**Taller No. 1 – POO**

**Cristhian Urrego Salazar**

**Profesor**

**Carlos Alberto Londoño Loaiza**

**Centro de Estudios Tecnológicos del Norte del Valle**

**COTECNOVA**

**Tecnico Profesional en Programación de Aplicaciones Informáticas**

**Cartago Valle**

**2017**

**Metodología para la resolución de un problema a través de la computadora**

La solución de un problema por computadora, requiere de siete pasos, dispuestos de tal forma que cada uno es dependiente de los anteriores, lo cual indica que se trata de un proceso complementario y por lo tanto cada paso exige el mismo cuidado en su elaboración. Los siete pasos de la metodología son los siguientes:

1. Definición del problema.

2. Análisis de la solución.

3. Diseño de la solución.

4. Codificación.

5. Prueba y depuración.

6. Documentación.

7. Mantenimiento.

**1. Definición de problema:** Es el enunciado del problema, el cual debe ser claro y completo. Es fundamental conocer y delimitar por completo el problema, saber qué es lo que se desea que realice la computadora, mientras esto no se conozca del todo, no tiene caso continuar con el siguiente paso.

**2. Análisis de la solución:** Consiste en establecer una serie de preguntas acerca de lo que establece el problema, para poder determinar si se cuenta con los elementos suficientes para llevar a cabo la solución del mismo.

**3. Diseño de la solución:** Una vez definido y analizado el problema, se produce a la creación del algoritmo (Diagrama de flujo o pseudocódigo) en el cual se da la serie de pasos ordenados que nos proporcione un método explicito para la solución del problema.

**4. Codificación:** Consiste en escribir la solución del problema (de acuerdo al pseudocódigo); en una serie de instrucciones detalladas en un código reconocible por la computadora; es decir un lenguaje de programación (ya sea de bajo o alto nivel), a esta serie de instrucciones se le conoce como PROGRAMA.

**5. Prueba y depuración:** Prueba es el proceso de identificar los errores que se presenten durante la ejecución de programa. La depuración consiste en eliminar los errores que se hayan detectado durante la prueba, para dar paso a una situación adecuada y sin errores.

**6. Documentación:** Es la guía o comunicación escrita que sirve como ayuda para usar el programa, o facilitar futuras modificaciones. A menudo, un programa escrito por una persona es usado por muchas otras, por ello la documentación es muy importante; esta debe presentarse en tres formas: EXTERNA, INTERNA y al USUARIO FINAL.

-**Interna**: Consiste en los comentarios o mensajes que se agregan al código de programa, que aplican las funciones que realizan ciertos procesos, cálculos o formulas, para el entendimiento del mismo.

-**Externa**: Está integrada por los siguientes elementos: Descripción del problema, nombre del autor, diagrama de flujo y/o pseudocódigo, listas de variables y constantes, y codificación del programa, esto con la finalidad de permitir su posterior adecuación a los cambios.

-**Usuario Final**: es la documentación que se le proporciona al usuario final, es una guía que indica al usuario como navegar en el programa, presentando todas las pantallas y menús que se va a encontrar y una explicación de los mismos, no contiene información de tipo técnico.

**7. Mantenimiento:** Se lleva a cabo después de  terminado el programa, cuando se ha estado trabajando un tiempo, y se detecta que es necesario hacer un cambio, ajuste y/o complementación al programa para que siga trabajando de manera correcta. Para realizar esta función el programa debe estar debidamente documentado, lo cual facilitará la tarea.

**Definiciones**

**Lenguaje – Programa – Compilador - Intérprete**

**Lenguaje:** Un lenguaje de programación consiste en todos los símbolos, caracteres y reglas de uso que permiten a las personas "comunicarse" con las computadoras. Existen varios cientos de lenguajes y dialectos de programación diferentes. Algunos se crean para una aplicación especial, mientras que otros son herramientas de uso general más flexibles que son apropiadas para muchos tipos de aplicaciones. Los lenguajes de programación deben tener instrucciones que pertenecen a las categorías ya familiares de entrada/salida, cálculo/manipulación de textos, lógica/comparación y almacenamiento / recuperación.

Aunque todos los lenguajes de programación tienen un conjunto de instrucciones que permiten realizar dichas operaciones, existe una marcada diferencia en los símbolos, caracteres y sintaxis de los lenguajes de máquina, lenguajes ensambladores y lenguajes de alto nivel.

**Programa:** Es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora. Sin programas, estas máquinas no pueden funcionar. Al conjunto general de programas, se le denomina software, que más genéricamente se refiere al equipamiento lógico o soporte lógico de una computadora digital.

**Compilador:** Un compilador analiza el programa y lo traduce al idioma "maquina". La acción fundamental los compiladores es equivalente a la de un traductor humano, que toma nota de lo que está escuchando y reproduce por escrito en otra lengua.

**Interprete:** Analiza el programa fuente y lo ejecuta directamente, o sea en el ejemplo del traductor humano, éste sería un traductor humano que conforme a lo que está escuchando va ejecutando, sin generar ningún escrito, es decir que sobre la marcha va traduciendo.

**Antecedentes de la Programación Orientada a Objetos**

La Programación Orientación a Objetos (P.O.O.) surge en Noruega en 1967 con un lenguaje llamado Simula 67, desarrollado por Krinsten Nygaard y Ole-Johan Dahl, en el centro de cálculo noruego.

Simula 67 introdujo por primera vez los conceptos de clases, corrutinas y subclases (conceptos muy similares a los lenguajes Orientados a Objetos de hoy en día).

El nacimiento de la Programación Orientación a Objetos en Europa pasó inadvertido para gran parte de los programadores. Hoy tenemos la Orientación a Objetos como un niño de 33 años al que todos quieren bautizar.

Uno de los problemas de inicio de los años setentas era que pocos sistemas lograban terminarse, pocos se terminaban con los requisitos iniciales y no todos los que se terminaban cumpliendo con los requerimientos se usaban según lo planificado. El problema consistía en cómo adaptar el software a nuevos requerimientos imposibles de haber sido planificados inicialmente.

Este alto grado de planificación y previsión es contrario a la propia realidad. El hombre aprende y crea a través de la experimentación, no de la planeación. La Orientación a Objetos brinda estos métodos de experimentación, no exige la planificación de un proyecto por completo antes de escribir la primera línea de código.

En los 70’s científicos del centro de investigación en Palo Alto Xerox (Xerox park) inventaron el lenguaje Small talk que dio respuesta al problema anterior (investigar no planificar).

Small talk fue el primer lenguaje Orientado a Objetos puro de los lenguajes Orientados a Objetos, es decir, únicamente utiliza clases y objetos (Java usa tipos de datos primitivos, o bien los Wrappers que son clases que encapsulan tipos de datos primitivos).

Quien tuvo la idea fue D. Parnas cuando propuso la disciplina de ocultar la información. Su idea era encapsular cada una de las variables globales de la aplicación en un solo módulo junto con sus operaciones asociadas, sólo mediante las cuales se podía tener acceso a esas variables.

El resto de los módulos (objetos) podían acceder a las variables sólo de forma indirecta mediante las operaciones diseñadas para tal efecto.

En los años 80’s Bjarne Stroustrup de AT&T Labs., amplió el lenguaje C para crear C++ que soporta la programación Orientada a Objetos.

En esta misma década se desarrollaron otros lenguajes Orientados a Objetos como Objective C, Common Lisp Object System (CIOS), object Pascal, Ada y otros.

Posteriores mejoras en herramientas y lanzamientos comerciales de C++ por distintos fabricantes, justificaron la mayor atención hacia la programación Orientada a Objetos en la comunidad de desarrollo de software. El desarrollo técnico del hardware y su disminución del costo fue el detonante final. Con más computadoras al alcance de más personas más programadores, más problemas y más algoritmos surgieron.

En el inicio de los 90’s se consolida la Orientación a Objetos como una de las mejores maneras para resolver problemas. Aumenta la necesidad de generar prototipos más rápidamente (concepto RAD Rapid Aplication Developments). Sin esperar a que los requerimientos iniciales estén totalmente precisos.

En 1996 surge un desarrollo llamado JAVA (extensión de C++). Su filosofía es aprovechar el software existente. Facilitar la adaptación del mismo a otros usos diferentes a los originales sin necesidad de modificar el código ya existente.

En 1997-98 se desarrollan herramientas ‘CASE’ orientadas a objetos (como el diseño asistido por computadora).

Del 98 a la fecha se desarrolla la arquitectura de objetos distribuidos RMI, Corba, COM, DCOM.

Actualmente la orientación a objetos parece ser el mejor paradigma, no obstante, no es una solución a todos los problemas. Trata de eliminar la crisis del software. Entre los creadores de metodologías orientadas a objetos se encuentran: G. Booch, Rambaught, Ivar Jacobson y Peter Cheng.

**Conceptos Involucrados en la Programación Orientada a Objetos**

**Clase**: Las clases son uno de los principales componentes de un lenguaje de programación, pues en ellas ocurren todos los procesos lógicos requeridos para un sistema, en si podemos definirlas como estructuras que representan objetos del mundo real, tomando como objetos a personas, lugares o cosas, en general las clases poseen propiedades, comportamientos y relaciones con otras clases del sistema. Una clase se compone por tres partes fundamentales:

Nombre: Contiene el Nombre de la Clase.

Atributos: Representan las propiedades que caracterizan la clase.

Métodos: Representan el comportamiento u operaciones, la forma como interactúa la clase con su entorno.

**Objeto:** Los objetos representan una entidad concreta o abstracta del mundo real, en programación básicamente se le conoce como la instancia de una clase en si es lo que da el sentido a estas. Al igual que las clases se componen de tres partes fundamentales:

Estado: Representa los atributos o características con valores concretos del objeto.

Comportamiento: Se define por los métodos u operaciones que se pueden realizar con él.

Identidad: Es la propiedad única que representa al objeto y lo diferencia del resto.

En java se representa creando una instancia de la clase por medio de la palabra *new* al hacer eso creamos el objeto de la clase y podemos hacer uso de los métodos o atributos de esta (dependiendo de la visibilidad de los mismos) por medio de un punto (.).

**Herencia:** La herencia en java representa lo que conocemos de herencia en el mundo real, básicamente mediante esta obtenemos las características o rasgos comunes de nuestros padres o abuelos, en java es el mismo enfoque permitiendo la creación de nuevas clases basadas en clases ya existentes, con las cuales podemos obtener las características de las clases padres, heredando campos, atributos, métodos o funcionalidades.

En Java solo se puede heredar de una sola clase padre y se representa mediante la palabra *extends*.

**Encapsulamiento:** Este concepto es uno de los más importantes en términos de seguridad dentro de nuestra aplicación, la encapsulación es la forma de proteger nuestros datos dentro del sistema, estableciendo básicamente los permisos o niveles de visibilidad o acceso de nuestros datos. Se representa por 3 niveles:

Público: Se puede acceder a todos los atributos o métodos de la clase.

Protegido: Se puede acceder a los atributos o métodos solo en la misma jerarquía de herencia.

Privado: Solo se puede acceder a los atributos o métodos de la clase en la que se encuentran.

Con la Encapsulación mantenemos nuestros datos seguros, ya que podemos evitar que por ejemplo se hagan modificaciones al estado o comportamiento de un objeto desde una clase externa, una buena práctica es trabajar con métodos *setter* y *getter* que permiten manipular nuestros datos de forma segura.

**Clases Abstractas:** La abstracción permite resaltar la parte más representativa de algo, ignorando detalles para centrarse en lo principal. La imagen es muy fácil de identificar, con base a ella podemos crear una clase persona, o la clase hombre, humano entre otras, pero obviamente vemos que la imagen no tiene elementos como ojos, nariz, boca, rostro en general, ni dedos, pies, manos o cuello, ¿pero entonces porque decimos que es una persona?, Precisamente aquí estamos aplicando el concepto de abstracción, ya que nos fijamos en lo más representativo de algo, en este caso vemos que se tiene una cabeza, tronco, brazos y pies, con esto es suficiente para saber que es una persona sin fijarnos en los detalles mencionados anteriormente.

Las clases abstractas permiten crear métodos generales con un comportamiento común para otras clases concretas sin importar sus características ni el comportamiento que usen para dichos métodos.

La Abstracción en java solo tiene lógica mediante la Herencia, ya que una clase abstracta posee al menos un método abstracto el cual no tiene implementación, el comportamiento de estos métodos lo definen las clases concretas que lo hereden. Podemos usarlos cuando existan varias clases con características o acciones comunes pero con diferentes comportamientos, mediante el uso de la herencia y componentes abstractos hacemos más óptima y organizada nuestra aplicación. (*Hay que tener en cuenta que a diferencia de las clases concretas, las clases abstractas no se pueden instanciar*).

**Interfaces:** Las interfaces son el mecanismo que utiliza Java para simular la herencia múltiple, como mencionamos en Java solo se puede extender de una sola clase, mediante el uso de interfaces esto se puede simular ya que el lenguaje permite implementar el número de interfaces que necesitemos, básicamente son clases completamente abstractas, es común relacionarlas con un contrato en el que se define que se debe hacer, así cada clase concreta que implemente una interfaz está obligada a implementar todos los métodos que la compongan. Las interfaces definen lo que la clase que la implemente deberá hacer, más no la forma como lo hará.

Al decir que las interfaces son clases completamente abstractas significa que todos sus métodos lo son y por ende no poseen implementación, no requieren el uso de la palabra reservada *abstract*, ya que al ser completamente abstracta todo dentro de ella lo es, al igual que las clases abstractas la implementación de los métodos depende de las clases concretas que las usen.

*Se debe tener en cuenta que toda variable definida en una interfaz automáticamente se convierte en una constante, además tampoco se puede instanciar una interfaz.*

**Polimorfismo:** Este tal vez sea uno de los conceptos de la programación orientada a objetos más usados pero muchas veces sin saber que se aplica ya que el concepto inicialmente puede ser un poco confuso, básicamente mediante el polimorfismo programamos de forma general en lugar de hacerlo de forma específica, se usa cuando se trabajen con la herencia y objetos de características comunes los cuales comparten la misma *superClase* y árbol jerárquico, al trabajar con este concepto optimizamos y simplificamos en gran medida nuestro trabajo. Básicamente podemos definirlo como la capacidad que tienen los objetos de comportarse de múltiples formas sin olvidar que para esto se requiere de la herencia, en si consiste en hacer referencia a objetos de una clase que puedan tomar comportamientos de objetos descendientes de esta.

*Con el polimorfismo usamos la generalización olvidando los detalles concretos de los objetos para centrarnos en un punto en común mediante una clase padre.*

Tomando como ejemplo podemos decir que un objeto de la clase *FiguraGeometrica* puede usarse para referirse a cualquier objeto de cualquier *subClase* de *FiguraGeometrica*, en otras palabras una figura geométrica puede ser un *cuadro*, un *triángulo*, un *cuadrado* o cualquier figura que en términos generales sea geométrica.

**Método Para la Resolución de Problemas Mediante la Programación Orientada a Objetos**

Resulta útil establecer un método que mermita llegar a la solución de un determinado problema, a través de etapas bien definidas. Este método presenta 6 etapas:

**Etapa 01 – Descripción del problema:** Es necesario saber y poder comprender cual es el problema principal, es decir, que es lo que se desea que se resuelva. Para esto, se deberá escribir un enunciado claro, concreto y conciso del problema a resolver.

**Etapa 02 – Definición de la solución:** Es necesario estudiar a fondo determinado problema para poder solucionarlo, saber exactamente en que consiste, descomponerlo en cada una de sus partes. Esta es una regla que siempre deberá ser aplicada se utilice o no una computadora en la solución de un problema.

Una vez entendido el problema, se está en condiciones de estudiar y plantear soluciones alternativas que permiten llegar al resultado. Finalmente, se selecciona la mejor alternativa.

Pasos:

* Conocer y definir el resultado deseado.
* Determinar los datos que se deben ingresar o generar para conseguir el resultado.
* Determinar la forma en que los datos serán manipulados o procesados para transformar los datos en información.

**Etapa 03 – Diseño de la lógica:** Definida la solución, se procede a diseñar la lógica utilizando UML y Pseudocódigo o diagrama de flujo. Una vez elaborado el DF o Pseudocódigo de la solución al problema (algoritmo) es necesario verificar si se han incluido todas las alternativas que puedan presentarse en la realidad. A este tipo de prueba se le denomina “*Prueba de escritorio*”.

**Etapa 04 – Desarrollo de la codificación:** Una vez diseñada la lógica para todas las alternativas que pueden presentarse en la realidad, se podrá proceder al desarrollo de la codificación del problema en algún lenguaje de programación. La codificación involucra traducir las especificaciones de la *Clase* (expresado en notación UML) y los pasos del Algoritmo de cada método (expresado en DF o Pseudocódigo), en sentencias de un lenguaje de programación determinado. Estas sentencias son almacenadas en un archivo y son las instrucciones que, posteriormente, la computadora podrá ejecutar.

**Etapa 05 – Depuración y pruebas:** Luego que se desarrolle la codificación de los programas, deberán probar mediante la ejecución de los mismos (esto es conocido como corrida del programa). Al realizarse esta prueba surgen generalmente errores de diferente tipo (principalmente los errores de lógica y sintaxis). Hay que corregir el programa; anular, modificar o crear nuevas instrucciones, y volver a probar el programa hasta conseguir el resultado deseado.

**Etapa 06 – Documentación:** Una buena documentación ahora ayuda a desarrollar buenos hábitos de programación que se necesitaran después. La documentación forma parte del proceso e programación y debe incluir, como mínimo:

* Descripción del problema.
* Resultados esperados y datos necesarios para generar dichos resultados.
* Diagramas UML y, DF o Pseudocódigo.
* Pruebas desarrolladas.
* Listado de programas con comentarios internos.

**Tipos de Datos que se Utilizan en el Desarrollo de Programas**

Un tipo de dato indica un conjunto de valores que significan lo mismo, hay algunos tipos que no representan valores en la aplicación que se ejecutan. Un tipo es una caracterización de propiedades estructurales o de comportamiento que utilizan una serie de entidades.

Tengamos en cuenta que la buena comprobación de tipos nos impide que se mezclen abstracciones. Existen 3 maneras de comprobar los tipos: **estático, dinámico, estricto**, este último casi siempre suele tomarse como tipo estático.

## ****Estático:**** Consiste en qué el tipo exacto de cada expresión pueda ser localizado en tiempo de compilación mediante un análisis estático de la aplicación. El tipo estático detecta anomalías en tiempo de compilación, pero puede ser muy restrictivo. Entre los lenguajes que utilización tipado estático podemos mencionar, Java o C++.

***E****stos permiten que los errores sean detectados antes de la ejecución, haciendo así la aplicación más eficiente.*

**Estricto:** Todas las expresiones de los tipos deben de ser consistentes en tiempo de compilación. Dejando más claro los tipos de datos estrictos aseguran que no se asignen accidentalmente un tipo de valor incorrecto o una variable, este tipo de datos también asegura que no se acceda a propiedades o métodos que no formen parte de dicho tipo de objeto.

La consistencia es la cualidad que tiene un objeto que resiste sin corromperse fácilmente. La consistencia se define a través de tres restricciones fundamentales.

* **Restricción de declaración**: Indica que todos las entidades deben tener un tipo declarado.
* **Restricción de Compatibilidad:** El tipo fuente debe ser compatible con el tipo de destino.
* **Restricción de llamada a característica:** Para poder llamar a un atributo o clase X desde la clase Y, X tiene que estar definida en y en sus antecesores.

***Pesimismo*** Se llama así cuando se tienen operaciones con tipos las cuales creemos estar seguros que funcionaran o serán válidas siempre, pero si no se tiene dicha seguridad, entonces es mejor que no se permitan.

Un ejemplo de Pesimismo sería, una Clase ***Animal***, una clase ***perro***que tiene le método correr.

Qué sucedería si yo hago lo siguiente:

Animal UnAnimal = agregue un perro

Luego tomo la variable **UnAnimal** y hago uso del método ***correr***(***UnAnimal.correr***)

Esto funcionaría si todos los Animales corrieran ¿pero qué pasa si yo en ***UnAnimal*** agrego un perico?

Para este problema tenemos un par de soluciones una de ellas llamada el **Type-Cast o Run-Time Type Information**.

## ****Dinámico:**** Se realizan las comprobaciones en tiempo real (ejecución). Esto quiere decir que una variable puede tomar valores de diferentes tipos en diferentes momentos.

Entre los lenguajes que podemos mencionar que utilizan este tipado está **Phyton** y **PHP**.

Ahora bien vamos a ver dos términos que suelen confundir algunas veces, **débilmente tipado y fuertemente tipado**.

**¿A qué se refiere el término fuertemente tipado?**

Esto nos exige tratar un tipo de dato solamente como ese mismo tipo, no permite conversiones incluidas, las conversiones de datos son explicitas.

Un lenguaje fuertemente tipado no permite tratar un tipo de dato como otro, no permiten violaciones de los tipos de datos, es decir, un tipo concreto no se puede usar como si fuera un tipo diferente a menos que se haga una conversión.

La mayoría de estos lenguajes realiza la conversión implícitamente.

**¿A qué se refiere el término débilmente tipado?**

Se refiere a las conversiones incluidas que se realizan, por ejemplo si se quiere convertir un real a entero, podría redondear el decimal a entero, quitar los decimales o mostrar solo el entero, esto depende de cómo trate la conversión el lenguaje débilmente tipado que se esté utilizando.

El tipo de dato no se define sobre la variable, si no que se define sobre el valor, debido a esto se puede usar variables de cualquier tipo en un mismo escenario.

Gracias a esto podemos mencionar que Phyton es un lenguaje de tipo dinámico (porqué no es necesario especificar el tipo de dato) y débilmente tipado (porqué una variable puede tomar distintos tipos de datos.

**Definición**

**Variables – Expresiones – Constantes**

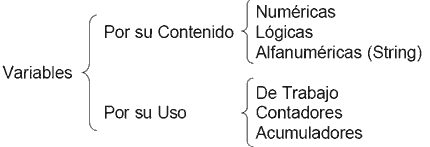
**Variable:** Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de un proceso, su contenido puede cambiar durante la ejecución del programa.

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

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

Ejemplo:

pi = 3.1416

**Formatos Para Sentencias de**

**Asignación – Entrada y Salida**

La asignación de contenido a variables se hace en Java de la siguiente manera:

variable = expresión;

Por ejemplo precio = precioBase + 0.35; En toda asignación debe haber coincidencia de tipos. Si peso es una variable de tipo int y sobrepeso es una variable de tipo double ó boolean debemos evitar sentencias como sobrepeso = 2 \* peso; Aunque hay ocasiones en que la falta de coincidencia de tipos será asumida por Java, debemos tratar de evitarlo en todo caso porque introduce inseguridad en la programación. En Java una sentencia como saldo = saldo + cantidad; se puede escribir también así: saldo += cantidad;

En programación se denominan sentencias de entrada a aquellas que permiten enviar información desde el exterior al interior de un programa y sentencias de salida a aquellas que permiten enviar información desde el interior del programa al exterior.

En los lengua­jes de progra­mación existe una gran variedad de sen­tencias de entra­da y de salida. Normalmente estas sentencias se escriben de forma distinta en función de la información, por ejemplo se escribiría de una forma si la información de entrada es numérica y de otra si la información de entrada son caracteres. También se pueden escribir las sentencias de entrada y de salida de distinta forma en función de donde reciban la información o donde la envíen (a la pantalla, a la impresora, al disco duro, a la disquetera, al diskette, a otro ordenador, a un dispositivo de control remoto, etc.).

Las sentencias de entrada y de salida se las conoce como sentencias de E/S (Entrada/Salida) o sentencias de I/O

(Input/Output cuya traducción sería input = entrada y output = salida).

**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.

**JAVA**

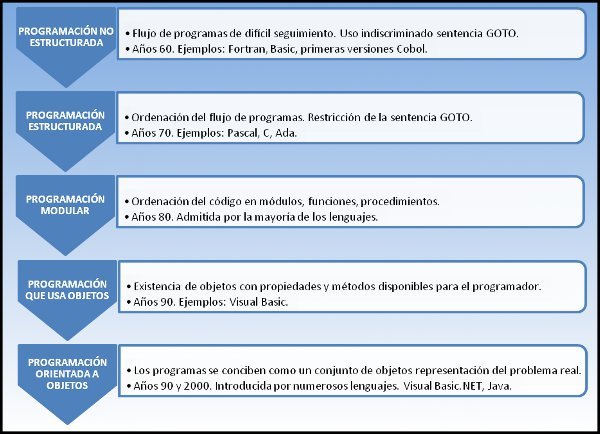
Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

Los antecedentes de Java habría que buscarlos en los lenguajes de programación C y C++. El lenguaje C fue desarrollado en la década de los 70 y constituye un lenguaje muy robusto usado como núcleo del sistema operativo Unix. C no admite la orientación a objetos y está considerado un lenguaje “poco amigable” desde el punto de vista de que su sintaxis, elementos de programación que emplea (manejo directo de memoria) y otras cuestiones hacen que sea un lenguaje difícil de aprender. C++ fue una evolución de C desarrollada en los años 80. Introdujo el diseño orientado a objetos, pero manteniendo la compatibilidad con C. Esto suponía que C++ permitiera tanto el uso de la programación estructurada “tradicional” como la programación orientada a objetos. Además C++ mantuvo ciertas características de C como el manejo directo de la memoria, el uso de variables globales, sentencia goto, etc. que hicieron que la evolución fuera “parcial”.

Como paso final en esta evolución tenemos Java, un lenguaje que evoluciona a partir de C y C++, pero que elimina diversos aspectos de estos lenguajes y se constituye en un lenguaje definitivamente orientado a objetos. El romper con distintos aspectos de C++ cuyo manejo inadecuado por parte de muchos programadores daba lugar a problemas en las aplicaciones ha sido un factor decisivo para convertir a Java en un lenguaje popular y de amplio uso.

Nosotros vamos a quedarnos con el lado práctico de lo que hemos comentado respondiendo esta pregunta: **¿Es necesario saber C ó C++ para programar en Java?** No, no es necesario. Aunque puede suponer una ventaja para aquellas personas que tengan conocimientos previos en estos lenguajes, no recomendamos de forma explícita su estudio en profundidad como paso previo al aprendizaje de Java.

De modo orientativo, veamos un esquema sobre la evolución de los lenguajes.



 Este esquema es meramente orientativo: es imposible reflejar la diversidad de lenguajes y su evolución en un gráfico tan simplificado. También los datos relativos a fechas son orientativos.

No hay que suponer que lo único válido sea la programación orientada a objetos por ser lo más moderno. Al contrario, muchísima programación de la que se hace hoy en día se basa en lenguajes o código no orientado a objetos. Además, la misma programación orientada a objetos se basa en conceptos muy antiguos de programación.

Tener en cuenta que algunos lenguajes que nacieron en los años 60 han perdido vigencia y ya no se usan, mientras que otros se han ido modernizando y continúan usándose más o menos ampliamente, como es el caso de Cobol. Lo que consideramos interesante con este esquema es que se vea que Java es una evolución que por un lado incorpora cosas que se venían usando desde hace mucho tiempo en programación, y por otro introduce ciertas novedades que lo convierten en un lenguaje moderno.

*Java es un lenguaje útil para casi todo tipo de problemas. Podemos citar como funcionalidades de Java varias:*

 1**. Aplicaciones “cliente”**: son las que se ejecutan en un solo ordenador (por ejemplo el portátil de tu casa) sin necesidad de conectarse a otra máquina. Pueden servirte por ejemplo para realizar cálculos o gestionar datos.

2**. Aplicaciones “cliente/servidor”**: son programas que necesitan conectarse a otra máquina (por ejemplo un servidor de datos) para pedirle algún servicio de forma más o menos continua, como podría ser el uso de una base de datos. Pueden servir por ejemplo para el teletrabajo: trabajar desde casa pero conectados a un ordenador de una empresa.

3. **Podemos hablar también de “aplicaciones web”**, que son programas Java que se ejecutan en un servidor de páginas web. Estas aplicaciones reciben “solicitudes” desde un ordenador y envían al navegador (Internet Explorer, Firefox, Safari, etc.) que actúa como su cliente páginas de respuesta en HTML.

 Éstos son sólo algunos ejemplos de todo el potencial que hay detrás de Java como lenguaje para aprender y obtener muchos beneficios con su uso. Obviamente por determinados términos empleados (cliente, cliente/servidor, base de datos, HTML…), te darás cuenta de que el lenguaje Java tiene mucha potencialidad, pero también de que su conocimiento a fondo requeriría mucho tiempo. Nosotros en este curso vamos a estudiar únicamente los aspectos más básicos de Java.

# Bibliografía

*San Francisco Xavier a la par con la tecnologia*. (2008). Recuperado el 10 de Febrero de 2017, de http://sis324loo.blogspot.com.co/2008/09/historia-de-los-lenguajes-de\_29.html

*pasossoluciondeproblemas*. (15 de Enero de 2012). Recuperado el 07 de Febrero de 2017, de http://pasossoluciondeproblemas.blogspot.com.co/2012/01/metodologia-para-la-solucion-de-pasos.html

*algoritmiafordummies*. (s.f.). Recuperado el 11 de Febrero de 2017, de https://algoritmiafordummies.wikispaces.com/4.+Sentencias+de+entrada+y+salida

Cueto, J. J. (2003). *Google Books*. Recuperado el 10 de Febrero de 2017, de Metodo para la solucion de problemas utilizando la Programacion Orientada a Objetos: https://books.google.com.co/books?id=SwKI-CMJUS0C&pg=PA15&lpg=PA15&dq=resolucion+de+problemas+programaci%C3%B3n+orientada+a+objetos&source=bl&ots=D9TMsHxgig&sig=BbsKkiKBit32RxYybyDCPSJ05JA&hl=es-419&sa=X&redir\_esc=y#v=onepage&q=resolucion%20de%20problemas

*ecured*. (s.f.). Recuperado el 09 de Febrero de 2017, de https://www.ecured.cu/Programa\_inform%C3%A1tico

*funprogramacion*. (s.f.). Recuperado el 09 de Febrero de 2017, de https://funprogramacion.wikispaces.com/Compiladores+e+Int%C3%A9rpretes

Henao, C. (19 de Mayo de 2013). *codejavu*. Recuperado el 10 de Febrero de 2017, de http://codejavu.blogspot.com.co/2013/05/conceptos-de-programacion-orientada.html

Krall, C. (s.f.). *aprenderaprogramar*. Recuperado el 11 de Febrero de 2017, de http://aprenderaprogramar.com/index.php?option=com\_content&view=article&id=688:ique-es-y-para-que-sirve-uml-versiones-de-uml-lenguaje-unificado-de-modelado-tipos-de-diagramas-uml&catid=46:lenguajes-y-entornos&Itemid=163

Lara, D. (17 de Julio de 2015). *Styde*. Recuperado el 11 de Febrero de 2017, de Tipos de Datos POO: https://styde.net/tipos-de-datos-programacion-orientada-a-objetos/

oracle. (s.f.). *java*. Recuperado el 11 de Febrero de 2017, de https://www.java.com/es/download/faq/whatis\_java.xml

Rodríguez, A. (2006). *aprenderaprogramar*. Recuperado el 11 de Febrero de 2017, de http://www.aprenderaprogramar.com/index.php?option=com\_attachments&task=download&id=425

Rodríguez, A. (s.f.). *aprenderaprogramar*. Recuperado el 11 de Febrero de 2017, de http://aprenderaprogramar.com/index.php?option=com\_content&view=article&id=368:ique-es-java-concepto-de-programacion-orientada-a-objetos-vs-programacion-estructurada-cu00603b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

*upiicsa*. (s.f.). Recuperado el 07 de Febrero de 2017, de http://www.sites.upiicsa.ipn.mx/polilibros/portal/polilibros/P\_terminados/PolilibroFC/Unidad\_III/Unidad%20III\_4.htm

Urbaez, W. (20 de Septiembre de 2005). *desarrolloweb*. Recuperado el 10 de Febrero de 2017, de https://desarrolloweb.com/articulos/2164.php