PROGRAMACION ORIENTADA A OBJETOS

LUIS EDIN IBARGUEN MOSQUERA

Taller N°. 1

ING. CARLOS ALBERTO LONDOÑO LOAIZA

CORPORACIÓN DE ESTUDIOS

TECNOLÓGICOS

DEL NORTE DEL VALLE

TECNICO PROFESIONAL EN PROGRAMACION DE APLICACIONES INFORMATICAS

CARTAGO

2017

**Taller No. 1 - POO**

1. Definir los conceptos fundamentales involucrados en el proceso de resolución de un problema a través de la computadora.
2. Definir lenguaje, programa, compilador e intérprete.
3. Identificar los antecedentes de la programación orientada a objetos.
4. Definir los conceptos involucrados en la programación orientada a objetos.
5. Identificar las etapas para la resolución de un problema bajo un enfoque orientado a objetos.
6. Identificar los tipos de datos que se emplean en el desarrollo de un programa.
7. Definir los conceptos de variables, expresiones y constantes.
8. Describir el procedimiento para declarar los formatos de las sentencias de asignación y de entrada y salida.
9. ¿Qué es UML y qué proceso se puede realizar con este lenguaje?
10. ¿Qué es JAVA, porque es importante para la industria del desarrollo del software y en qué proyectos se usa?

**DESARROLLO**

1.R/Los pasos para la resolución de un problema son:

- Análisis del problema.

- Diseño del algoritmo.

- Codificación.

- Compilación y ejecución.

- Verificación y depuración.

**ANÁLISIS DEL PROBLEMA:** Como primer paso se debe analizar el problema. Este paso requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y el resultado o solución deseada. Dado que se busca una solución por computadora, se precisan especificaciones detalladas de entrada y salida. Para poder definir un problema es conveniente responder a las siguientes preguntas:

- ¿Qué entradas se requieren (tipo y cantidad)?

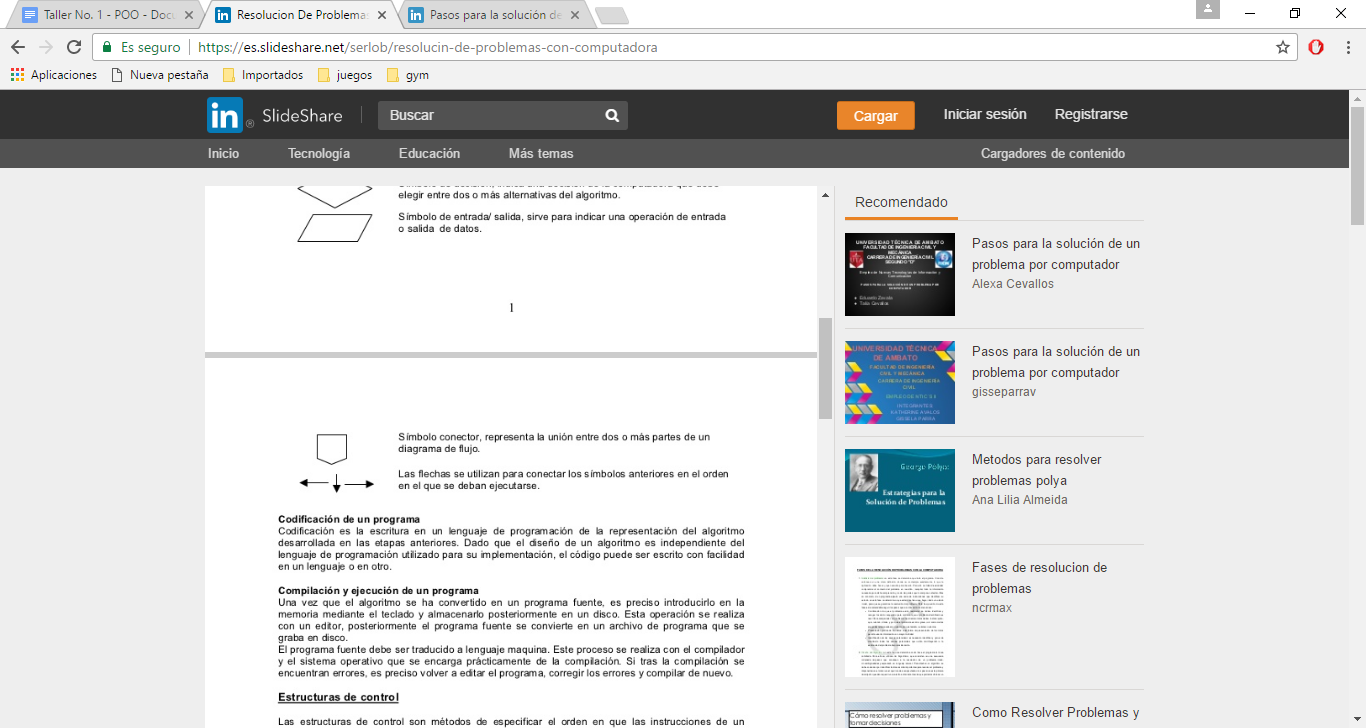
- ¿Cuál es la salida deseada (tipo y cantidad)?

- ¿Qué método produce la salida deseada?

**Diseño del algoritmo:** En la etapa de análisis del proceso de programación se determina que hace el programa. En la etapa de diseño se determina cómo hace el programa la tarea solicitada. Para ello vamos a utilizar una herramienta de programación llamada Diagrama de Flujo (flowchart).

Diagramas de flujo Un diagrama de flujo es una representación gráfica de un algoritmo. Los símbolos más utilizados se detallan a continuación:





**CODIFICACIÓN**: es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas anteriores. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizado para su implementación, el código puede ser escrito con facilidad en un lenguaje o en otro.

**COMPILACIÓN Y EJECUCIÓN DE UN PROGRAMA**: Una vez que el algoritmo se ha convertido en un programa fuente, es preciso introducirlo en la memoria mediante el teclado y almacenarlo posteriormente en un disco. Esta operación se realiza con un editor, posteriormente el programa fuente se convierte en un archivo de programa que se graba en disco. El programa fuente debe ser traducido a lenguaje máquina. Este proceso se realiza con el compilador y el sistema operativo que se encarga prácticamente de la compilación. Si tras la compilación se encuentran errores, es preciso volver a editar el programa, corregir los errores y compilar de nuevo.

2.R/

**-El lenguaje informático** es el lenguaje usado por, o asociado con, la computación. Si bien es muy frecuente que este término se emplee como sinónimo de lenguaje de programación; un lenguaje informático no tiene por qué ser, necesariamente, un lenguaje de programación.

Un buen ejemplo de esto es el HTML, el cual no es un lenguaje de programación, pero sí es un lenguaje informático.

En general, como cualquier otro lenguaje, el informático ha sido creado con la finalidad de transmitir una información de algo a alguien basado en la computación.

Por otra parte, el lenguaje de programación es el medio que utilizan los desarrolladores para crear un programa de computación; mientras que el informático, es un lenguaje de marcas empleado como medio para describir a un computador el formato o la estructura de un documento.

Asimismo, el lenguaje de programación es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones y es utilizado para controlar el comportamiento físico y lógico de una máquina.

-**PROGRAMA (MÁQUINA),** lista de instrucciones escritas en un lenguaje de programación utilizadas para controlar las tareas de una máquina.

-**PROGRAMA INFORMÁTICO**, secuencia de instrucciones que una computadora puede interpretar y ejecutar.

**COMPILADORES E INTERPRETES**

**Compilador**, que analiza el programa fuente y "lo traduce" a otro equivalente escrito en otro lenguaje (por ejemplo, en el lenguaje de la máquina). Su acción equivale a la de un traductor humano, que toma un libro y produce otro equivalente escrito en otra lengua.

**Intérprete**, que analiza el programa fuente y "lo ejecuta directamente", sin generar ningún código equivalente. Su acción equivale a la de un intérprete humano, que traduce las frases que oye sobre la marcha, sin producir ningún escrito permanente. Intérpretes y compiladores tienen diversas ventajas e inconvenientes que los hacen complementarios:

**Un intérprete** facilita la búsqueda de errores, pues la ejecución de un programa puede interrumpirse en cualquier momento para estudiar el entorno (valores de las variables, etc.). Además, el programa puede modificarse sobre la marcha, sin necesidad de volver a comenzar la ejecución.

**Un compilador** suele generar programas más rápidos y eficientes, ya que el análisis del lenguaje fuente se hace una sola vez, durante la generación del programa equivalente. En cambio, un intérprete se ve obligado generalmente a analizar cada instrucción tantas veces como se ejecute (incluso miles o millones de veces).

Un intérprete permite utilizar funciones y operadores más potentes, como por ejemplo ejecutar código contenido en una variable en forma de cadenas de caracteres. Usualmente, este tipo de instrucciones es imposible de tratar por medio de compiladores. Los lenguajes que incluyen este tipo de operadores y que, por tanto, exigen un intérprete, se llaman interpretativos. Los lenguajes compilativos, que permiten el uso de un compilador, prescinden de este tipo de operadores. En pocas palabras "el compilador sólo traduce; el intérprete decodifica y ejecuta. "

3.R/

**Antecedentes de la programación**

orientada a objetos (POO)

La POO es una metodología para el desarrollo lógico de programas que se

remonta a la década de 1960. Sus conceptos fundamentales ya existían en lenguajes

como Simula (desarrollado en 1967) y Smalltalk (que data de la década

de 1970). A finales de la década de 1960 y principios de la de 1970 surgió una

crisis de software derivada, por un lado, de la dificultad de cambiar y modificar

los grandes y complejos programas existentes, y por otro, de la disminución de

programadores dispuestos a dar mantenimiento a los programas. Esto trajo

consigo la introducción de una nueva metodología de desarrollo denominada

programación estructurada, mediante la cual se descompone en procedimientos

individuales (funciones) a los programas. Un procedimiento realiza una tarea

específica (esto es una abstracción funcional): este concepto constituye la base

sobre el cual trabaja la programación estructurada. En un programa estructurado,

cada procedimiento debe realizar una tarea específica; cómo se realiza la

tarea no es tan importante, pues mientras el procedimiento sea fiable, se

puede utilizar.

En la programación estructurada es crítica la coordinación entre los desarrolladores

implicados en el proyecto, pues dado que cada uno de ellos estará manipulando

funciones separadas que pueden referirse a tipos de datos mutuamente compartidos,

los cambios de un desarrollador deben reflejarse en el trabajo del resto del

equipo.

Una debilidad de la programación estructurada es la separación conceptual de

datos y código; esto se agrava a medida que el programa crece y se hace más

complejo.

Actualmente, las sofisticadas interfaces de usuarios, los sistemas de ventanas,

los programas multitareas, entre otros elementos, hacen más compleja la manipulación

de las relaciones existentes entre los componentes de software. Por tanto,

se requiere de una nueva técnica que mejore las limitaciones de los sistemas

anteriores.

La programación orientada a objetos no es la solución definitiva, pero permite

construir programas complejos a partir de entidades de software más simples

llamadas objetos, que son instancias reales o muestras de clases, lo cual permite

heredar datos y códigos de aplicaciones existentes. Esto mejora la productividad

del programador y facilita la extensión y la reutilización de clases en otras aplicaciones

con mínima modificación al código original.

Así como la programación estructurada o procedural se construye alrededor de

funciones, la programación orientada a objetos se construye alrededor de clases.

¿Existe alguna relación entre ambas? Sí: las clases que hacemos son construidas

usando elementos de la programación estructurada, llamadas funciones (lo que

en POO llamamos métodos).

4.R/**Conceptos fundamentales**

La POO es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

**Clase**

Definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ella.

**Herencia**

Por ejemplo, herencia de la clase C a la clase D, es la facilidad mediante la cual la clase D hereda en ella cada uno de los atributos y operaciones de C, como si esos atributos y operaciones hubiesen sido definidos por la misma D. Por lo tanto, puede usar los mismos métodos y variables públicas declaradas en C. Los componentes registrados como "privados" (private) también se heredan, pero como no pertenecen a la clase, se mantienen escondidos al programador y sólo pueden ser accedidos a través de otros métodos públicos. En el caso de los componentes registrados como "protegidos" (protected) también se heredan, pero solo para esa clase, no para futuras clases heredadas. Esto es así para mantener hegemónico el ideal de POO.

**Objeto**

Instancia de una clase. Entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos), los mismos que consecuentemente reaccionan a eventos. Se corresponden con los objetos reales del mundo que nos rodea, o con objetos internos del sistema (del programa).

**Método**

Algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

**Evento**

Es un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento la reacción que puede desencadenar un objeto; es decir, la acción que genera.

**Atributos**

Características que tiene la clase.

**Mensaje**

Una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

**Propiedad o atributo**

Contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

**Estado interno**

Es una variable que se declara privada, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos). No es visible al programador que maneja una instancia de la clase.

**Componentes de un objeto**

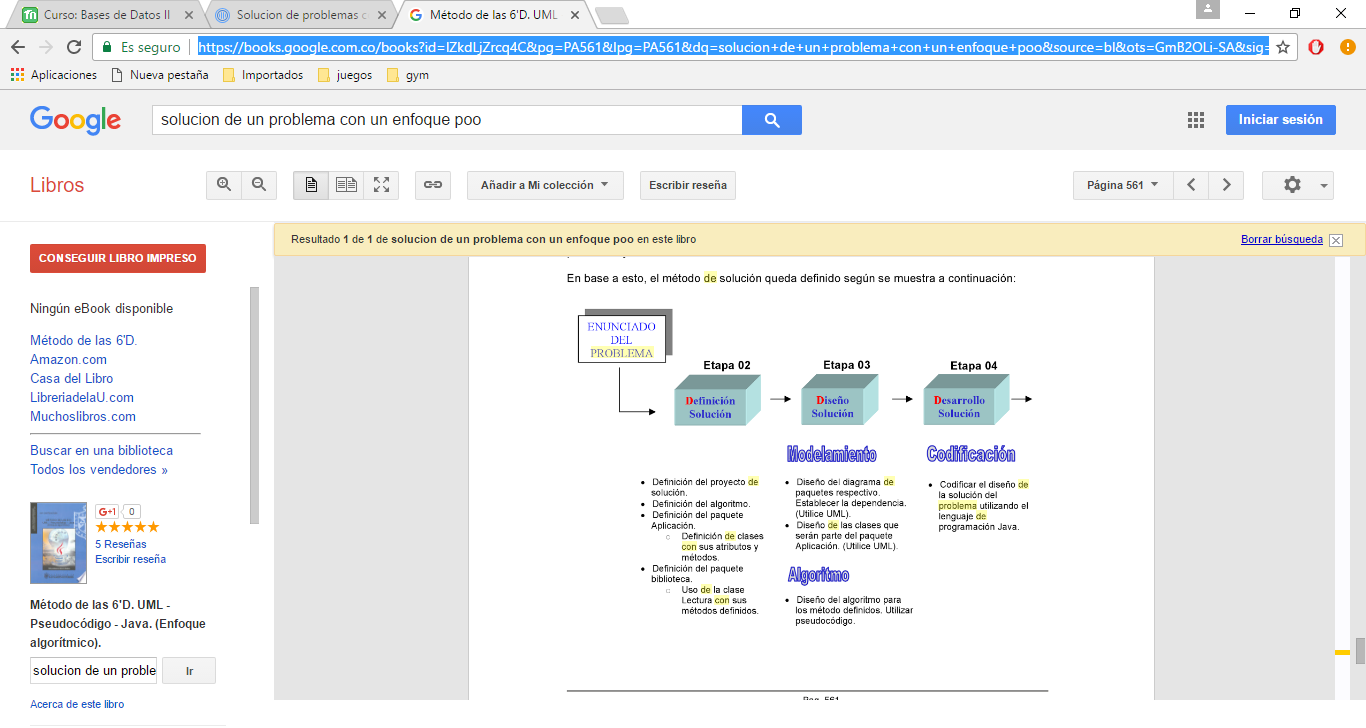
Atributos, identidad, relaciones y métodos.

**Identificación de un objeto**

Un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

En comparación con un lenguaje imperativo, una "variable" no es más que un contenedor interno del atributo del objeto o de un estado interno, así como la "función" es un procedimiento interno del método del objeto.

5.R/



6.R/

**TIPOS DE DATOS**

**A. Tipos de datos simples**

Es uno de los conceptos fundamentales de cualquier lenguaje de programación. Estos definen los métodos de almacenamiento disponibles para representar información, junto con la manera en que dicha información ha de ser interpretada.

Para crear una variable (de un tipo simple) en memoria debe declararse indicando su tipo de variable y su identificador que la identificará de forma única. La sintaxis de declaración de variables es la siguiente:

TipoSimple Identificador1, Identificador2;

Esta sentencia indica al compilador que reserve memoria para dos variables del tipo simple TipoSimple con nombres Identificador1 e Identificador2.

Los tipos de datos en Java pueden dividirse en dos categorías: simples y compuestos. Los simples son tipos nucleares que no se derivan de otros tipos, como los enteros, de coma flotante, booleanos y de carácter. Los tipos compuestos se basan en los tipos simples, e incluyen las cadenas, las matrices y tanto las clases como las interfaces, en general.

Cada tipo de datos simple soporta un conjunto de literales que le pueden ser asignados, para darles valor. En este apartado se explican los tipos de datos simples (o primitivos) que presenta Java, así como los literales que soporta (sintaxis de los valores que se les puede asignar).

**a.) Tipos de datos enteros**

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

Tipo Tamaño

byte 1Byte (8 bits)

short 2 Bytes (16 bits)

int 4 Bytes (32 bits)

long 8 Bytes (64 bits)

Tipos de datos enteros

Literales enteros

Son básicos en la programación en Java y presentan tres formatos:

• Decimal: Los literales decimales aparecen como números ordinarios sin ninguna notación especial.

• Hexadecimal: Los hexadecimales (base 16) aparecen con un 0x ó 0X inicial, notación similar a la utilizada en C y C++.

• Octal: Los octales aparecen con un 0 inicial delante de los dígitos.

Por ejemplo, un literal entero para el número decimal 12 se representa en Java como 12 en decimal, como 0xC en hexadecimal, y como 014 en octal.

Los literales enteros se almacenan por defecto en el tipo int, (4 bytes con signo), o si se trabaja con números muy grandes, con el tipo long, (8 bytes con signo), añadiendo una L ó l al final del número.

La declaración de variables enteras es muy sencilla. Un ejemplo de ello sería:

long numeroLargo = 0xC; // Por defecto vale 12

**b.) Tipos de datos en coma flotante**

Se usan para representar números con partes fraccionarias. Hay dos tipos de coma flotante: float y double. El primero reserva almacenamiento para un número de precisión simple de 4 bytes y el segundo lo hace para un numero de precisión doble de 8 bytes.

Tipo Tamaño

float 4 Byte (32 bits)

double 8 Bytes (64 bits)

Tipos de datos numéricos en coma flotante

Literales en coma flotante

Representan números decimales con partes fraccionarias. Pueden representarse con notación estándar (563,84) o científica (5.6384e2).

De forma predeterminada son del tipo double (8 bytes). Existe la opción de usar un tipo más corto (el tipo float de 4 bytes), especificándolo con una F ó f al final del número.

La declaración de variables de coma flotante es muy similar a la de las variables enteras. Por ejemplo:

double miPi = 314.16e-2 ; // Aproximadamente

float temperatura = (float)36.6; // Paciente sin fiebre

Se realiza un moldeado a temperatura, porque todos los literales con decimales por defecto se consideran double.

**c.) Tipo de datos boolean**

Se usa para almacenar variables que presenten dos estados, que serán representados por los valores true y false. Representan valores bi-estado, provenientes del denominado álgebra de Boole.

Literales Booleanos

Java utiliza dos palabras clave para los estados: true (para verdadero) y false (para falso). Este tipo de literales es nuevo respecto a C/C++, lenguajes en los que el valor de falso se representaba por un 0 numérico, y verdadero cualquier número que no fuese el 0.

Para declarar un dato del tipo booleano se utiliza la palabra reservada boolean:

boolean reciboPagado = false; // ¡¿Aun no nos han pagado?!

**d.) Tipo de datos carácter**

Se usa para almacenar caracteres Unicode simples. Debido a que el conjunto de caracteres Unicode se compone de valores de 16 bits, el tipo de datos char se almacena en un entero sin signo de 16 bits.

Java a diferencia de C/C++ distingue entre matrices de caracteres y cadenas.

Literales carácter

Representan un único carácter (de la tabla de caracteres Unicode 1.1) y aparecen dentro de un par de comillas simples. De forma similar que en C/C++. Los caracteres especiales (de control y no imprimibles) se representan con una barra invertida ('\') seguida del código carácter.

Descripción Representación Valor Unicode

Caracter Unicode \udddd

Numero octal \ddd

Barra invertida \\ \u005C

Continuación \ \

Retroceso \b \u0008

Retorno de carro \r \u000D

Alimentación de formularios \f \u000C

Tabulación horizontal \t \u0009

Línea nueva \n \u000A

Comillas simples \’ \u0027

Comillas dobles \" \u0022

Números arábigos ASCII 0-9 \u0030 a \u0039

Alfabeto ASCII en mayúsculas A.-Z \u0041 a \u005A

Alfabeto ASCII en minúsculas a.-z \u0061 a \u007A

Caracteres especiales Java

Las variables de tipo char se declaran de la siguiente forma:

char letraMayuscula = 'A'; // Observe la necesidad de las ' '

char letraV = '\u0056'; // Letra 'V'

**e.) Conversión de tipos de datos**

En Java es posible transformar el tipo de una variable u objeto en otro diferente al original con el que fue declarado. Este proceso se denomina "conversión", "moldeado" o "tipado". La conversión se lleva a cabo colocando el tipo destino entre paréntesis, a la izquierda del valor que queremos convertir de la forma siguiente:

char c = (char)System.in.read();

La función read devuelve un valor int, que se convierte en un char debido a la conversión (char), y el valor resultante se almacena en la variable de tipo carácter c.

El tamaño de los tipos que queremos convertir es muy importante. No todos los tipos se convertirán de forma segura. Por ejemplo, al convertir un long en un int, el compilador corta los 32 bits superiores del long (de 64 bits), de forma que encajen en los 32 bits del int, con lo que, si contienen información útil, esta se perderá.

Por ello se establece la norma de que "en las conversiones el tipo destino siempre debe ser igual o mayor que el tipo fuente":

Tipo Origen Tipo Destino

byte double, float, long, int, char, short

short double, float, long, int

char double, float, long, int

int double, float, long

long double, float

float double

Conversiones sin pérdidas de información

**B. Vectores y Matrices**

Una matriz es una construcción que proporciona almacenaje a una lista de elementos del mismo tipo, ya sea simple o compuesto. Si la matriz tiene solo una dimensión, se la denomina vector.

En Java los vectores se declaran utilizando corchetes ( [ y ] ), tras la declaración del tipo de datos que contendrá el vector. Por ejemplo, esta sería la declaración de un vector de números enteros (int):

int vectorNumeros[ ]; // Vector de números

Se observa la ausencia de un número que indique cuántos elementos componen el vector, debido a que Java no deja indicar el tamaño de un vector vacío cuando le declara. La asignación de memoria al vector se realiza de forma explícita en algún momento del programa.

Para ello o se utiliza el operador new:

int vectorNumeros = new int[ 5 ]; // Vector de 5 números

O se asigna una lista de elementos al vector:

int vectorIni = { 2, 5, 8}; // == int vectorIni[3]=new int[3];

Se puede observar que los corchetes son opcionales en este tipo de declaración de vector, tanto después del tipo de variable como después del identificador.

Si se utiliza la forma de new se establecerá el valor 0 a cada uno de los elementos del vector.

**C. Cadenas**

En Java se tratan como una clase especial llamada String. Las cadenas se gestionan internamente por medio de una instancia de la clase String. Una instancia de la clase String es un objeto que ha sido creado siguiendo la descripción de la clase.

Cadenas constantes

Representan múltiples caracteres y aparecen dentro de un par de comillas dobles. Se implementan en Java con la clase String. Esta representación es muy diferente de la de C/C++ de cadenas como una matriz de caracteres.

Cuando Java encuentra una constante de cadena, crea un caso de la clase String y define su estado, con los caracteres que aparecen dentro de las comillas dobles.

Vemos un ejemplo de cadena declarada con la clase String de Java:

String capitalUSA = "Washington D.C.";

String nombreBonito = "Amelia";

Cómo utilizar los tipos de datos de Java

public class DataTypes {

public static void main(String[] args) {

boolean isReal=true; // Los nombres son sensibles a // mayúsculas y minúsculas, // deben empezar por una letra y // pueden contener números,\_,$

byte d= 122; // Deben ser inferiores a 127

short e= -29000; // Deben ser inferiores a 32767

int f= 100000; // Deben ser inferiores a 2100 mill.

long g= 999999999999L; // Deben poner L al final

float h= 234.99F; // Deben ser < 3E38; F al final

double i= 55E100; char cvalue= '4'; // char '4' no es el entero 4

//Las cadenas (strings) son objetos, no primitivos. //Ejemplo: String nombre= “Claudius”;

}

}

7.R/

**CONSTANTES**

Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.

Ejemplo:

pi = 3.1416

**VARIABLE**

Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa.

Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo.

Ejemplo:

area = pi \* radio ^ 2

Las variables son: el radio, el area y la constate es pi

Clasificación de las Variables

Por su contenido

Variables Numéricas: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal.

Ejemplo:

IVA = 0.15 pi = 3.1416 costo = 2500

Variables Lógicas: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

Variables Alfanuméricas: Está formada por caracteres alfanuméricos (letras, números y caracteres especiales).

Ejemplo:

letra = ’a’ apellido = ’lopez’ direccion = ’Av. Libertad #190’

Por su uso

Variables de Trabajo: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa.

Ejemplo:

Suma = a + b /c

Contadores: Se utilizan para llevar el control del número de ocasiones en que se realiza una operación o se cumple una condición. Con los incrementos generalmente de uno en uno.

Acumuladores: Forma que toma una variable y que sirve para llevar la suma acumulativa de una serie de valores que se van leyendo o calculando progresivamente.

**EXPRESIONES**

Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales.

Por ejemplo:

a + (b + 3) / c

Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones indicadas.

Una expresión consta de operadores y operandos. Según sea el tipo de datos que manipulan, se clasifican las expresiones en:

Aritméticas

Relacionales

Lógicas

8.R/

9.R/

**DEFINICIÓN Y CONCEPTO DE UML**

UML son las siglas de “Unified Modeling Language” o “Lenguaje Unificado de Modelado”. Se trata de un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa a los desarrollos de software (programas informáticos).

¿QUÉ ES Y PARA QUÉ SIRVE UML?

El término “lenguaje” ha generado bastante confusión respecto a lo que es UML. En realidad, el término lenguaje quizás no es el más apropiado, ya que no es un lenguaje propiamente dicho, sino una serie de normas y estándares gráficos respecto a cómo se deben representar los esquemas relativos al software. Mucha gente piensa por confusión que UML es un lenguaje de programación y esta idea es errónea: UML no es un lenguaje de programación. Como decimos, UML son una serie de normas y estándares que dicen cómo se debe representar algo.

UML es una herramienta propia de personas que tienen conocimientos relativamente avanzados de programación y es frecuentemente usada por analistas funcionales (aquellos que definen qué debe hacer un programa sin entrar a escribir el código) y analistas-programadores (aquellos que, dado un problema, lo estudian y escriben el código informático para resolverlo en un lenguaje como Java, C#, Python o cualquier otro). Por tanto, si estás dando tus primeros pasos en programación, te recomendaríamos que te olvides de UML hasta que tengas unos conocimientos mínimos como uso de condicionales, bucles, y conocimiento de la programación orientada a objetos. Esto es solo una recomendación, en realidad prácticamente cualquier persona puede usar UML, incluso podría usarse para realizar esquemas o documentación de procesos que no tengan que ver con la informática.

10.R/

**Descripción y características de este potente y moderno lenguaje de programación.**

Java es un lenguaje de programación con el que podemos realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras.

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que si hacemos un programa en Java podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo Windows, Linux, Apple, etc. Esto lo consigue porque se ha creado una Máquina de Java para cada sistema que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente.

La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Pero no se queda ahí, Java está desarrollándose incluso para distintos tipos de dispositivos además del ordenador como móviles, agendas y en general para cualquier cosa que se le ocurra a la industria.

Pasado y presente

Java fue pensado originalmente para utilizarse en cualquier tipo de electrodoméstico, pero la idea fracasó. Uno de los fundadores de Sun rescató la idea para utilizarla en el ámbito de Internet y convirtieron a Java en un lenguaje potente, seguro y universal gracias a que lo puede utilizar todo el mundo y es gratuito. Una de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página web, hasta entonces impensable con el HTML.

Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Para lo que nos interesa a nosotros, con Java podemos programar páginas web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que deseemos hacer con acceso a través web se puede hacer utilizando Java.

# Bibliografía

*books.google*. (s.f.). Obtenido de books.google: https://books.google.com.co/books?id=lZkdLjZrcq4C&pg=PA561&lpg=PA561&dq=solucion+de+un+problema+con+un+enfoque+poo&source=bl&ots=GmB2OLi-SA&sig=TLemdaZlU17sDxDgEGi7AjC\_5rA&hl=es&sa=X&ved=0ahUKEwiPnZ\_F6azSAhVLSCYKHVQwApcQ6AEIOjAF#v=onepage&q=solucion%20de%

*desarrolloweb.com*. (s.f.). Obtenido de desarrolloweb.com: https://desarrolloweb.com/articulos/497.php

*monografias*. (s.f.). Obtenido de monografias: http://www.monografias.com/trabajos14/progorie/progorie.shtml#CARACT

*poo-karinajarillolaguna*. (s.f.). Obtenido de poo-karinajarillolaguna: http://poo-karinajarillolaguna.blogspot.com.co/2010/05/tipos-de-datos-en-java.html

*slideshare*. (s.f.). Obtenido de slideshare: https://es.slideshare.net/serlob/resolucin-de-problemas-con-computadora

*wikipedia*. (s.f.). Obtenido de wikipedia: https://es.wikipedia.org/wiki/Programaci%C3%B3n\_orientada\_a\_objetos#Conceptos\_fundamentales