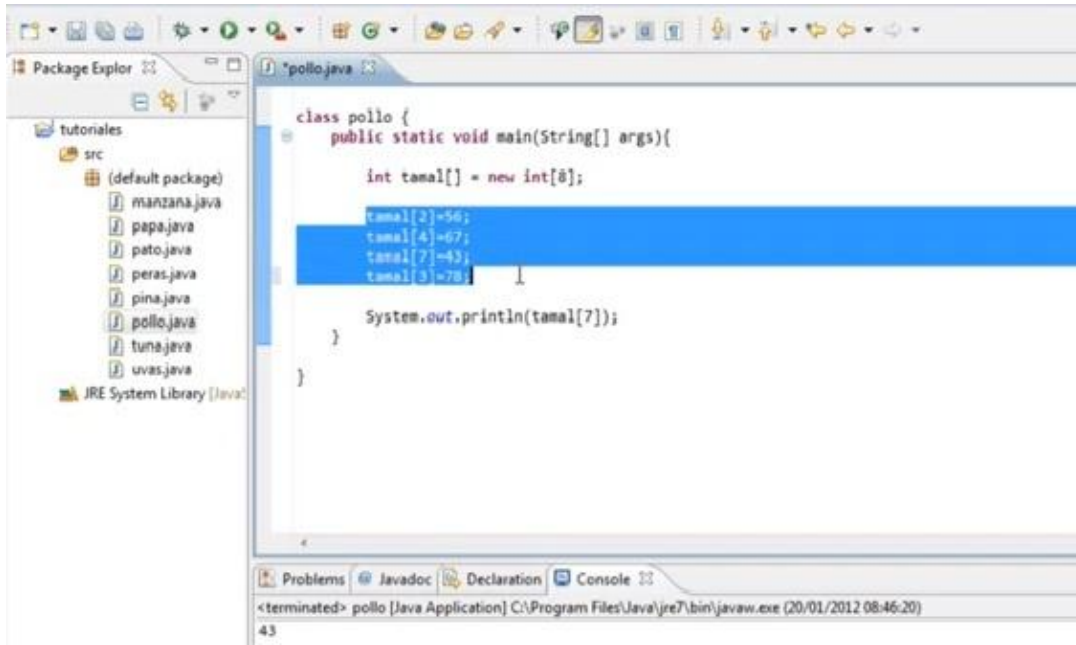


Imagen relacionada del video de youtube

OBJETIVO GENERAL

Entender qué es una GUI (Interfaz Gráfica de Usuario), definiendo su importancia en la implementación de aplicaciones que interactúan gráficamente con el usuario final.

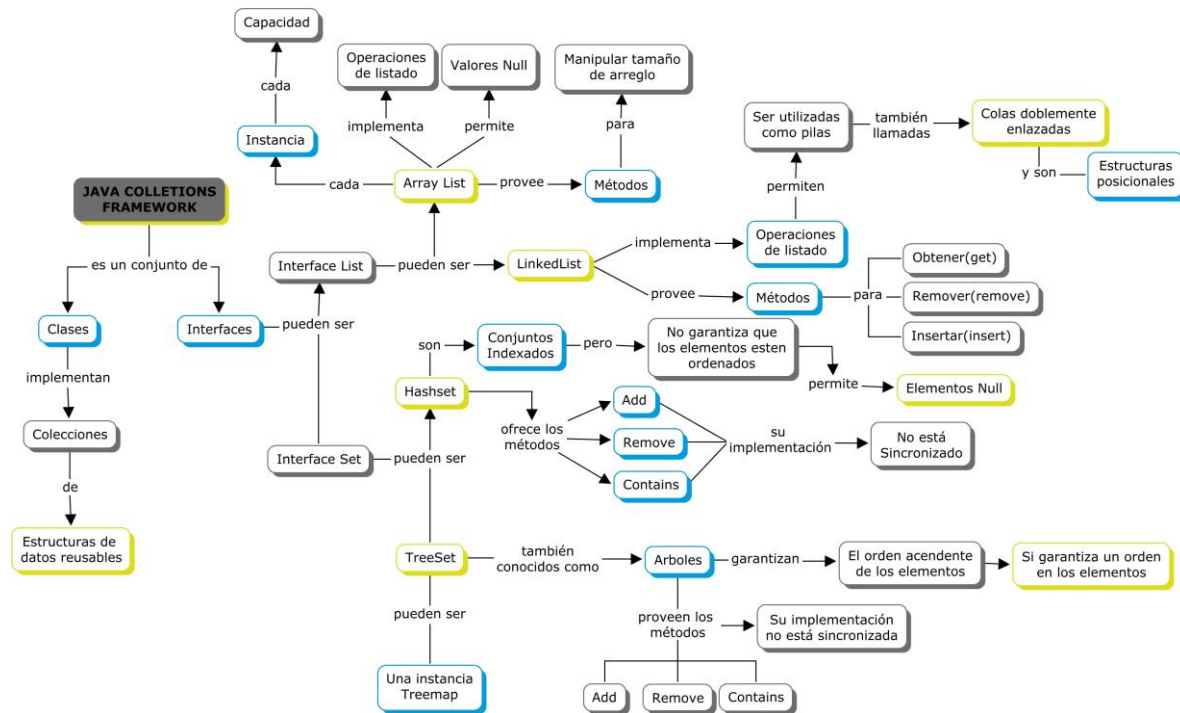
OBJETIVOS ESPECÍFICOS

- ❑ Desarrollar habilidades con los componentes que brinda el sistema.
- ❑ Identificar las características gráficas, herramientas que le darán un ambiente más profesional y de gran atracción al usuario final que maneje el aplicativo.
- ❑ Estudiar las técnicas que debe tener un aplicativo de mayor envergadura.

4.1. Prueba inicial

1. Cree una clase Java para leer un arreglo de datos booleanos (valores true y false) y que imprima el arreglo, pero que en vez de que salgan los datos true y false en la impresión, aparezcan las letras V por cada valor true y F por cada valor false.
2. Construya una clase Java que cree un arreglo de caracteres (chars) y lo pueble con las letras del alfabeto de manera automática. Recuerde utilizar el concepto del código ASCII, el uso de casting y el manejo de la sentencia repeticional for o while para poblar el arreglo mediante el uso de un método, para que el usuario no tenga que digitar las 27 letras del alfabeto (recuerde que se debe incluir la Ñ junto las con las 26 letras estándar del alfabeto occidental latino).

4.2. Relación de Conceptos



Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Java Collection Framework	http://code.google.com/p/provectolispuvg/downloads/detail?name=JCF.jpg&can=2&q=	Diana ortiz	29/11/2011

4.3. Algoritmos de búsqueda de datos de vectores

A medida que se van desarrollando las aplicaciones escritas en el lenguaje Java, ya se tiene a nuestra disposición el concepto del dato primitivo como mecanismo para el almacenamiento de datos de manera individual, de los ocho tipos primitivos existentes (byte, short, char, int, float, double, long y boolean). No obstante, gracias al concepto del Javabeen o value object, sabemos que existen datos de tipo abstracto o de tipo de clase que permiten almacenar infinitud de registros, en virtud de la cantidad de atributos o variables de instancia establecidos en dichas clases creadas por el programador. Pero qué ocurre, cuando se hace necesario almacenar grandes cantidades de datos de un tipo o incluso de varios tipos, como lo que podría ser un listado enorme de números o una colección muy grande de cadenas de caracteres? Ejemplos puntuales se pueden encontrar en el contenido de una agenda telefónica o en una tabla de datos de un proceso estadístico.

Todo arreglo siempre tiene tres características:

- Una dimensión, es decir, una longitud o cantidad total de datos que lo forman.
- Una indización, que indica la posición de cada elemento dentro de la estructura.
- Un tipo, que define qué clases de datos se van a almacenar en él.

Cuando no tenemos certeza de si los datos van a ser del mismo tipo para todas las posiciones, lenguajes como Java ofrecen la maravilla de contar con arreglos de tipo Object (objeto) para poder almacenar en las posiciones de un arreglo cualquier tipo dato, incluyendo por supuesto datos de tipo objeto.

Si se quisiera crear un arreglo como el del ejemplo, debemos instanciar un nuevo array o arreglo de 6 posiciones, mediante la instrucción de instanciación:

```
int [] arregloNumerosEnteros = new int[6];
```

Donde los corchetes ([]) le indican al compilador que debe crearse un arreglo de un tipo específico de datos (int). El uso del constructor int [] con un valor paramétrico entre los corchetes (6) le indica al compilador que en tiempo de ejecución debe construirse un arreglo de 6 posiciones y nada más.

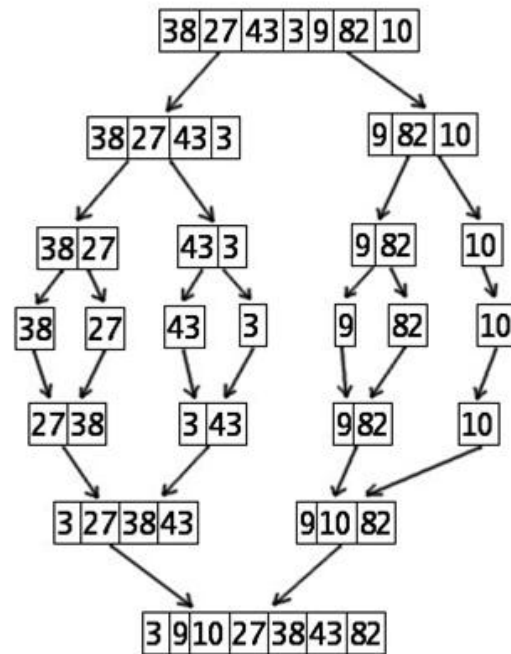


Imagen 4.1

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ordenamiento de datos	http://estructuras-de-datos.wikispaces.com/Ordenamiento+por+Mezcla	wipispaces	29/10/2011

Utilizar la palabra clave "new" y un Inicializador

El uso de la palabra new como operador de instanciación de objetos es una herramienta fundamental para la creación de nuevos elementos a lo largo del ciclo de vida de una aplicación.

Gracias a ella se realiza a nivel de computación, el proceso de reservación de memoria necesario para que la aplicación pueda definir los tamaños específicos en bytes y kilobytes que habrán de ser consumidos por los recursos (variables, constantes e instrucciones) desarrolladas a lo largo del código de todos los métodos y atributos definidos en la clase. Es importante tener en cuenta que el proceso de reservación de memoria debe realizarse de acuerdo con la siguiente estructura de declaración de instrucciones:

```

Constructor nombreNuevoObjeto = new Constructor();
o
Constructor nombreNuevoObjeto = new
Constructor(Párametros del Constructor);
  
```

Y en este caso, en el de la declaración de nuevos arreglos

```
TipoDatoDelArreglo [] nombreNuevoArreglo = new  
TipoDatoArreglo[dimensiones];
```

Así, si se desea crear un arreglo de números enteros de 20 posiciones, debemos escribir:

```
int [] arregloDatosEnteros = new int [20];
```

Igualmente, si se quiere construir un arreglo de cadenas de caracteres de n posiciones, entonces se deberá generar una instanciación de la forma:

```
String [] coleccionCadenasCaracteres = new String[n];
```

Donde n es una variable previamente declarada y que en algún punto de alguno de los métodos de la clase fue debidamente almacenada con un valor leído desde una fuente externa al programa (como por ejemplo el teclado o de un archivo de texto, etc.).

EJERCICIO DE AUTOEVALUACIÓN

A continuación debe implementar una clase para el manejo de datos de un arreglo unidimensional

```
import javax.swing.JOptionPane;  
/*****/  
/*****/  
public class RecorridoArreglos  
{  
/*****/  
/*****/  
//Datos  
private int [] datos;  
/*****/  
/*****/  
public RecorridoArreglos()  
{  
// TODO Auto-generated constructor stub  
}  
  
/*****/  
/*****/
```

```
public RecorridoArreglos(int numeroElementos)
{
// TODO Auto-generated constructor stub
datos = new int[numeroElementos];
}

public void poblarArregloConDatos (int numeroDatos)
{
int i = 0;

        try
        {
for (i = 0; i < numeroDatos; i++)
{
datos[i] = (int)(Math.random()*10000);
}
}
catch (Exception errorPoblado)
{
JOptionPane.showMessageDialog(null,"Error de poblamiento: " + errorPoblado.getMessage());
errorPoblado.printStackTrace();
}

}

public void recorrerArregloConWhile ()
{
JOptionPane.showMessageDialog(null,"Recorrido de un arreglo con un while");
int i = 0;
int dato = 0;
String arreglo = "";

        {

i = 0;
while (i < datos.length)
{
dato = datos[i];
arreglo += String.valueOf(dato);
arreglo += " "; i++;
} //Fin del while

JOptionPane.showMessageDialog(null,arreglo);

}

catch (ArrayIndexOutOfBoundsException errorDesbordamiento)
{
}
```



```
JOptionPane.showMessageDialog(null,"Error de
desbordamiento: " +
errorDesbordamiento.getLocalizedMessage());
errorDesbordamiento.printStackTrace();
}

}

public void recorrerArregloConDoWhile (){
JOptionPane.showMessageDialog(null,"Recorrido de un arreglo con un do-while");
int i = 0;
int dato = 0;
String arreglo = "";
    Try
    {
i = 0;
do
{
dato = datos[i];
arreglo += String.valueOf(dato);
arreglo += " ";
i++;
} while (i < datos.length);
JOptionPane.showMessageDialog(null,arreglo);

}
catch (ArrayIndexOutOfBoundsException errorDesbordamiento)
{
JOptionPane.showMessageDialog(null,"Error de desbordamiento: " +
errorDesbordamiento.getLocalizedMessage());
errorDesbordamiento.printStackTrace();
}

}

public void recorrerArregloConFor ()
{
JOptionPane.showMessageDialog(null,"Recorrido de un arreglo con un ciclo for");
int i = 0;
int dato = 0;
String arreglo = "";
try
```

```
{
for (i = 0; i < datos.length; i++)
{
dato = datos[i];
arreglo += String.valueOf(dato);
arreglo += " ";
}
JOptionPane.showMessageDialog(null,arreglo);

}
catch (ArrayIndexOutOfBoundsException errorDesbordamiento)
{ JOptionPane.showMessageDialog(null,"Error de desbordamiento: " +
errorDesbordamiento.getLocalizedMessage());
errorDesbordamiento.printStackTrace();
}
}
public static void main(String[] args)
{
// TODO Auto-generated method stub int numeroDatos = 0;
numeroDatos = Integer.parseInt(JOptionPane.showInputDialog(null,"Ingrese el
número de elementos del arreglo: "));
RecorridoArreglos recorredorArreglos = new RecorridoArreglos(numeroDatos);

//Lo poblamos de datos recorredorArreglos.poblarArregloConDatos(numeroDatos);
//Lo recorremos con las tres estructuras
//while recorredorArreglos.recorrerArregloConWhile();

//Lo poblamos de datos recorredorArreglos.poblarArregloConDatos(numeroDatos);
//do-while recorredorArreglos.recorrerArregloConDoWhile();

//Lo poblamos de datos recorredorArreglos.poblarArregloConDatos(numeroDatos);
//for recorredorArreglos.recorrerArregloConFor();
} //Fin del main
/*****/
/*****/
}
```

Algoritmos de búsqueda de datos en vectores y métodos de ordenamiento

En ocasiones se hace necesario localizar datos dentro de un arreglo, puesto que ese es el principio básico que todo sistema de búsqueda de información debe implementar como requerimiento funcional mínimo. Debe tenerse en cuenta que la búsqueda es un proceso de comparación que debe verificar si el dato se encuentra efectivamente o no dentro de la estructura de datos, pero dado que un arreglo podría tener una cantidad enorme de elementos, sería altamente ineficiente realizar la comparación por todas las posiciones del arreglo de datos. Por ello, el principio reza que una vez el dato ha sido encontrado debe detenerse el ciclo y salirse del flujo de control del ciclo de búsqueda del dato.

Las búsquedas de datos pueden realizarse mediante dos procesos: el lineal y el binario.

a) Búsqueda Lineal: se utiliza cuando el contenido del vector no se encuentra o no puede ser ordenado. Consiste en buscar el elemento comparándolo secuencialmente (de ahí su nombre) con cada elemento del arreglo hasta que éste se encuentre, o hasta que se llegue al final del arreglo. La existencia se puede asegurar desde el momento que el elemento es localizado, pero no podemos asegurar la no existencia hasta no haber analizado todos los elementos del arreglo.

```
public boolean buscarElementoAlgoritmoLineal (int [] datos, int elementoABuscar)
{
    int i = 0;
    boolean estaEnElArreglo = false;
    try
    {
        for (i = 0; i < datos.length; i++)
        {
            if (datos[i] == elementoABuscar)
            {
                estaEnElArreglo = true;
                //Si lo encontré paramos el ciclo break;
            }
        }
        //Fin del for
    }
    catch (Exception errorBusquedaLineal)
    {
        JOptionPane.showMessageDialog(null,"Error encontrando el dato: " +
        errorBusquedaLineal.toString()); errorBusquedaLineal.printStackTrace();
        return (false);
    }
}
```

```

return (estaEnElArreglo);
}
    
```

b) Búsqueda Binaria (Proceso de Divide y Vencerás): se utiliza cuando el vector en el que queremos determinar la existencia o no de un elemento está ordenado, o puede estarlo. Este algoritmo reduce el tiempo de búsqueda considerablemente, ya que disminuye exponencialmente con el número de iteraciones.

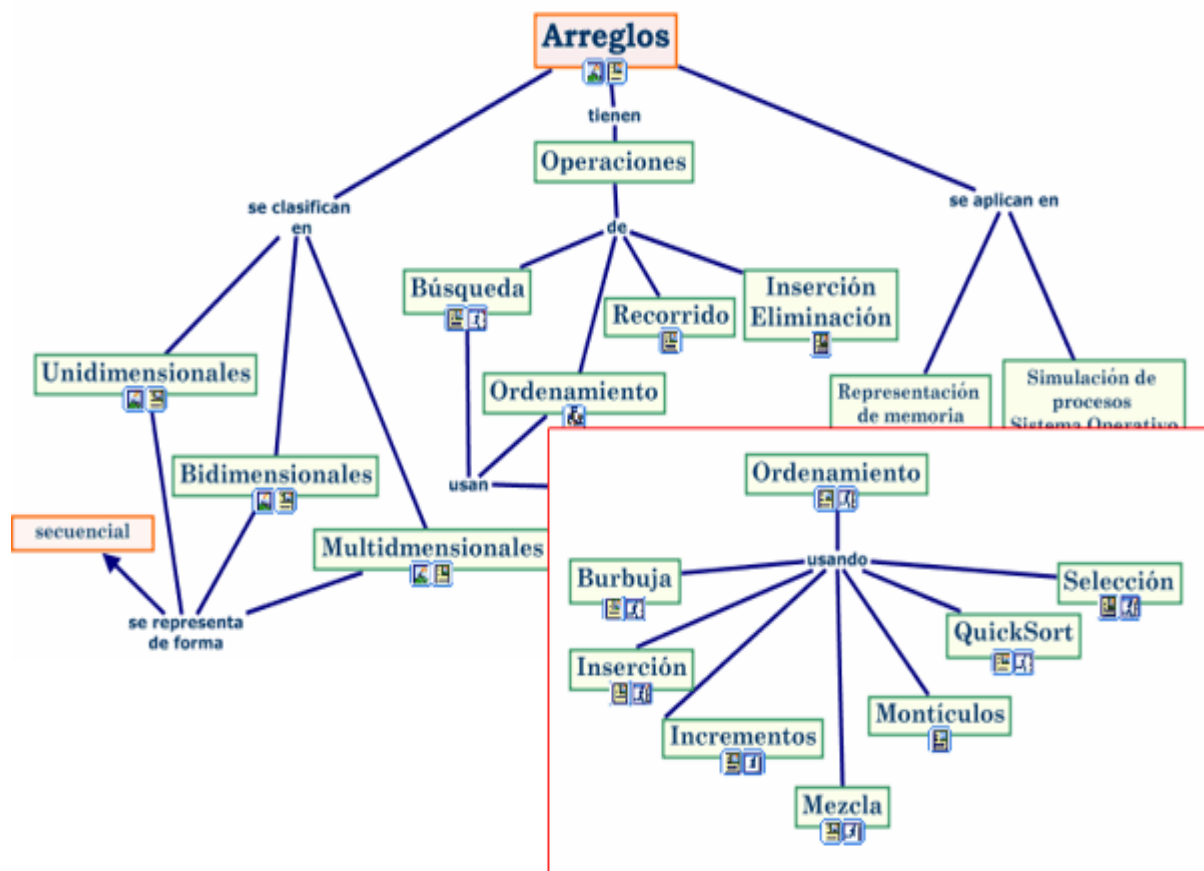


Imagen 4.2

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Métodos de ordenamiento de datos	http://www.nosolousabilidad.com/articulos/mapas_conceptuales.htm	Yolanda Soler Pellicer	29/10/2011

Pista de aprendizaje:

Tener en cuenta: los vectores son arreglos estáticos, que manejan información temporal en la memoria.

Tenga presente: son herramientas muy útiles para desarrollar la lógica y en muchos casos para complemento de aplicativos.

Traer a la memoria: la forma de declararlos y recorrerlos.

4.4. Creación de interfaz Gráfica y comparación con las aplicaciones de consola

Sobre el JFC y Swing

Hasta el momento, en los ejercicios previos, ejemplos e implementaciones, hemos observado el desarrollo de aplicaciones que utilizan el componente gráfico `JOptionPane` para desplegar diferentes ventanas o cuadros de diálogo (algunos de ellos para impresión y otros para ingreso de datos), que nos permiten llevar la esencia lineal y estructurada de la captura de datos y despliegue de resultados que la lógica de negocio de nuestros métodos han venido desarrollando. No obstante, si hablamos de aplicaciones que deban ser más dinámicas para los usuarios, es decir, que tengan que desplegar una gran cantidad de interfaces gráficas para requerir la captura de datos y el despliegue de resultados, este tipo de componentes se nos hacen cortos y pobres para lograr el objetivo deseado.

Por ello, Sun a través de las APIs de la Plataforma Java, ofrece diferentes paquetes para el desarrollo de las GUI (Graphical User Interfaces, Interfaces Gráficas de Usuario), que como se señaló en el fascículo III, no deben confundirse con las interfaces del lenguaje (las que ya sabemos muy bien que son colecciones de métodos para favorecer la implementación a partir de diferentes clases con base en diferentes lógicas de negocio). Entre las APIs más conocidas encontramos los proyectos AWT (Abstract Windowing Toolkit), Swing y ahora, Look and Feel, que ofrecen una colección rica y variada de métodos, objetos y propiedades para el diseño de ventanas gráficas, como la que se muestra a continuación:



Imagen 4.2

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ejemplo de una ventana desarrollada bajo Swing y con los controles de la API Look and Feel.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Para facilitar el desarrollo de este tipo de ambientes visuales, que generan lo que se conoce como la capa de presentación o interfaz gráfica de una aplicación informática, Java, utiliza el desarrollo de este tipo de programas empleando lo que se conoce como JFC (Java Foundation Classes) y las APIs Swing o AWT, cuyas diferencias estriban más que todo en la forma de presentar los bordes, colores, tramas y diseños de los controles gráficos de presentación y en algunos métodos necesarios para generar los mecanismos de control de dichos componentes y su respuesta de interacción ante los eventos de manipulación gestionados por el usuario.

JFC es la abreviatura de Java Foundation Classes, que comprende un grupo de características para ayudar a construir interfaces gráficos de usuario (GUIs). En el desarrollo de este proyecto, encontramos las siguientes APIs que refuerzan el desarrollo de las aplicaciones gráficas bajo Java.

Los componentes Swing: incluye todo desde botones hasta splitpanes o tablas.

b) Soporte de Aspecto y Comportamiento Conectable: le ofrece a cualquier componente Swing una amplia selección de aspectos y comportamientos. Por ejemplo, el mismo programa puede usar el Aspecto y Comportamiento Java o el Aspecto y Comportamiento Windows.

c) API de Accesibilidad: permite tecnologías que posibilitan asistir a los desarrolladores en tareas como el desarrollo de pantalla y display Braille para obtener información desde la interfaz de usuario.

- d) Java 2D API (sólo JDK 1.2 en adelante): permite a los desarrolladores incorporar fácilmente gráficos 2D de alta calidad, texto, e imágenes en aplicaciones y applets Java.
- e) Soporte de Drag and Drop (sólo JDK 1.2 en adelante): Proporciona la habilidad de arrastrar y soltar entre aplicaciones Java y aplicaciones nativas.

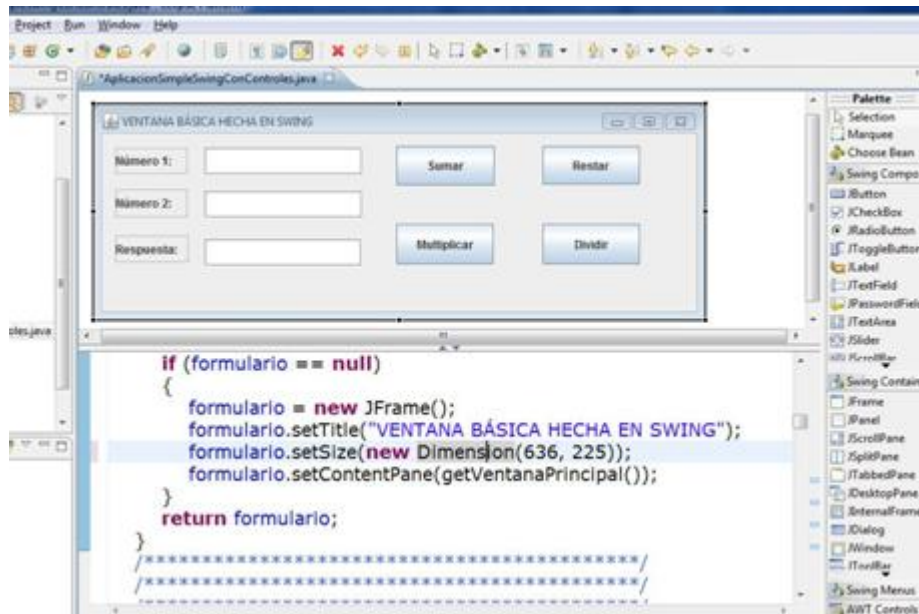


Imagen 4.3

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Diseño de la GUI para una aplicación de cálculo de operaciones matemáticas.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Visita Rápida por el Código de un Programa Swing

Las aplicaciones hechas en modo visual poseen una estructura formada por los siguientes elementos a saber:

- Un contenedor gráfico principal conocido como Frame (objeto JFrame) que define los límites de la ventana.
- Un panel (un objeto JPanel) donde se agregan todos los contenedores gráficos y que se inserta en el interior del frame de la ventana.
- Un conjunto de controles gráficos para la manipulación de los datos que la aplicación debe manipular.
- El código de los métodos de interacción del usuario con la aplicación y los métodos de lógica de negocio necesarios para el procesamiento.

EJERCICIOS DE AUTOEVALUACIÓN

Implementar el siguiente código

```
package interfazgrafica;
import javax.swing.JFrame;
import javax.swing.JPanel;
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.*;

public class AplicacionSimpleSwing
{
    //Atributos de la clase
    private JFrame formulario = null;
    private JPanel ventanaPrincipal = null;
    /**/
    //Constructor de la clase
    public AplicacionSimpleSwing()
    {
        //Creamos un nuevo frame
        JFrame ventana = getFormulario();
        ventana.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
        //Definimos el tamaño de la ventana ventana.setSize(800,600);
        //La mostramos ventana.setVisible(true);
    }
    private JFrame getFormulario()
    {
        if (formulario == null)
        {
            formulario = new JFrame();
            formulario.setSize(new Dimension(636, 367));
            formulario.setTitle("VENTANA BÁSICA HECHA EN SWING");
        }
    }
}
```



```
formulario.setContentPane(getVentanaPrincipal());return formulario;
}
private JPanel getVentanaPrincipal()
{
if (ventanaPrincipal == null)
{
ventanaPrincipal = new JPanel();
ventanaPrincipal.setLayout(new BorderLayout());
}
return ventanaPrincipal;
}
public static void main(String [] args)
{
/*****/
try
{
new AplicacionSimpleSwing();
}

catch (Exception errorMain)
{
errorMain.toString();
errorMain.printStackTrace();
}
/*****/
} //Fin del main
/*****/
} //Fin de la clase
```



Imagen 4.4

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ventana generada por la aplicación AplicacionSimpleSwing.java.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Herencia de Componentes y Contenedores

La API Swing usa precisamente el concepto de herencia para manejar el desarrollo de los contenedores gráficos

a) Contenedores de Alto Nivel: representan estructuras de ventana que almacenan todos los componentes principales. En el caso que nos compete, representa los objetos de tipo Panel, Frame y Window que permiten desplegar ventanas para ser ejecutadas en programas bajo entornos gráficos de Sistemas Operativos como Windows o GNU Linux. Dichos contenedores deben tener asociados igualmente manejadores de eventos que permitan controlar las operaciones como la maximización o minimización de la ventana o el proceso de cierre de la aplicación en caso de que el usuario pulse el botón X de la ventana en su panel lateral derecho de botones de control.

b) Contenedores intermedios: representan contenedores que se agregan en el interior de contenedores principales y se utilizan para realizar el proceso de distribución de componentes más pequeños como botones, cuadros de texto, listas de selección desplegables o marcas de verificación, entre otros. Dichos contenedores permiten establecer a su vez la forma arquitectónica de desplegar los componentes, ya que Java ofrece diferentes formas de desplegar los componentes gráficos (por ejemplo, es posible colocar coordenadas absolutas sobre la superficie del

contenedor intermedio, que se denomina técnicamente como Layout (diseño) o hacer que los controles se coloquen en forma secuencial uno detrás de otro. Esto es lo que hace que sobre el componente intermedio se defina la propiedad del Layout mediante opciones como BorderLayout (con distribución de coordenadas) o null (sin ningún tipo de parametrización de coordenadas), o GridLayout para establecer una retícula de columnas y filas sobre la cual colocar cada elemento gráfico (botón, lista, cuadro de texto, etc.)).

c) Componentes atómicos: representan fundamentalmente los controles de la GUI que permiten al usuario la interacción con la capa de ejecución de la solución. Los controles son fundamentalmente elementos gráficos individuales que permiten el ingreso de los datos y la modificación de los mismos, cuando dichos datos se cargan sobre ellos

```
//Creamos la ventana principal
frame = new JFrame(...);

//Creamos los componentes atómicos para la ventana
button = new JButton(...);
label = new JLabel(...);
pane = new JPanel();
//Los agregamos al panel que los va agregar al frame pane.add(button);
pane.add(label);

//Empaquetamos todo en el frame principal
frame.getContentPane().add(pane, BorderLayout.CENTER);
```

Control de Distribución

Control de Distribución es el proceso de determinar el tamaño y posición de los componentes. Por defecto, cada contenedor tiene un controlador de distribución (un objeto que realiza el control de la distribución de los componentes dentro del contenedor). Los componentes pueden proporcionarle al controlador de disposición sus preferencias en cuanto a tamaño y alineamiento, pero la última palabra la tiene el controlador de disposición.

La plataforma Java suministra cinco controladores de disposición comúnmente utilizados: BorderLayout, BoxLayout, FlowLayout, GridBagLayout, y GridLayout. Estos controladores de distribución están diseñados para mostrar múltiples componentes a la vez. Una sexta clase proporcionada, CardLayout, es un controlador de disposición de propósito general usado en combinación con otros controladores de distribución. Cuando no se quiere tener un controlador de distribución específico que diga cómo deben ir los componentes, el parámetro para el método setter del controlador se puede establecer con un valor nulo (null).

Siempre que se use el método add para poner un componente en un contenedor, debemos tener en cuenta el controlador de distribución del contenedor. Algunos controladores como BorderLayout requiere que especifiquemos la posición relativa del componente en el contenedor, usando un argumento extra para el método add. Ocasionalmente, un controlador de distribución como GridBagLayout requiere elaborados procesos de configuración. Sin embargo, muchos controladores de distribución simplemente sitúan los componentes en el orden en que fueron añadidos a su contenedor. A continuación, observaremos el desarrollo de los diferentes componentes de tipo atómico que se pueden construir gracias a la API Swing de Java:

EJERCICIOS DE AUTOEVALUACIÓN

Creación de botones: El siguiente código se encarga de instanciar sobre un Frame con un controlador de distribución BorderLayout, 4 controles (un botón de pulsación automática, un botón de conmutación, una marca de verificación y un radiobotón).

```
package interfazgrafica;
//Importaciones import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class VentanaConBotones extends JPanel
{
    private static final long serialVersionUID = 3L;
    /***** /

    //Definición del constructor que agrega los botones //al JPanel
    public VentanaConBotones()
    {
        //Botón normal o pulsado
        add( new JButton("JButton"));
        //Botón de presionado permanente
        add( new JToggleButton("JToggleButton"));
        //Marca de verificación o checkbox add( new JCheckBox("JCheckBox"));
        //Radiobotón o botón de selección única add( new JRadioButton("JRadioButton"));
    }
    public static void main(String [] args)
    { try
    {
        Ventana Con Botones panel = new VentanaConBotones();
```

//Creamos el frame

```

Ventana Con Botones panel = new VentanaConBotones();
//Creamos el frame
JFrame ventana = new JFrame();
//Agregamos el panel de botones ventana.getContentPane().add(
panel, BorderLayout.CENTER );
ventana.addWindowListener( new WindowAdapter()
{
public void windowClosing( WindowEvent evt )
{
System.exit( 0 );
}}); ventana.setTitle("EJEMPLO DEL USO DE BOTONES DE SWING");
ventana.setSize(600,400);
ventana.setVisible( true );
} //find el trye
catch (Exception errorMain)
{
errorMain.toString();
errorMain.printStackTrace();
}}
}
    
```



Imagen 4.5

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Creación de cuatro controles en un frame (botón de pulsación, botón de conmutación, marca de verificación y radiobotón).	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

b) Listas de selección desplegables: Estas listas permiten que un usuario seleccione un valor de un conjunto posible de opciones. Existen dos alternativas de listas: Los ComboBoxes que despliegan una lista cuando se hace clic sobre el control y los JList que muestran listas y habilitan una barra de desplazamiento vertical y horizontal para poder navegar a través de los ítems. Estos controles pueden ser cargados con datos estáticos (como los que pueden estar en un arreglo de

enteros (int []) o de cadenas (String [])) o con datos dinámicos, como los provenientes de un archivo o de una consulta a una base de datos relacional (Por ejemplo, un listado de proveedores, productos o nombres de empleados).

```
package interfazgrafica;
//Importaciones import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class VentanaConListasDesplegables extends JPanel
{
    /**/
    private static final long serialVersionUID = 2L;
    //Ítems de la lista desplegable static String [] etiquetas =
    {
        "Archivo","Edición","Refactorizar","Navegar","Búsqueda","Proyecto","Correr","Ventana","Ayuda"
    };
    public VentanaConListasDesplegables()
    {
        //Creamos un nuevo layout setLayout(new GridLayout(2,1));
        //Instanciamos la lista de ítems
        JList lista = new JList(etiquetas);
        //La agregamos a un Scroll Pane o panel deslizable add(new JScrollPane(lista));
        //Construimos un JComboBox
        JComboBox combo = new JComboBox();
        //Lo poblamos con los números del 0 al 99 for( int i=0; i < 100; i++ )
        {
            combo.addItem(Integer.toString(i));
        }//Agregamos el combo add(combo);
        }
        public static void main(String [] args)
        { VentanaConListasDesplegables lista = new VentanaConListasDesplegables();
          JFrame ventana = new JFrame();
          //Agregamos la lista y el combobox ventana.getContentPane().add(lista,BorderLayout.CENTER);
          ventana.addWindowListener(new WindowAdapter()
          {
              public void windowClosing( WindowEvent evt )
              { System.exit( 0 );
                }
          });
          ventana.setTitle("Manejo de Listas y ComboBoxes");
```

```

ventana.setSize(400,100);
ventana.setVisible(true);
}
/*****/
} //Fin de la clase
    
```

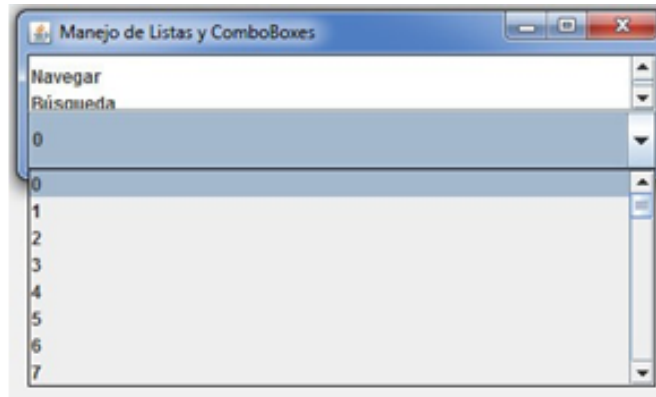


Imagen 4.6

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Aplicación que despliega un JList (parte superior) y JComboBox (parte inferior).	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Pista de aprendizaje:

Tener en cuenta: la interfaz gráfica con complementos a la lógica del problema.

Tenga presente: implementan comandos, funciones y métodos adicionales a los utilizados para solucionar un problema.

Traer a la memoria: que muchos de estos procesos los adiciona automáticamente el sistema aunque no es externo a aprender su comportamiento.

4.5. Definición de controles gráficos

A lo largo de los ejemplos que hemos visto, todas las aplicaciones que hemos desarrollado comparten una sentencia de código en común:

```
ventana.addWindowListener( new WindowAdapter()
```

```
{  
public void windowClosing( WindowEvent evt )  
{  
System.exit( 0 );  
}  
});
```

Dicha sentencia, como se observa, implementa en su interior un método denominado `windowClosing`, que recibe como parámetro un objeto de tipo `WindowEvent`, y además visualizamos que dicho método despliega la instrucción de finalización de la aplicación `System.exit(0)`. Dicha instrucción, observamos que va vinculada al frame principal o ventana que ha sido instanciada para desplegar la aplicación. Este conjunto de instrucciones asociadas a la ventana principal es lo que se conoce como un oyente o despachador de evento y es un elemento fundamental en el diseño de las interfaces gráficas de usuario de un sistema de Software.

Todo manejador de eventos requiere tres partes de código:

1) Donde se declare la clase del manejador de eventos, el código especifica que la clase o implementa un interface de oyente, o descende una clase que implementa un interface de oyente. Por ejemplo.

```
public class MiClaseManejadoraDelEvento implements ActionListener  
{  
....  
}
```

2) El código que registra un ejemplar de la clase de manejo de eventos de un oyente sobre uno o más componentes. Por ejemplo.

```
componenteAdministrar.addActionListener(new MiClaseManejadoraDelEvento);
```

3) La implementación de los métodos de la interfaz oyente. Por ejemplo:

```
public void actionPerformed(ActionEvent e)  
{  
//Código que debe ejecutarse al producirse el evento sobre el //componente  
//Como pulsar un botón o seleccionar un ítem de una lista de //selección  
//Ejemplo ejecutarTransaccionBancaria(codigoUsuario,fecha);  
}
```


Un escenario de manejo de eventos típico ocurre con los botones (JButton). Para detectar cuando el usuario pulsa un botón de la pantalla (o pulsa la tecla equivalente), un programa debe tener un objeto que implementa la interfaz ActionListener. El programa debe registrar este objeto como un oyente de acción del botón (la fuente del evento), usando el método addActionListener. Cuando el usuario pulsa el botón de la pantalla, éste dispara el evento action. Esto resulta en una llamada al método actionPerformed del oyente de action, el único método del interface ActionListener). El único argumento del método es un objeto ActionEvent que ofrece información sobre el evento y su fuente. Dentro del código del método actionPerformed se deben invocar las instrucciones o el método de lógica de negocio que debe ejecutarse en caso tal de haber sido pulsado el botón.

EJERCICIOS DE AUTOEVALUACIÓN

Escribir una interfaz gráfica que despliegue una ventana con los siguientes widgets: una etiqueta que diga "Archivo:", un campo de texto en donde el usuario pueda ingresar el nombre de un archivo y por último un botón etiquetado con la leyenda "mostrar". Solución:

```
import tools.*;
import java.awt.*;

class Type extends Program {
    // Componentes o widgets
    Frame vent;    // La ventana
    Label etiq;    // una etiqueta que dice "Archivo: "
    TextField campo; // El campo de texto
    Button boton;  // El boton

    void run() {
        // Crear ventana
        vent= new Frame("Type");
        // Establecer una politica de organizacion (layout) de la ventana
        vent.setLayout(new FlowLayout());
        // Crear los widgets
        etiq= new Label("Archivo: ");
        campo= new TextField(15);
        boton= new Button("mostrar");
        // Colocarlos en la ventana
        vent.add(etiq);
        vent.add(campo);
        vent.add(boton);
    }
}
```

```
// Mostrar la ventana
vent.pack();
vent.show();
}
}
```

Escribir un programa que despliegue una interfaz como la de Type.java pero que además incluya un area de texto en donde mostrar el contenido de un archivo. El programa debe capturar el evento de acción asociado al botón "mostrar" mostrando en el area de texto el archivo que aparezca indicado en el campo de texto.

Solución:

```
import tools.*;
import java.awt.*;
import java.awt.event.*;

class ShowFile extends Program {
    // Componentes o widgets
    Frame vent;    // La ventana
    Panel panel;   // Un panel para colocar la etiqueta, el campo y el boton
    Label etiq;    // una etiqueta que dice "Archivo: "
    TextField campo; // Un campo de texto para el ingreso del nombre del archivo
    Button boton;  // Un boton para indicar cuando se debe mostrar el archivo
    Button salir;  // Un boton para terminar la aplicacion
    TextArea texto; // Un area de texto en donde se mostrara el archivo

    void run() {
        // Crear la ventana
        vent= new Frame("Type");
        // Establecer una politica de organizacion (layout) de la ventana
        vent.setLayout(new BorderLayout());
        // Crear los widgets
        panel= new Panel();
        panel.setLayout(new FlowLayout());
        etiq= new Label("Archivo: ");
        campo= new TextField(15);
        boton= new Button("mostrar");
        salir= new Button("salir");
        texto= new TextArea(5, 15);
        // Colocarlos en el panel y la ventana
    }
}
```

```
panel.add(etiq);
panel.add(campo);
panel.add(boton);
panel.add(salir);
vent.add("Center", texto);
vent.add("South", panel);
// Agregar oyentes
ActionListener oyente= new MuestraArchivo(this);
boton.addActionListener(oyente);
campo.addActionListener(oyente);
salir.addActionListener(new Salir());
// Mostrar la ventana
vent.pack();
vent.show();
}
}
```

```
class MuestraArchivo extends Program
    implements ActionListener {
    ShowFile show;
    MuestraArchivo(ShowFile show) {
        this.show= show;
    }
    public void actionPerformed(ActionEvent e) {
        String nomArch= show.campo.getText();
        show.campo.setText("");
        TextReader lect= new TextReader(nomArch);
        StringBuffer buff= new StringBuffer();
        while(true) {
            String lin= lect.readLine();
            if (lect.eofReached())
                break;
            buff.append(lin);
            buff.append("\n");
        }
        show.texto.setText(buff.toString());
    }
}
```

```
class Salir extends Program
```

```
implements ActionListener {  
public void actionPerformed(ActionEvent e) {  
    System.exit(0);  
}  
}
```

EJERCICIOS DE AUTOEVALUACIÓN

1. Desarrolle una aplicación Java que permita desplegar un formulario de ingreso de datos para la creación de un producto y que contenga los siguientes campos. Anexo a cada dato, encontrará el control gráfico o componente atómico que debe realizar la implementación:

Nombre del producto	Campo de texto(JtextField)
Código del producto	Campo de texto(JtextField)
Fecha de ingreso	Campo de texto(JtextField)
Tipo de producto	Lista desplegable(JcomboBox) defina tres valores para dicha lista (comercial, industrial y social)
Botones	Botón aceptar, botón cancelar y botón ayuda

3. Desarrolle un formulario que permita implementar todos los controles de un formulario (Campo de texto, Lista desplegable, ComboBox, Área de Texto multilinea, radiobotón, marca de verificación y botones). Construya la especificación del formulario, con base en la definición de un caso de uso relacionado con el ingreso de los datos de un estudiante, denominado “Crear registro de un estudiante”. Identifique de acuerdo a la relación de los datos descritos en el flujo básico de eventos del caso de uso, qué dato debe corresponder a cada control del formulario.

Pista de aprendizaje:

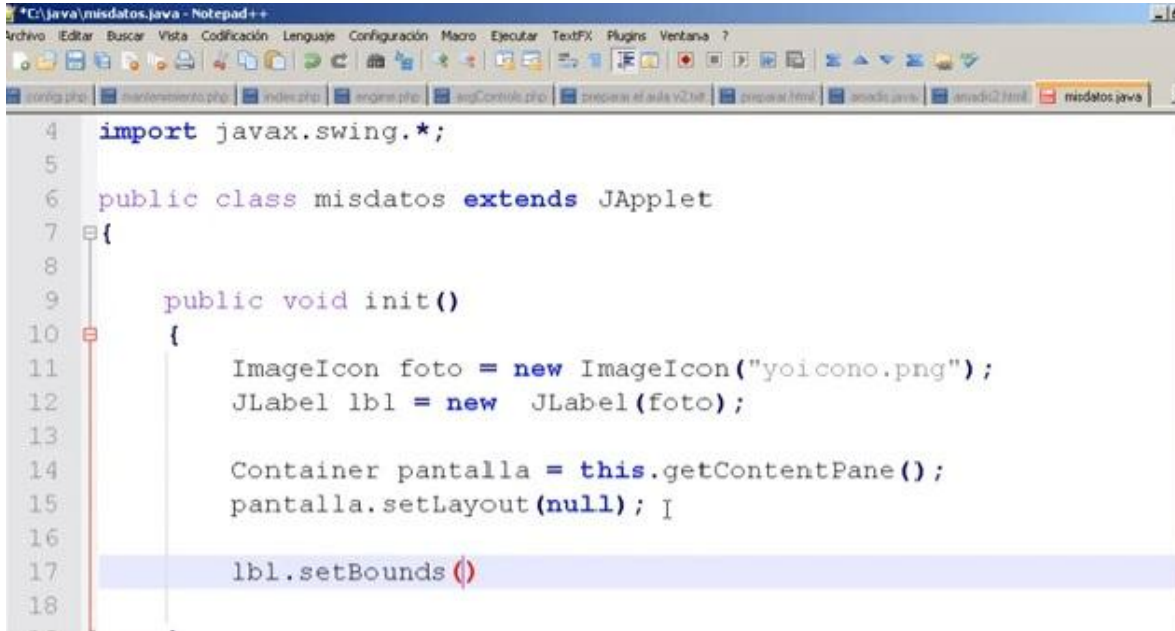
Tener en cuenta: un aplicativo es un conjunto de muchos componentes que debemos cumplir para un final apropiado.

Tenga presente: que si nos falta una etapa de desarrollo el aplicativo no es funcional y puede ser vulnerable.

Traer a la memoria: que el aplicativo es para un usuario no para nosotros como desarrolladores.

5. COMO CREAR DIALOGOS

<http://www.youtube.com/watch?v=3F46d9eh1U0>



```
*C:\java\misdatos.java - Notepad++
Archivo  Editor  Buscar  Vista  Codificación  Lenguaje  Configuración  Macro  Ejecutar  TextFX  Plugins  Ventana  ?

4  import javax.swing.*;
5
6  public class misdatos extends JApplet
7  {
8
9      public void init()
10     {
11         ImageIcon foto = new ImageIcon("yoicono.png");
12         JLabel lbl = new JLabel(foto);
13
14         Container pantalla = this.getContentPane();
15         pantalla.setLayout(null);
16
17         lbl.setBounds(0, 0, 100, 100);
18     }
19 }
```

Imagen relacionada del video de youtube

OBJETIVOS GENERAL

Comprender que gracias a la interacción de ventanas es posible la construcción una solución que facilita el acceso y la comprensión del usuario de los datos y su importancia para la gestión de los procesos.

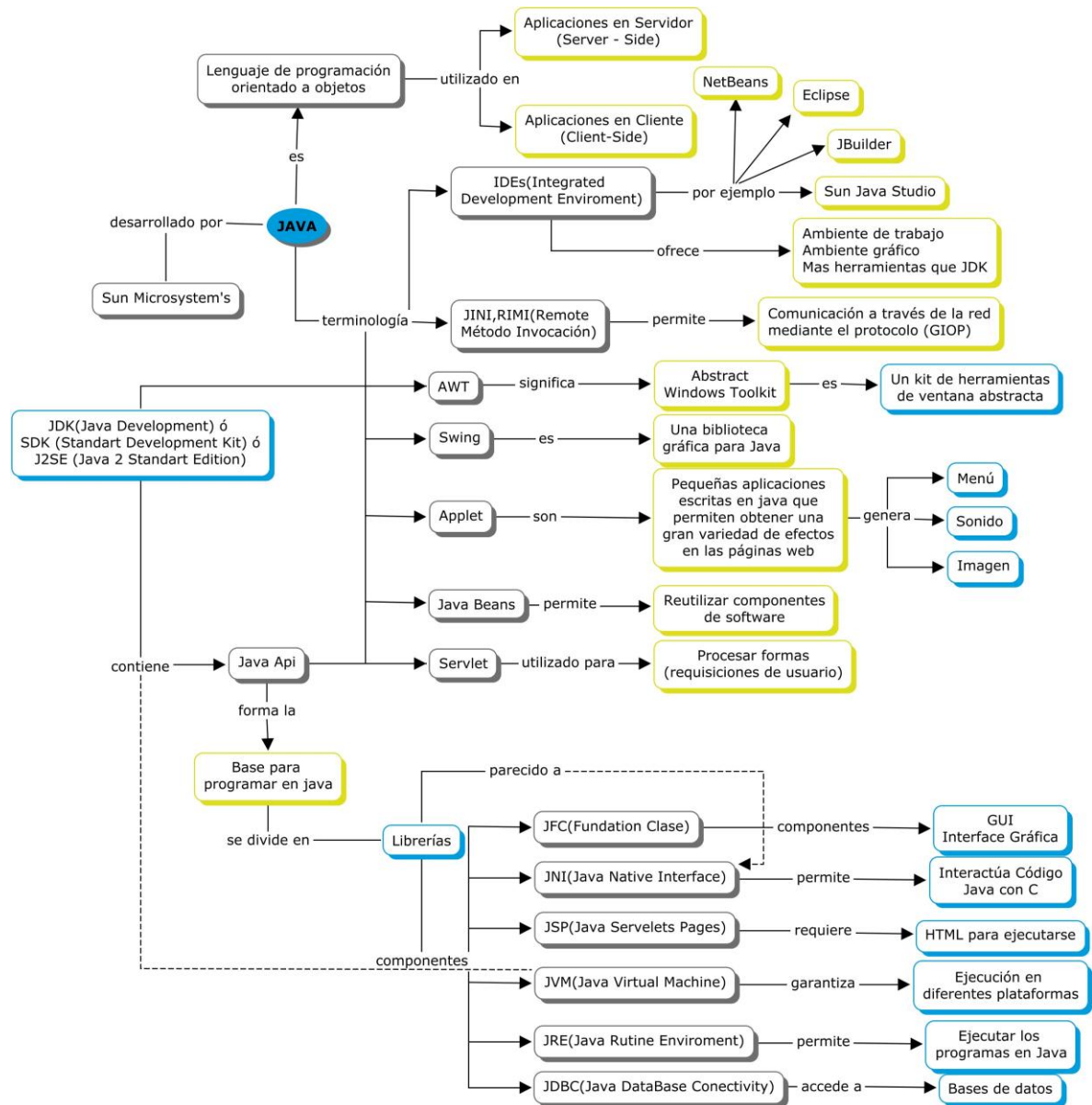
OBJETIVOS OBJETIVOS

- ❑ Determinar la importancia de los applets en el desarrollo del lenguaje de programación .
- ❑ Reconocer que muchos de los componentes del lenguaje tienen sus propios intérpretes.
- ❑ Conocer las distintas etapas que se recorren con un applet.

5.1. Prueba Inicial

1. Desarrolle un Applet Java que permita la impresión de un párrafo de texto. Utilice un String [] para almacenar las líneas del párrafo, de manera que pueda utilizar una estructura de repetición para (for) o while para descargar en el método paint el contenido del String [].utilizando la primitiva de trazado drawString que acabamos de ver en el ejemplo anterior.
2. Desarrolle un Applet Java que imprima las operaciones básicas (suma, resta, producto y cociente) de dos valores numéricos y que muestre la raíz cuadrada, el Logaritmo en base 10 y la potenciación de a^b y b^a , donde a y b, son los números de las operaciones solicitadas en esta aplicación. Cree un método para cada operación de cálculo aritmético e invóquelo en la definición del método paint para este Applet solicitado.

5.2. Relación de Conceptos



Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Mapa conceptual Java	http://cmaps.cmappers.net/rid=1HPBF0Q6X-1NZ3779-DNC/Mapa%20conceptual%20java.cmap	IHMC Cmaps Tools	29/10/2011

5.3. Que es un Applet

¿Qué es un Applet?

Al iniciarse el proceso de desarrollo de aplicaciones con Java, se caracterizaron al principios dos tipos de programas básicos (Los programas de consola que permitían visualizar el funcionamiento de las clases, pero que ofrecían una interfaz de usuario prácticamente inexistente, y donde los procesos de entrada y salida de datos se realizaban vía esa consola a través de métodos como `System.in.read ()` y `System.out.println ()`).

Por otro lado, y consciente de la necesidad de poseer un espacio gráfico que permitiese desarrollar aplicaciones con mayor nivel de interactividad para el usuario, Sun creó el concepto de Applet como mecanismo para facilitar la construcción de ese tipo de soluciones. Dada la necesidad de garantizar la portabilidad entre sistemas operativos (principio funcional de la plataforma Java), el objetivo de los Applets es permitir que las aplicaciones corran dentro de un contenedor externo (por lo general una página Web escrita en el lenguaje HTML). A diferencia de un programa, un Applet no puede ejecutarse de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un Applet normalmente lleva a cabo una función muy específica que carece de uso independiente. Un Java Applet es un código JAVA que carece de un método `main`, por eso se utiliza principalmente para el trabajo de páginas web, ya que es un pequeño programa que es utilizado en una página HTML y representado por una pequeña pantalla gráfica dentro de ésta. Por otra parte, la diferencia entre una aplicación JAVA y un Applet radica en cómo se ejecutan. Para cargar una aplicación JAVA se utiliza el intérprete de JAVA (pcGRASP de Auburn University, Visual J++ de Microsoft, Forte de Sun de Visual Café, Eclipse IDE, etc.). En cambio, un Applet se puede cargar y ejecutar desde cualquier explorador que soporte JAVA (Netscape, Internet Explorer, Mozilla Firefox, Opera, etc.).

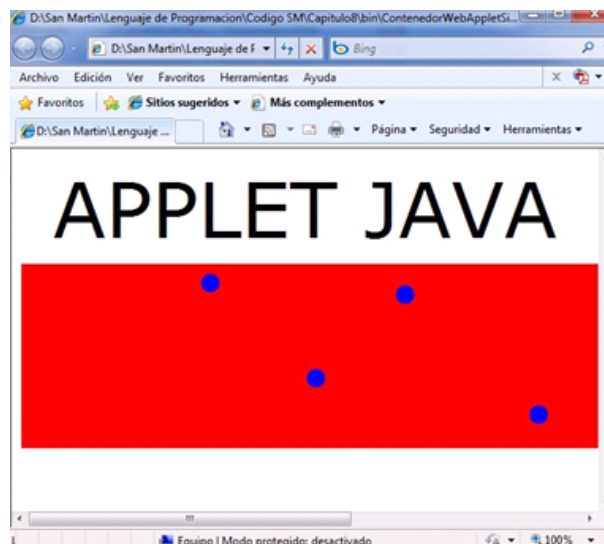


Imagen 5.1

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ejecución de un Applet Java dentro de un navegador Web HTML.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

EJERCICIOS DE AUTOEVALUACIÓN

A continuación observemos la estructura básica de un Applet Java:

```

package Appletssimples;
import java.Applet.Applet;
import java.awt.*;
public class AppletImpresion extends Applet
{
private static final long serialVersionUID = 9L;
private String cadenaImprimir;
private Font fuente;
private Color colorFuente;

private int n1;
private int n2;
private int resultado;

public AppletImpresion() throws HeadlessException
{
// TODO Auto-generated constructor stub
} public void init()

```

```
{ fuente = new Font("monospaced",Font.BOLD,26);
cadenalmpimir = "Ejemplo de un Applet Java";
colorFuente = Color.GREEN;
//Definimos el fondo del canvas setBackground(Color.RED);
//Definimos su tamaño setSize(800,600);
} public void paint (Graphics g)
{
//El objeto graphics establece un patron para pintar
//sobre el canvas del Applet
//Mejoremos la presentacion del Applet g.setFont(fuente);
g.setColor(colorFuente);
//Pintamos g.drawString(cadenalmpimir,25,40);

//Ejecuto los metodos de logica del Applet
//Damos valores a los atributos this.n1 = 23;
this.n2 = 1004;

//Obtenemos el resultado
this.resultado = sumarValoresApplet(this.n1,this.n2);
//Definimos el color
//Imprimimos el resultado de la suma en el canvas g.setColor(Color.BLUE);
g.drawString(String.valueOf(this.resultado),25,200);
}
int resultado = 0; resultado = n1 + n2; return (resultado);
}
/*****/ }
```

Hemos observado hasta el momento, cómo es posible construir una aplicación Java basada en el uso del entorno gráfico de la API Swing, que permite definir los contenedores principales, intermedios y componentes atómicos que estructuran la interfaz y consolidan las pantallas que el usuario finalmente va a visualizar en el momento del despliegue de la solución. Por otro lado, comenzamos a visualizar alguno de los elementos que permiten el desarrollo de los eventos y que profundizaremos con más detalle en este fascículo para ver cómo es posible, por medio de la escritura de oyentes de eventos, controlar la lógica de la aplicación y manejar apropiadamente la interacción entre el usuario y la aplicación.

Cuando desarrollamos una aplicación, necesitamos en muchas ocasiones, tener la capacidad de poder ingresar datos de forma rápida, sin tener que proceder a la implementación de un formulario que únicamente imple- mente dos botones y que a partir de la definición del oyente

escuchador de sus botones permita obtener, por ejemplo, un sí o un no como respuesta para la definición del valor de bifurcación de una decisión lógica.

Para este tipo de propósitos, la API Swing nos ofrece la posibilidad de utilizar una serie de ventanas prefabricadas y cuadros de diálogo que aceleran el proceso de desarrollo y cuyo uso se convierte en un ejercicio invocación de este tipo de objetos, más que de construcción y parametrización de los mismos. Ya nos hemos acercado a estos cuadros de diálogo en aplicaciones previas, gracias al uso del componente y objeto `JOptionPane`, el cual exploraremos con más detalle en los siguientes ejemplos.

Un cuadro de diálogo se convierte en una herramienta muy útil para propósitos tales como:

- a) Capturar información simple (cadenas de texto, números o valores alfanuméricos)
- b) Desplegar resultados, informar sobre potenciales mensajes de error (como hemos visto en la gestión de Excepciones tratada en otro fascículo), y/o
- c) Facilitar la captura de datos establecidos en el sistema, tales como la ruta o ubicación de un archivo o carpeta, o de apoyar el proceso de captura de un valor lógico booleano (como lo que podría ser una respuesta tipo Sí/No, formulada en algún momento o punto específico del tiempo de la ejecución de una aplicación).

Para poder hacer esto, se hace necesario utilizar objetos específicos de la API Swing que faciliten el despliegue de dichos controles preestablecidos y acá es donde nuevamente, empleamos los controles de Swing y de su objeto `JOptionPane` para facilitar el desarrollo de estos cuadros de diálogo automáticos.

El siguiente ejemplo, ilustra con más precisión, el uso de los diálogos automáticos:

```
package interfazgrafica;
import java.awt.event.*;
import javax.swing.*;

public class VentanaMensaje extends JFrame implements ActionListener
{
    private static final long serialVersionUID = 9L;
    public VentanaMensaje()
    {
        //Creamos el botón para desplegar la ventana JButton boton = new JButton("Mostrar la Ventana");
        getContentPane().add( boton,"Center" ); setSize(400,100);
        //Agregamos el manejador de evento para el botón
        boton.addActionListener( this );
    }
}
```

```

    }
    public void actionPerformed( ActionEvent evt )
    {
        JOptionPane.showMessageDialog(null,
        "Mensaje generado en un diálogo de respuesta","Ejemplo de
        JOptionPane",JOptionPane.WARNING_MESSAGE );
    }
    /*****/
    //Programa principal
    public static void main( String [] args)
    {
        try
        {
            new VentanaMensaje().setVisible( true );
        }

        catch (Exception errorMain)
        {
            errorMain.toString();
            errorMain.printStackTrace();
        }
        /*****/
        }//Fin del main
        /*****/
        }//Fin de la clase
    
```



Figura 5.2

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Despliegue de la ventana contenedora principal de una aplicación bajo Swing y de un componente de respuesta al evento, de tipo JOptionPane para mostrar el procesamiento generado por la interacción del usuario con el botón "Mostrar Ventana".	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Es de observar, que la aplicación implementa dos objetos muy específicos: Uno que en este caso es el definido por la clase Java VentanaMensaje, que vemos que extiende o hereda de la clase JFrame (para constituirse en un marco gráfico o ventana principal) y que a su vez, se encarga de implementar los métodos de la interfaz ActionListener, lo que permite que dentro del código de la misma clase sea posible definir los manejadores del oyente de eventos necesarios para modelar la respuesta del Sistema, ante la pulsación de un botón, en este caso, el botón agregado al panel principal del frame establecido por el objeto VentanaMensaje.

Observemos ahora con más detalle la estructuración en el constructor de la forma como se construye la aplicación:

```
//Creamos el botón para desplegar la ventana
JButton boton = new JButton("Mostrar la Ventana");
getContentPane().add( boton,"Center" );
setSize(400,100);
//Agregamos el manejador de evento para el botón
boton.addActionListener( this );
```

Vemos como una vez el botón ha sido instanciado y agregado al ContentPane principal, se invoca el manejador u oyente (Listener) del evento de pulsación del botón. Vemos igualmente que como parámetro se le pasa el operador de referenciación this. Este operador que ya lo habíamos explorado previamente y que sabemos que se emplea para referenciar variables de la propia clase que está siendo implementada, le indica en este caso al método del escuchador, que debe estar pendiente de los eventos ocurridos sobre el botón definido para la clase implícita actual (en este caso VentanaMensaje.java). Con ello, es posible, sin tener que hacer la implementación del oyente fuera de la clase, y dado que la clase VentanaMensaje.java está con sus métodos implementando la interfaz ActionListener, definir, dentro de su paquete de métodos, la creación y especificación funcional de evento actionPerformed, que como vimos anteriormente, recibe como parámetro un argumento de tipo Event que le informa al administrador de sucesos de la máquina virtual, que el botón implementado por la clase ha sido pulsado en determinado momento.

Dicho evento debe establecerse con el objetivo de poder controlar que botón ha sido presionado. Para ello, cada botón debe agregar su propio manejador de eventos a través del método

actionPerformed. Y, cómo observamos en la implementación, al pulsar este botón se dispara la generación de un cuadro de diálogo de advertencia `JOptionPane.showMessageDialog` (que ya conocemos) y que perfectamente nos puede informar de la eventualidad suscitada a raíz de la pulsación del botón.

```
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.event.*;
import javax.swing.*;

public class VentanaMensaje extends JFrame implements ActionListener
{
    private static final long serialVersionUID = 20L;
    public VentanaMensaje(){
        //Definimos el layout para poder colocar los dos botones setLayout(new GridLayout(2,1));
        //Creamos el botón para desplegar la ventana JButton boton = new JButton("Mostrar la Ventana");
        JButton boton2 = new JButton("Mostrar la Ventana 2"); getContentPane().add( boton);
        getContentPane().add( boton2);
        setSize(400,100);
        //Agregamos el manejador de evento para el botón 1 y 2
        boton.addActionListener( this ); boton2.addActionListener(this);
        //Establecemos la definición de los comandos para cada
        boton2.setActionCommand("boton2");
    } public void actionPerformed( ActionEvent evt )
    {
        if ("boton1".equals(evt.getActionCommand()))
        {
            JOptionPane.showMessageDialog(null,
            "Mensaje generado en un diálogo de respuesta para el botón 1", "Ejemplo de
            JOptionPane", JOptionPane.WARNING_MESSAGE );
        }
        else
        if ("boton2".equals(evt.getActionCommand()))
        {
            //El sistema detectó la pulsación del botón 2 de la //GUI JOptionPane.showMessageDialog(null,
            "Mensaje generado en un diálogo de respuesta para el botón 2", "Ejemplo de
            JOptionPane",JOptionPane.WARNING_MESSAGE );
        }
    } public static void main( String [] args)
    { try
```

```

{ new VentanaMensaje().setVisible( true ); }
catch (Exception errorMain)
{ errorMain.toString();errorMain.printStackTrace();
}
}
} //Fin del main
} //Fin de la clase
    
```



Imagen 5.3

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Administración del manejador de evento para cada botón en la GUI de la aplicación	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Contenedores Intermedios

Los paneles son los contenedores de propósito general más frecuentemente utilizados. Implementados con la clase JPanel, los paneles no añaden casi ninguna funcionalidad más allá de las que tienen los objetos JComponent (Botones, marcas de verificación, cuadros de texto, etc). Normalmente se usan para agrupar componentes, porque los componentes están relacionados o sólo porque agruparlos hace que la distribución sea más sencilla. Un panel puede usar cualquier controlador de distribución, y se les puede dotar de bordes fácilmente.

Otros cuatro contenedores Swing proporcionan más funcionalidad. Un scroll pane proporciona barras de desplazamiento alrededor de un sólo componente. Un split pane permite al usuario personalizar la cantidad relativa de espacio dedicada a cada uno de dos componentes. Un tabbed

pane muestra sólo un componente a la vez, permitiendo fácilmente cambiar entre componentes. Un tool bar contiene un grupo de componentes (normalmente botones) en una fila o columna, y opcionalmente permite al usuario arrastrar la barra de herramientas a diferentes localizaciones.

Clase Java para implementar el uso de un JScrollPane. En este ejemplo, construimos un TextArea que se inserta en el interior del JScrollPane. Si lo observamos desde la perspectiva funcional, un editor de texto puede implementarse de esta forma, pues a medida que ingresamos texto y este se hace cada vez más grande, se hace necesario poder desplazarnos alrededor de todo el contenido almacenado en el control del formulario.

```

// mi programa
import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PanelScrolling
{
    private static JFrame ventana;
    private static JScrollPane panel1;
    private static JTextArea jt1;

    public PanelScrolling()
    {
        //declaracion, creacion e inicializacion de componentes, objetos y variables
        ventana = new JFrame();
        panel1 = new JScrollPane();
        jt1 = new JTextArea();
        // TODO Auto-generated constructor stub
        ventana.setSize(600,600);
        ventana.setTitle("mi programa");
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.getContentPane().add(panel1, BorderLayout.CENTER);
        panel1.setViewportView(jt1);
        ventana.setVisible(true);
    }

    // parte principal de programa
    public static void main(String[] args)
    {
        // area de definicion de propiedades de el objeto
        try
        {
            new PanelScrolling();
        }
    }
}
    
```

Imagen 5.4

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ejemplo de una ventana con JScrollPane	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Código para implementar un editor de texto sencillo utilizando el componente JScrollPane:

```

import java.lang.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class PanelScrolling
{
    private static JFrame ventana;
    private static JScrollPane panel1;
    private static JTextArea jt1;
    
```



```
public PanelScrolling()
{
    ventana= new JFrame();
    panel1 = new JScrollPane();
    jt1 = new JTextArea();
    // TODO Auto-generated constructor stub ventana.setSize(800,600);
    ventana.setTitle("mi programa");
    ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    ventana.getContentPane().add(panel1,BorderLayout.CENTER);
    panel1.setViewportViewView(jt1); ventana.setVisible(true);
}
public static void main(String[] args)
{
    // area de definicion de propiedades de el objeto try
    {
        new PanelScrolling();
    }

    catch (Exception errorMain)
    {
        errorMain.toString();
        errorMain.printStackTrace();
    }
}
}
```

Ejemplos de Manejo de Eventos

A continuación, visualizaremos un ejemplo concreto de los conceptos que hemos visto, reflejado en una aplicación Java capaz de implementar una calculadora sencilla. En dicho ejemplo, encontraremos el tema de distribución de los componentes, uso del layout, establecimiento de manejadores de eventos para los botones y la ejecución de procesos y métodos dependiendo de las acciones ejecutadas por el usuario contra la GUI.

```
package interfazgrafica;
//Importaciones
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import java.awt.Dimension;
```

```
import java.awt.event.*;
import javax.swing.JLabel;
import java.awt.Rectangle;
import javax.swing.JTextField;
import javax.swing.JButton;

public class AplicacionSimpleSwingConControles

{
/*****/
//Atributos de la clase
//CONTROLES DEL FORMULARIO
private JFrame formulario = null;
private JPanel ventanaPrincipal = null;
    private JLabel etiquetaNumero1 = null;
private JLabel etiquetaNumero2 = null;
private JLabel etiquetaRespuesta = null;
private JTextField campoNumero1 = null;
JTextField campoNumero2 = null;
private JTextField campoResultado = null;
private JButton botonSumar = null;
private JButton botonRestar = null;
private JButton botonMultiplicar = null;
private JButton botonDividir = null;
public AplicacionSimpleSwingConControles()
{
//Creamos un nuevo frame
JFrame ventana = getFormulario();
ventana.addWindowListener(new WindowAdapter()
{
public void windowClosing(WindowEvent e)
{
System.exit
}
});
ventana.setSize(600,200);
//La mostramos ventana.setVisible(true);
} private JFrame getFormulario()
{ if (formulario == null)
{ formulario = new JFrame();
```

```
formulario.setTitle("VENTANA BÁSICA HECHA EN SWING");
formulario.setSize(new Dimension(620, 232));
formulario.setContentPane(getVentanaPrincipal());
}
return formulario;
}
private JPanel getVentanaPrincipal()
{
    if (ventanaPrincipal == null)
    {
        etiquetaRespuesta = new JLabel();
        etiquetaRespuesta.setBounds(new Rectangle(15, 107, 76, 28));
        etiquetaRespuesta.setText("Respuesta:");
        etiquetaNumero2 = new JLabel();
        etiquetaNumero2.setBounds(new Rectangle(15, 59, 75, 29));
        etiquetaNumero2.setText("Número 2:");
        etiquetaNumero1 = new JLabel();
        etiquetaNumero1.setBounds(new Rectangle(15, 16, 75, 27));
        etiquetaNumero1.setText("Número 1:");
        ventanaPrincipal = new JPanel();
        ventanaPrincipal.setLayout(null);
        ventanaPrincipal.add(etiquetaNumero1, null);
        ventanaPrincipal.add(etiquetaNumero2, null);
        ventanaPrincipal.add(etiquetaRespuesta, null);
        ventanaPrincipal.add(getCampoNumero1(), null);
        ventanaPrincipal.add(getCampoNumero2(), null);
        ventanaPrincipal.add(getCampoResultado(), null);
        ventanaPrincipal.add(getBotonSumar(), null);
        ventanaPrincipal.add(getBotonRestar(), null);
        ventanaPrincipal.add(getBotonMultiplicar(), null);
        ventanaPrincipal.add(getBotonDividir(), null);
    }
    return ventanaPrincipal;
}
private JTextField getCampoNumero1() {
    if (campoNumero1 == null) {
        campoNumero1 = new JTextField();
        campoNumero1.setBounds(
            new Rectangle(108, 16, 164, 28));
    }
    return campoNumero1;
}
private JTextField getCampoNumero2() {
    if (campoNumero2 == null) {
        campoNumero2 = new JTextField();
        campoNumero2.setBounds(new Rectangle(108, 61, 164, 28));
    }
    return campoNumero2;
}
private JTextField getCampoResultado() {
    if (campoResultado == null) {
        campoResultado = new JTextField();
        campoResultado.setBounds(new Rectangle(108, 111, 163, 28));
    }
    return campoResultado;
}
```

```
}  
private JButton getBotonSumar() {  
    if (botonSumar == null) {  
        botonSumar = new JButton();  
        botonSumar.setBounds(new Rectangle(303, 15, 102, 41));  
        botonSumar.setBounds(new Rectangle(307, 14, 102, 41));  
        botonSumar.setText("Sumar");  
        botonSumar.addActionListener(new java.awt.event.ActionListener() {  
            public void actionPerformed(java.awt.event.ActionEvent e)  
            { System.out.println("actionPerformed() botón suma"); // TODO //Auto-generated  
              Event stub actionPerformed()  
              //Capturamos los valores int n1 = 0;  
              int n2 = 0;  
              int suma = 0; n1 = Integer.parseInt(campoNumero1.getText());  
              n2 = Integer.parseInt(campoNumero2.getText());  
              suma = n1 + n2; campoResultado.setText(String.valueOf(suma)); }  
            });  
        return botonSumar;  
    }  
    private JButton getBotonRestar() {  
        if (botonRestar == null) {  
            botonRestar = new JButton();  
            botonRestar.setBounds(new Rectangle(451, 15, 101, 40));  
            botonRestar.setBounds(new Rectangle(459, 14, 101, 40));  
            botonRestar.setText("Restar botonRestar.");  
  
            botonRestar.addActionListener(new java.awt.event.ActionListener() {  
                public void actionPerformed(java.awt.event.ActionEvent e)  
                {  
                    //Ejecutamos la resta  
                    System.out.println("actionPerformed() botón resta");  
                    // TODO Auto-generated Event stub actionPerformed()  
                    //Capturamos los valores  
                    int n1 = 0;  
                    int n2 = 0;  
                    int resta = 0;  
                    //Leemos los textos de los controles y los convertimos  
                    //en números enteros  
  
                    n1 = Integer.parseInt(campoNumero1.getText());
```

```
n2 = Integer.parseInt(campoNumero2.getText());
//Ejecuta mos la operación resta = n1 - n2;
//Actualizamos el control de respuesta
//Actualizamos el control

campoResultado.setText(String.valueOf(resta));
}
});}
return botonRestar;
}

private JButton getBotonMultiplicar() {
if (botonMultiplicar == null) {
botonMultiplicar = new JButton();
botonMultiplicar.setBounds(new Rectangle(302, 95, public static void main(String [] args)
{
/*****/
try
{
new AplicacionSimpleSwingConControles();
}

catch (Exception errorMain)
{
errorMain.toString();
errorMain.printStackTrace();
}
/*****/
} //Fin del main
} //Fin de la clase

botonMultiplicar.setBounds(new Rectangle(307, 95, 101, 42));
botonMultiplicar.setText("Multiplicar");
botonMultiplicar.addActionListener(new java.awt.event.ActionListener()
{
public void actionPerformed(java.awt.event.ActionEvent e)
{
//Ejecutamos la multiplicación
System.out.println("actionPerformed() botón producto");
// TODO Auto-generated Event stub actionPerformed()
//Captura mos los valores int n1 = 0;
```

```

int n2 = 0;
int producto = 0;
//Leemos los textos de los controles y los convertimos
//en números enteros
n1 = Integer.parseInt(campoNumero1.getText());
n2 = Integer.parseInt(campoNumero2.getText());

//Ejecutamos la operación
producto = n1 * n2;

campoResultado.setText(String.valueOf(producto));
}
});
    
```

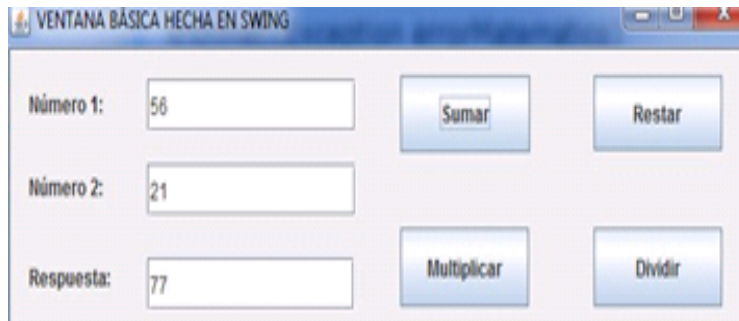


Imagen 5.5

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ventana básica swing	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Pista de aprendizaje:

Tener en cuenta: son herramientas de apoyo, temporales, gráficas que permiten acercarnos a ambientes más potentes.

Tenga presente: no es para especializarnos y quedarnos con esta tecnología, solo es un apoyo de trabajo.

Traer a la memoria: que la mayor de estos controles se utilizará más adelante con mucha de la codificación intacta.

5.4. Applet Viewer

El visualizador de Applets (Appletviewer) es una aplicación que permite ver en funcionamiento Applets, sin necesidad de la utilización de un navegador World-Wide-Web como HotJava, Microsoft Explorer o Netscape. Dicha herramienta permite la depuración de los Applets y visualizar su funcionamiento antes del despliegue de la aplicación final dentro de uno de los navegadores anteriormente citados. El objetivo del Appletviewer es el de facilitar que el desarrollador pueda revisar el Applet antes de su despliegue final en el contenedor Web y probarlo en la funcionalidad de todos sus métodos.

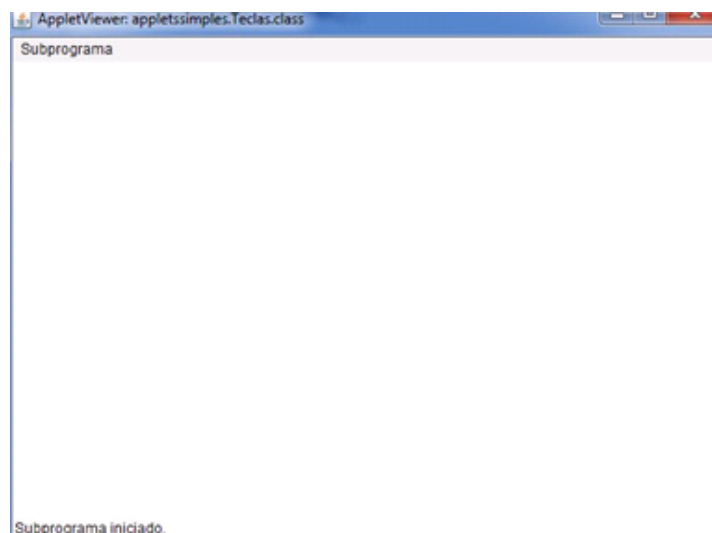


Imagen 5.6

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Herramienta de ejecución y depuración de Applets Java Appletviewer	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Como se observa en la siguiente figura, el Applet que se visualiza despliega dos cadenas de caracteres, la primera, un valor estático que se define en el código de una de las variables de instancia de la clase AppletImpresion.java. La segunda cadena de caracteres que se imprime, muestra el resultado de un cálculo generado por el método sumarValoresApplet que se despliega como la última función implementada dentro del código del método.

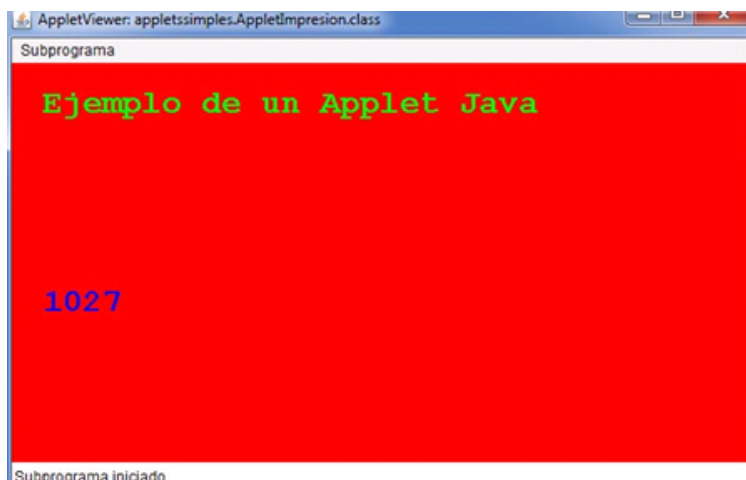


Imagen 5.7

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Vista de un Applet en ejecución sobre el Appletviewer de Java.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

El Appletviewer igualmente permite controlar el despliegue de nuevas instancias del Applet, reiniciarlo cuando este comienza a presentar problemas de ejecución o clonar el objeto de instancia del Applet actual (algo muy útil cuando se tiene que verificar el funcionamiento de varios procesos del usuario en el momento de ejecutar la aplicación). También gracias a él es posible resetear el Applet cuando este deja de funcionar y detener su operación para corregir errores en el él y lanzarlo nuevamente a su despliegue.

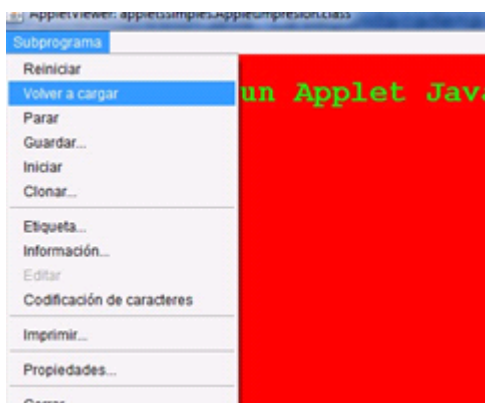


Figura 5.8

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Opciones de administración del Appletviewer para el control de los Applets en ejecución.	Ventana después de implementar el código	Leyder Hernán López	29/10/2011

Pista de aprendizaje:

Tener en cuenta: sin esta herramienta no podrá poner en marcha un applet.

Tenga presente: la compilación es un proceso y la puesta en marcha es otra.

Traer a la memoria: que conocer la sintaxis de ejecución hace más fácil el trabajo cotidiano con esta herramienta.

5.5. Ciclo de vida de un Applet

A diferencia de una aplicación estándar que deja de ejecutarse en el momento que finaliza la última instrucción de su método main, un Applet tiene un ciclo de vida algo diferente. El siguiente diagrama de estados ilustra su funcionamiento básico:

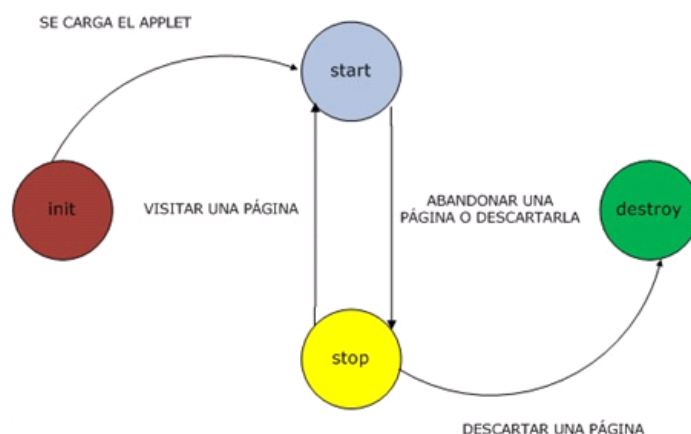


Imagen 5.9

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Ciclo de vida de un Applet Java.	Diseño en Word	Leyder Hernán López	29/10/2011

Cada círculo representa una fase en el ciclo de vida de la Applet. Las flechas representan transiciones y el texto representa la acción que causa la transición. Cada fase está marcada con una invocación a un método del Applet Java:

a) Método `init()`: Como se había mencionado su función es crear la instancia base del Applet generado y ejecutar todas las instrucciones para que el canvas se pueda ejecutar apropiadamente.

b) Método `start()`: Se encarga de inicializar el proceso de interacción del usuario con el Applet. Básicamente su función es la de facilitar la respuesta del Applet ante los eventos de interacción del usuario con el navegador (restaurar el navegador, actualizarlo, minimizarlo, maximizarlo, etc. El método se conjuga con el uso de threads (hilos de ejecución) para permitir que puedan ejecutarse subprocesos dentro del Applet principal.

c) Método `stop()`: Se invoca para detener el Applet de manera tal que ya no se puedan ejecutar más sus funciones. El método `stop` detiene el Applet y hace que los eventos y funciones de actualización ya no puedan ejecutarse más.

d) Método `destroy()`: Se encarga de liberar los recursos que el Applet esté utilizando en dicho momento. Dicho método es invocado por el cargador de clases de la máquina virtual, cuando el Applet detecta que el contenedor que lo anida (la página Web en el navegador HTML se cierra automáticamente).

Esto indica que el Applet, se está cargando:

1. Una instancia de la clase Applet es creada.
2. El Applet es iniciado, mediante su método `init()`.
3. El Applet empieza a ejecutarse, mediante su método `start()`. Cuando el usuario se encuentra con una página Web, que contiene una Applet y salta a otra página, entonces el Applet se detiene invocando a su método `stop()`. Si el usuario retorna a la página donde reside el Applet, ésta vuelve a ejecutarse nuevamente invocando a su método `start()`.
4. Cuando el usuario sale del navegador el Applet tiene un tiempo para finalizar su ejecución y hacer una limpieza final, mediante el método `destroy()`. El garbage collector de la máquina virtual elimina la instancia y libera cualquier recurso y objeto que estuviera siendo utilizado por el Applet en ese momento, liberando memoria para otras aplicaciones.

Etiqueta **APPLET** y paso de parámetros

Con la evolución del lenguaje HTML, como lenguaje para la construcción y definición de documentos hipervinculados para la World Wide Web (telaraña mundial de la información), se hizo necesario agregar mucha más interactividad al desarrollo de las aplicaciones construidas para el entorno mundial de Internet. La aparición de Java como plataforma tecnológica de desarrollo para el despliegue de aplicaciones hizo necesario que el World Wide Web Consortium (W3C) insertara en la especificación del lenguaje una etiqueta (tag) especial para la inclusión de los Applets Java. Su estructura es la siguiente:

`<APPLET code="ClaseAppletJava.class" width="ancho" heigth="alto"></APPLET>`

Donde el parámetro code le indica al navegador que debe cargar en el browser una instancia del JRE local instalado en el equipo de la máquina cliente donde el Applet intenta ser desplegado. Esto hace que el submotor de procesamiento Java configurado en el equipo local suba el class loader (cargador de clases) de la máquina virtual y la copia del Applet que es descargada al cache del equipo cliente, se ejecute localmente en dicho navegador, brindando al usuario la sensación de que la aplicación se ejecuta nativamente en su propia computadora. Los parámetros width (ancho) y heigth (alto), le indican al navegador la cantidad de espacio que debe asignar para que el componente Java se despliegue en la página Web. Dichos valores no deben confundirse con los parámetros del método Java setSize(ancho,alto) visto en el método init del Applet, puesto que dichos valores establecen el tamaño para el canvas, por lo que si los valores de la etiqueta son más pequeños que los del método setSize, se perderá gran parte de la superficie del canvas en el momento del despliegue del Applet sobre el contenedor Web (página HTML).

Dada la capacidad de desarrollar otros componentes, que en lugar de ser beneficiosos, podrían ser nocivos como los troyanos y los spywares, que igualmente son aplicaciones programadas, los navegadores Web más modernos siempre restringen la ejecución de cualquier tipo de código, incluyendo el de Sun Microsystems, por lo que al intentar cargar el Applet, se despliega una advertencia de seguridad, como se muestra a continuación:



Imagen 5.10

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
Advertencia de seguridad desplegada por el intento de cargar una aplicación Java (Applet) dentro de un navegador como el Microsoft Internet Explorer	Ventana después del código implementado	Leyder Hernán López	29/10/2011

En muchas ocasiones, se hace necesario que el Applet, por cuestiones tanto de diseño como de seguridad, lea ciertos valores desde la página Web que lo contiene, con el fin de cargar parámetros de configuración que permitan el despliegue de la aplicación. Para poder hacer esto, se hace necesario incluir unos tags (etiquetas) HTML adicionales que le indiquen al Applet que debe hacer en el método init (como por ejemplo, el tamaño, el ancho, el alto, el color o la lectura de algún valor necesario para la ejecución de un método de lógica de negocio incluido en el código del Applet o en objetos Java que complementen su implementación). Para hacer esto, el código HTML del contenido Web debe contener las siguientes etiquetas:

```
<APPLET code="ClaseAppletJava.class" width="1024" height="768">  
<param name="alto" value="1024"/>  
<param name="ancho" value="768"/>  
<param name="numeropuntos" value="30"/>  
</APPLET>
```

Dichos valores deben establecerse mediante el parámetro de tag (etiqueta) llamado param y debe contener dos valores: Un nombre (name) que indica el nombre del parámetro y un valor (value), que debe establecerse entre comillas y que puede ser una cadena de caracteres, un valor numérico o cualquier dato que deba ser recuperado por el método init en el momento de la instanciación del Applet. Como hemos observado en otros fascículos, dicho valor es tratado en todos los casos como un dato de tipo String, por lo que si se quiere enviar un valor numérico al Applet desde un parámetro, en el código del Applet dicho valor debe ser transformado en su correspondiente valor numérico por el método de turno asociado (parseDouble, parseInt, parseLong, etc.).

En el ejemplo anterior, observamos tres parámetros que definirían el alto y el ancho (para el método setSize) y un parámetro de lógica de negocio que denominaríamos numeropuntos, que sería utilizado por el Applet para procesar uno de sus métodos de operación. Esto significaría, por lo tanto, que en el método init debemos encontrar las siguientes instrucciones Java para procesar los datos direccionados por el contenedor Web HTML:

```
public void init()  
{  
    try  
    {  
        valorAlto = getParameter("alto"); valorAncho = getParameter("ancho");  
        numeroPuntosArranque = getParameter("numeropuntos");  
        if (valorAlto != null && valorAncho != null && numeroPuntosArranque != null)  
        {
```

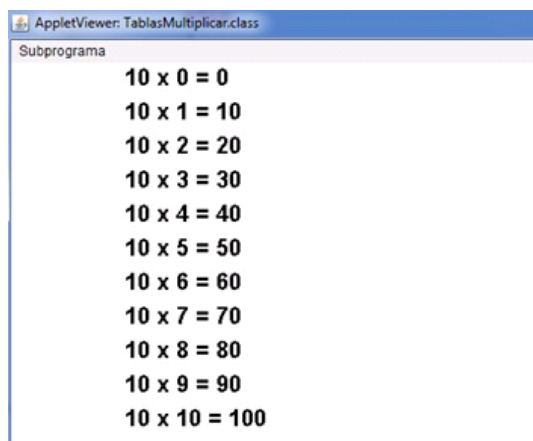
```
alto = Integer.parseInt(valorAlto); ancho = Integer.parseInt(valorAncho); numeroPuntos
=Integer.parseInt(numeroPuntosArranque);
} else
{
alto = 800; ancho = 600; numeroPuntos = 50;
}
//Tamaño del canvas
PUNTOSMAXIMOS = numeroPuntos; puntosx = new int[PUNTOSMAXIMOS]; puntosy = new
int[PUNTOSMAXIMOS]; setSize(alto,ancho);
setBackground(Color.red);
}
catch (Exception errorInit)
{ JOptionPane.showMessageDialog(null,"No se puede cargar el color de fondo. Applet no
inicializable: " + errorInit.toString());
errorInit.printStackTrace(); System.exit(0);
}
}
```

EJERCICIOS DE AUTOEVALUACIÓN

1. Desarrolle un Applet Java que implemente tres botones y un cuadro de texto (Textbox) en la superficie del canvas y que se encargue de implementar, cada uno de ellos, las siguientes funcionalidades:

- ✖ Botón pintar cuadrado. Al pulsarlo debe llamar la primitiva fillRect y pintar de forma aleatoria la cantidad de cuadrados especificados por el valor digitado en el textbox. Los colores de cada cuadrado deben ser igualmente aleatorios (Utilice la función random del paquete Math para generar los valores de las coordenadas y parámetros solicitados por el método de pintado de paralelogramos del objeto Graphics).
- ✖ Botón Borrar. Al pulsarlo, todas las figuras que estén pintadas en la superficie del Applet, deben desaparecer automáticamente. Tome como base la implementación de EditorPoligonos.java para realizar dicha implementación.
- ✖ Botón Pintar Línea. Utilizando la base del Applet puntos locos, defina que al pulsar este botón, el canvas debe mostrar una línea recta trazada entre dos puntos que previamente hayan sido seleccionados en la superficie del canvas e indicados debidamente sobre dicha superficie, a través del uso de la primitiva drawFillOval, tal como actualmente lo hace el Applet de los puntos locos.

2. Desarrolle un Applet Java que permita la impresión de las tablas de multiplicar de un valor previamente establecido en un parámetro del código HTML que debe contener el valor sobre el cual quiere hacerse la tabla de multiplicar. El Applet debe correr, tal como lo muestra la siguiente figura: El parámetro en la página HTML que contenga al Applet debe ser:



```
<param name="valorlimite" value="10">
```

Imagen 5.11

Nombre de la imagen	Dirección Web	Autor	Fecha de Consulta
AppletViewer	Ventana después del código implementado	Yolanda Soler Pellicer	29/10/2011

1. Realice las siguientes actividades:

- ❏ Desarrolle un Applet Java que permita ilustrar la fecha actual del computador, tal como lo hace el reloj del Sistema Operativo. Cuando el Applet se ejecute, debe mostrar la fecha en el siguiente formato:
- ❏ Hoy es Lunes, 31 de Octubre de 2011 y son las 6:30:00 P.M. Construya un Applet Java que utilizando el evento de manejo del teclado permita capturar un valor numérico. Una vez el usuario haya hecho click con el botón izquierdo del ratón, el programa debe mostrar la impresión en el canvas de la suma del número con 100, la resta del número con 50, la multiplicación del producto con 20 y la división del número con 2. Si el número ingresado es 0, en lugar de aparecer un error de división, debe aparecer el mensaje “ERROR – DIVISIÓN POR CERO”. Utilice una excepción aritmética como vimos en fascículos anteriores para controlar el error.

Pista de aprendizaje:

Tener en cuenta: el ciclo de vida siempre será el mismo desde el inicio hasta el final de la ejecución.

Tenga presente: es un proceso poco apreciado puede ser de importancia si queremos profundizar en los recursos que consume el producto.

Traer a la memoria: estos procesos se dan en cada aplicativo que se desarrolle.

6. PISTAS DE APRENDIZAJE

Tener en cuenta: Un “blueprint” es un término usualmente utilizado para hacer alusión a un plan detallado para el diseño de un sistema.

Tener presente: En la Programación Orientada a Objetos, debe recordar que la esencia de la programación está en la algorítmica, es decir, en la ciencia de diseñar estructuras de análisis lógico, que fundamentadas en los condicionales, los ciclos, las asignaciones y la definición de variables, permitan resolver cualquier problema de procesamiento de información.

Traer a la memoria: UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).

Tener presente: Un Pseudocódigo, es una serie de palabras léxicas y gramaticales referidas a los lenguajes de programación, pero sin llegar a la rigidez de la sintaxis de estos ni a la fluidez del lenguaje coloquial.

Tener en cuenta: La plataforma Java suministra cinco controladores de disposición comúnmente utilizados: BorderLayout, BoxLayout, FlowLayout, GridBagLayout, y GridLayout. Estos controladores de distribución están diseñados para mostrar múltiples componentes a la vez.

Traer a la memoria: Los paneles son los contenedores de propósito general más frecuentemente utilizados. Implementados con la clase JPanel, los paneles no añaden casi ninguna funcionalidad más allá de las que tienen los objetos JComponent (Botones, marcas de verificación, cuadros de texto, etc).

Tener presente: Cuando el AWT llama a un método oyente de evento, este método se ejecuta en el thread de eventos del AWT. Como todos los demás manejos de eventos y los métodos de dibujo se ejecutan en el mismo thread (proceso de ejecución), un lento método de manejo de eventos puede hacer que el programa parezca no responder y puede ralentizar el propio dibujado.

Tener en cuenta: Un Applet no puede ejecutarse de manera independiente, ofrece información gráfica y a veces interactúa con el usuario, típicamente carece de sesión y tiene privilegios de seguridad restringidos. Un Applet normalmente lleva a cabo una función muy específica que carece de uso independiente.

7. GLOSARIO

Abstract Windowing Toolkit (AWT).- Biblioteca de módulos para representar interfaces gráficas provisto por Sun en la API de Java.

Ámbito.- Parte de un programa en el que es válida una referencia a una variable.

Análisis.- Proceso de conocer los requerimientos de software que tienen el cliente y el usuario final.

API.- Application Programming Interface.

Aplicación.- Programa informático, que se ejecuta sin necesidad de otro programa

Applet.- Programa informático que se ejecuta necesitando de otro programa, normalmente un navegador.

Application Programming Interface (API).- Conjunto de paquetes y clases Java, incluidos en el JDK que utilizan los programadores Java para realizar sus aplicaciones.

Argumentos.- Parámetros.

Array.- Vector.

ASCII.- American Standard Code for Information Interchange.

AWT.- Abstract Windowing Toolkit.

BDK.- Beans Developer Kit.

Beans Developer Kit (BDK).- Conjunto de herramientas para desarrollar JavaBeans.

Bit.- Unidad mínima de información digital que puede tomar los valores lógicos de 0 o de 1.

Bloque.- Código localizado entre corchetes.

Boolean.- Tipo de datos bi-estado, que puede tomar valor de cierto (true) o falso (false).

Byte.- Secuencia de 8 bits.

Cadena.- Secuencia de caracteres.

Carácter.- Símbolo que representa información, o la codificación en una computadora. Normalmente letras de alfabeto, números o signos ASCII.

Cargador de clases.- Parte del JRE de Java responsable de encontrar archivos de clase y cargarlos en la máquina virtual Java.

Casting.- Moldeado.

CGI.- Common Gateway Interfaz.

Clase.- Unidad fundamental de programación en Java, que sirve como plantilla para la creación de objetos. Una clase define datos y métodos y es la unidad de organización básica de un programa Java.

Common Gateway Interfaz (CGI).- Es un lenguaje de programación que permite dotar a las páginas Web de interactividad, construyendo una página Web correspondiente a un enlace de hipertexto en el mismo momento en que se hace "clic" sobre el enlace. Los *script cgi* pueden estar escritos en cualquier lenguaje de programación.

Common Object Request Broker Architecture (CORBA).- Estándar para la conexión entre objetos distribuidos, aunque esté codificados en distintos lenguajes.

Compilador.- Programa de software que traduce código fuente en un lenguaje de programación legible por una persona a código máquina interpretable por un ordenador.

Constante.- Valor utilizado en un programa de computadoras con la garantía de no cambiar en tiempo de ejecución. La garantía es a menudo reforzada por el compilador. En Java las constantes se declaran como `static final`.

Constructor.- Método que tiene el mismo nombre que la clase que inicia. Toma cero o más parámetros y proporciona unos datos u operaciones iniciales dentro de una clase, que no se pueden expresar como una simple asignación.

Contenedor.- En diseño de interfaces de usuario, es un objeto que contiene los componentes (como botones, barras de deslizamiento y campos de texto).

Conversión de tipos de datos.- Modificación de una expresión de un tipo de datos a otro.

CORBA.- Common Object Request Broker Architecture.

Entero.- Un número entero, sin parte decimal, positivo o negativo.

Estructura de datos.- Una construcción de software (en memoria o en disco duro) que contiene datos y las relaciones lógicas entre ellos.

Evento.- Un mensaje que significa un incidente importante, normalmente desde fuera del entorno de software.

Excepción.- Un evento que ocurre durante la ejecución de un programa que interrumpe el flujo normal de las instrucciones.

Flujo.- Stream.

Graphical User Interface (GUI).- Interfaz gráfica de usuario.

Hardware.- El aspecto físico de un sistema de computadora, como el procesador, disco duro e impresora.

Herencia múltiple.- La práctica (permitida en lenguajes como C++ pero no en Java) de derivar una clase de más de una clase base.

Herencia.- Mecanismo encargado de relacionar clases entre sí de una manera jerárquica. En Java, sólo existe herencia simple.

Hilo.- Thread.

HTML (HyperText Markup Language).- Lenguaje que se utiliza para crear páginas Web. Los programas de navegación de la Web muestran estas páginas de acuerdo con un esquema de representación definido por el programa de navegación.

IDE.- Integral Development Environment.

IDL.- Java Interface Definition Language.

Ingeniería del software.- Rama de la ingeniería concerniente con el análisis, diseño, implementación, prueba, y mantenimiento de programas de computadoras.

Instancia.- Objeto de software construido desde una clase. Por ejemplo, puede tener una clase avión, pero una flota de quince instancias de avión.

Integral Development Enviroment (IDE).- Una herramienta de desarrollo visual en la que un programa puede ser construido, ejecutado y depurado.

Interbloqueo.- Condición en la que dos o más entidades de software se bloquean mutuamente, cada una esperando los recursos que está utilizando la otra.

Interface Definition Language (IDL).- Herramienta mediante la cual los objetos pueden invocar métodos de otros objetos que se encuentren en máquinas remotas, mediante CORBA.

Interfaz gráfica de usuario (GUI).- Una interfaz entre la máquina y el hombre como el Windows de Microsoft, el Mac OS, o el Sistema X Windows, que depende de pantallas de alta resolución, un recurso gráfico de puntero como un ratón y una colección de controles en pantalla (denominados Widgets) que el usuario puede manejar directamente.

Interfaz.- Mecanismo Java para decirle al compilador que un conjunto de métodos serán definidos en futuras clases. (Esas clases estarán definidas para implementar la interfaz).

Java 2D.- Paquete que permite a los desarrolladores incorporar texto, imágenes y gráficos en dos dimensiones de gran calidad.

Java 3D.- Conjunto de clases para crear aplicaciones y applets con elementos en tres dimensiones. Es parte del JMF.

Java DataBase Connectivity (JDBC).- Lenguaje estándar de Java para interactuar con bases de datos, similar al SQL. Es independiente no sólo de la plataforma sino también de la base de datos con que interactúe. Desde la versión 1.2 del JDK se permite interactuar con ODBC.

Java Developer Connection (JDC).- Conexión de desarrollo en la que se publican las versiones beta de las bibliotecas de Java que se están desarrollando.

Java Foundation Classes (JFC).- Conjunto de componentes y características para construir programas con interfaces gráficas.

Java Media Framework (JMF).- Protocolo de transmisión de datos para la reproducción multimedia (vídeo y sonido).

Java Native Invocation (JNI).- Capacidad de Java para ejecutar código nativo, es decir, código compilado al lenguaje máquina de un determinado ordenador. Permite a la Máquina Virtual Java (JVM) interactuar con programas o bibliotecas escritos en otros lenguajes (C/C++, ensamblador...). No se puede utilizar en applets, pues viola las directrices de seguridad.

Java Runtime Environment (JRE).- Software suministrado por Sun que permite a los programas de Java ejecutarse en una máquina de usuario. El JRE incluye la Máquina Virtual Java (JVM).

JRE.- Java Runtime Environment.

JVM.- Java Virtual Machine.

Java Virtual Machine (JVM).- El intérprete de Java que ejecuta los códigos de byte en una plataforma particular.

JavaBeans.- Paquete que permite escribir componentes software Java, que se puedan incorporar gráficamente a otros componentes.

JDBC.- Java DataBase Connectivity.

JDC.- Java Developer Connection.

JFC.- Java Foundation Classes.

JMF.- Java Media Framework

JNI.- Java Native Invocation.

JVM.- Java Virtual Machine.

Llamada por referencia.- Una forma de transferir parámetros a una subrutina en la que se pasa un puntero o referencia a un elemento, de esta forma, la subrutina puede leer y cambiar el valor del elemento referenciado.

Llamada por valor.- Una forma de transferir parámetros a una subrutina en la que se pasa la copia del elemento; las modificaciones de la copia no afectan al elemento original.

Método.- Conjunto de sentencias que operan sobre los datos de la clase para manipular su estado.

Miniaplicación.- Applet.

Modelo.- En diseño orientado a objetos, una representación del mundo real en unas abstracciones de software denominadas clases y la relación entre ellas.

Moldeado.- Suplantación del tipo de un objeto o variable por otro nuevo tipo.

Multiproceso.- En sistemas operativos, la habilidad de efectuar dos o más programas independientes, comúnmente en un procesador solo (a través de Multitarea).

Navegador Web.- Software que permite al usuario conectarse a un servidor de Web utilizando Hypertext Transfer Protocol (HTTP). Microsoft Internet Explorer, Netscape Navigator, HotJava de Sun, son populares navegadores de Web.

Navegador.- Navegador Web.

Null.- Valor de Java que significa vacío.

Object DataBase Connectivity (ODBC). Lenguaje estándar de Microsoft; que utiliza un driver del fabricante de una base de datos, para interactuar con ella, más orientado a C/C++ que a Java.

ODBC.- Object DataBase Connectivity.

Paquete.- Nombre de Java para una biblioteca de clases.

Parámetros formales.- Nombres utilizados dentro de una subrutina por sus parámetros.

Parámetros.- Valores u objetos pasados entre una subrutina y la rutina de llamada.

Plug-in.- Un programa de una plataforma específica diseñado para ser llamado por un navegador

Web. Utilizado con frecuencia para mostrar información que el mismo navegador no puede mostrar.

Poliformismo.- En diseño orientado a objetos, la habilidad de utilizar una clase derivada en lugar de su clase base. Por ejemplo, un programador puede escribir un método expresarse () para la clase

Mamífero. Un Perro, una Vaca y un Gato se derivan de Mamífero, y todos pueden expresarse (), aunque sus voces sean bastantes diferentes.

Proceso.- Instancia de un programa ejecutable. Por ejemplo, si inicia dos copias de un intérprete de Java, tiene dos procesos de la máquina virtual de Java ejecutándose en su computadora.

Seudocódigo.- Documentación de diseño que describe el trabajo de un programa en inglés estructurado (o en otro lenguaje) en lugar de un lenguaje de computadora.

Recolector de basura.- En Java, el mecanismo por el cual se recupera y libera la memoria asociada con objetos no utilizados.

Remote Method Invocation (RMI).- Herramienta que incorpora métodos Java para localizar objetos remotos, comunicarse con ellos e incluso enviar objetos como parámetros de un objeto a otro.

RMI.- Remote Method Invocation.

Secure Sockets Layer (SSL).- Sistema para la creación de conexiones seguras en red.

Servlets.- Módulos que permiten sustituir o utilizar el lenguaje Java en lugar de programas CGI.

Shell.- Intérprete de órdenes de un sistema operativo.

Sistema operativo.- Software responsable de asignar a los usuarios los recursos de sistemas de computadoras (incluyendo procesos). UNIX, Windows, NT y Mac OS, son ejemplos de sistemas operativos.

SQL.- Structured Query Language.

SSL.- Secure Sockets Layer.

Estático.- En diseño orientado a objetos, representa la pertenencia a la clase, en vez de a una instancia. Es un espacio compartido por todas las instancias de una clase.

Stream.- Flujo de datos. Por ejemplo las entradas y salidas de un programa.

String.- Objeto Java estandarizado en el lenguaje, que representa una cadena de caracteres.

Structured Query Language (SQL).- Lenguaje para realizar consultas a Bases de Datos relacionales.

Subclase.- Clase descendiente de otra clase de la que hereda métodos y variables.

Superclase.- Clase de la cual heredan sus métodos y variables otras clases denominadas subclases.

Swing.- Paquete que permite incorporar elementos gráficos en las aplicaciones, de una manera más potente que con el AWT. Aparece en la versión 1.2 del JDK. Es no de los componentes que están incluidos en las Java Foundation Classes, o JFC.

Thread.- Un "proceso ligero" que puede ser arrancado y utilizado más rápidamente que por un fork o spawn. Véase también: fork, spawn y Proceso.

UML.- Unified Modeling Language.

Unicode.- Conjunto de caracteres de 16 bits, en lugar de los 8 que soportaba ASCII. Así se pueden representar la mayor parte de los lenguajes del mundo.

Unified Modeling Language (UML).- Notación estándar de facto utilizada en el análisis y diseño orientado a objetos, basado en el trabajo de Grady Booch, James Rumbaugh, e Ivar Jacobson.

Vector.- Estructura de datos que coloca un tipo de datos en celdas continuas.

Verificador de código de byte.- Rutinas en la máquina virtual de Java, que aseguran que las instrucciones en el archivo de clase no violan ciertas restricciones de seguridad.

8. BIBLIOGRAFÍA

Java Didáctica y Programación, K.Arnold e J. Gosling, Addison-Wesley Primera edición, (en italiano) Marzo de 1997

Manual QUE - Special Edition Using Java, 2nd Edition, versión encontrada en internet (en inglés).
Java2 Todo&Más, J. Jaworski, SAMS Publishing - APOGEO (en inglés)

Javatm 2D Graphics, J. Knudsen, O'REILLY (en inglés)

Ambiente para explorar los micromundos en competición, Pietro Castellucci, Tesina en Informática. (en italiano)

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).

[Eckel, 1997] Bruce Eckel. "Hands -on Java Seminar". President MindView Inc. 1997. 577 páginas. (Tutorial completo en inglés en formato PDF).

[García et al., 1999]. Javier García 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.