GUÍA DE TRABAJO # 2
ALGORITMOS SECUENCIALES

1. IDENTIFICACIÓN DE LA ASIGNATURA

Asignatura:	Lógica de Programación y Laboratorio						
Código:	000506001			**Área:**	Básicas de la ingeniería		
Plan de Estudios:	1			**Semestre:**	1		
Créditos: 5	**TPS: 6**		**TIS: 9**	**TPT: 96**	**TIT: 144**		
Trabajo Independiente:				**Trabajo Presencial:**			
Trabajo Teórico		Trabajo Práctico	X	Trabajo Teórico		Trabajo Práctico	X

2. TEMA Y COMPETENCIA:

Contenido temático:	Algoritmos secuenciales: una definición desde el planteamiento de clases en programación orientada por objetos
Competencia por desarrollar:	Genera soluciones algorítmicas, mediante el análisis lógico, basado en el uso de estructuras estáticas de datos y en el manejo deductivo de cada una de ellas, para implementarlas en un lenguaje de programación de alto nivel.
Resultado de aprendizaje:	Diseño de una solución algorítmica fundamentada en el paradigma objetual.

3. RECURSOS REQUERIDOS

- Equipo de cómputo con conexión a Internet
- Procesador de textos como Word o Notepad++

4. MARCO TEÓRICO

4.1 ETAPA DE DISEÑO DE UNA SOLUCIÓN

Como se ha mencionado, la primera etapa en el proceso de desarrollo de software es el Análisis del Problema. En esta etapa se define cuáles son las funcionalidades que debe tener el programa y se identifican cuáles son las entidades del mundo que participan en la solución del problema. Es decir, en el análisis del problema debemos respondemos *qué* va a hacer el programa.

Cuando pasamos a la etapa de Diseño de la Solución, esta define el *cómo* el programa realizará las funcionalidades que se han identificado. Es decir, en la etapa de diseño describimos de manera textual o gráfica la manera como se implementarán las funcionalidades del programa para solucionar el problema. El primer elemento de la etapa de diseño es el *Diagrama de Clases*, el cual muestra cuáles son las entidades del mundo del problema, sus atributos y las relaciones que hay entre esas entidades.

El siguiente paso en la etapa de diseño consiste en la construcción de los algoritmos de las clases para solucionar el problema, pero ¿qué es un algoritmo?

> **Para Recordar …**
>
> Un *Algoritmo* es una secuencia de pasos ordenados, bien definidos y finitos que se deben seguir para solucionar un problema específico o para para alcanzar un objetivo previamente establecido.
>
> 
>
> En esta definición, bien definido hace referencia a que si cada vez que se sigue el algoritmo se obtiene exactamente el mismo resultado, independiente de la persona que lo siga. Por otro lado, la palabra finito hace referencia a que el algoritmo debe tener un fin, es decir se debe finalizar en un número determinado de pasos.

Con base en la definición anterior, los algoritmos son una herramienta para describir de manera textual, e independiente de un lenguaje de programación, la secuencia de pasos que debe ejecutar el computador para solucionar un problema.

4.2 Formas para Representar un Algoritmo

Existen diferentes formas de representar un algoritmo. La más simple es el *Lenguaje Natural*. Esta forma de representación es ampliamente utilizada en algoritmos para solucionar problemas cotidianos y cualitativos; es decir, problemas en los que la solución se puede definir usando el lenguaje que usamos para comunicarnos día a día. Un ejemplo de este tipo de algoritmos son las recetas de cocina. Veamos un posible algoritmo para preparar una libra de arroz:

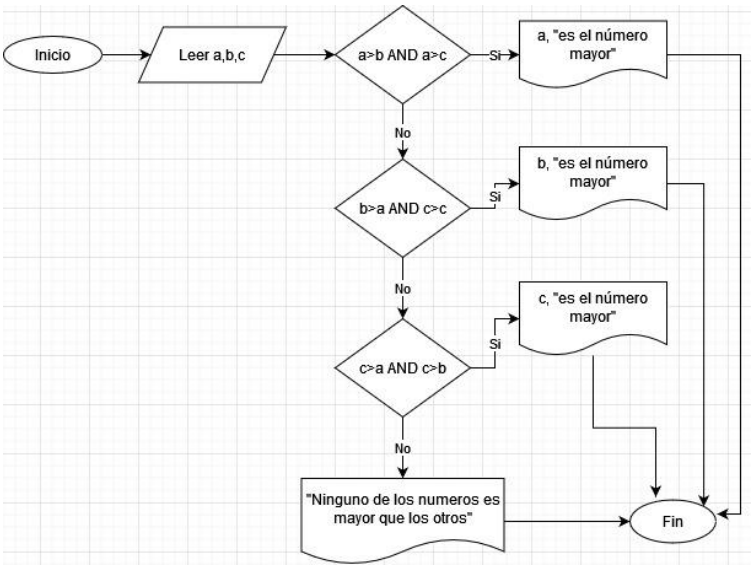
```
Ingredientes:
   Una libra de arroz
   Cuatro tazas de agua
   Una cebolla picada
   Una cucharada de aceite
   Una cucharadita de sal
   Una olla mediana con tapa
Inicio
   1. Ponga la olla a fuego medio en la estufa
   2. Agregue el aceite a la olla y espere un momento a que caliente
   3. Agregue la cebolla y sofríala por un minuto
   4. Agregue el arroz, revuélvalo y sofría por un minuto
   5. Agregue la sal, revuelva y suba fuego a alto
   6. Cuando seque por completo el agua, baje el fuego a bajo, tape la olla
   7. Trascurridos 15 minutos apague la estufa y deje reposar el arroz
Fin
```

A pesar de su simplicidad, el lenguaje natural no es la mejor opción para escribir algoritmos porque tiende a ser ambiguo y a estar definido vagamente, esto sucede porque este lenguaje carece de una estructura precisa y fija para representar un concepto siempre de la misma manera. Por esta razón los algoritmos en lenguaje natural no suelen usarse para solucionar problemas que conllevan a la implementación un programa de computador.

Otra forma de representación de algoritmos son los *Diagramas de Flujo*. Estos tienen una estructura más formal (son cajas conectadas por flechas) que limita la interpretación de las instrucciones que definen la secuencia de pasos para la solución de un problema. Los diagramas de flujo pueden usarse tanto para algoritmos cualitativos (como la receta de cocina) como también para algoritmos cuantitativos, es decir algoritmos cuyos pasos requieren hacer cálculos matemáticos. Entre las ventajas de los diagramas de flujo está su facilidad de comprensión, esto porque que el cerebro humano interpreta más fácilmente una representación gráfica que una representación textual.

Por otro lado, una de las desventajas de los diagramas de flujo es que su facilidad de interpretación disminuye a medida que aumenta la complejidad del problema, esto porque el diagrama crece considerablemente cuando el problema se

hace más complejo. Esta es la principal razón por la cual los diagramas de flujo no suelen usarse para solucionar problemas de complejidad media que conllevan a la implementación un programa de computador. Veamos un diagrama de flujo que ayuda a resolver el problema para determinar cuál es el mayor de tres números:



Finalmente, otra forma para representar un algoritmo es el Seudocódigo. Esta forma de representar un algoritmo usa una sintaxis simple que facilita la traducción de las instrucciones del algoritmo a un lenguaje de programación. Al ser un seudo lenguaje, el seudocódigo no puede ejecutarse o ser interpretado por una máquina, pero es completamente comprensible para los humanos.

Como veremos, el seudocódigo tiene unas palabras reservadas en español (o inglés), las cuales, a pesar de no estar estandarizadas, siguen una notación comprensible para todos los desarrolladores. Vemos un fragmento de seudocódigo para dar solución al problema de determinar cuál es el mayor de tres números:

```
Entero: a, b, c

Escribir ("Ingrese tres números: ")
Leer (a, b, c)
Si (a>b AND a>c) Entonces
  Escribir(a, "es el número mayor")
Sino
  Si (b>a AND b>c) Entonces
    Escribir(a, "es el número mayor")
  Sino
    Si (c>a AND c>b) Entonces
      Escribir(a, "es el número mayor")
    Sino
      Escribir(a, "Ninguno de los números es mayor que los otros")
    Fin_Si
  Fin_Si
Fin_Si
```

4.3 Instrucciones y Estructura de un Algoritmo en Seudocódigo

El pseudocódigo tiene algunas reglas de sintaxis y de gramática básicas que están definidas sobre diversos elementos del seudo lenguaje. Estos elementos son los comentarios, variables y constantes, operadores, expresiones y las estructuras de control. El uso de estos elementos en una estructura definida es lo que nos permite escribir algoritmos que puedan ser leídos e interpretados por el equipo de desarrollo. Veamos la sintaxis y gramática de estos elementos en del seudocódigo.

4.3.1 COMENTARIOS

Los comentarios son un elemento que nos permite explicar el funcionamiento de bloques de instrucciones de un algoritmo o programa. En el sentido estricto, los comentarios no son instrucciones y por tanto la máquina los ignora cuando las encuentra.

Dependiendo del lenguaje, pueden existir dos tipos de comentarios. En nuestro caso utilizaremos dos tipos de comentarios:

- **Comentarios de una línea:** se utilizan para explicar de manera muy corta una instrucción en el algoritmo. Un comentario de una sola línea inicia con el símbolo **//** y finaliza al terminar la línea donde se inició.

```
// Esto es un comentario, esta línea será ignorada
```

- **Comentarios de varias líneas:** se utilizan para explicar el funcionamiento de un bloque del algoritmo. Un comentario de múltiples líneas inicia con el símbolo y termina cuando se encuentra el símbolo ***/**.

```
/* Este es un comentario de varias líneas,
     es decir, todo lo que está entre estos dos símbolos es ignorado. */
```

4.3.2 VARIABLES Y CONSTANTES

Todo dato que se utiliza en un programa debe ser almacenado en la memoria RAM del computador. Esa memoria está dividida en "compartimientos" del mismo tamaño en los que se almacena información. El tamaño de esos compartimentos está definido en bits, lo que indica cuál es el valor máximo que se puede almacenar en ellos. También, cada compartimiento tiene asociado una dirección única que permite a un programa (incluido el sistema operativo) acceder a la información almacenada bien sea para consultarla, modificarla o borrarla. Esas direcciones de memoria están representadas normalmente por un código hexadecimal, como se ejemplifica a continuación.

Contenido de la RAM	Dirección Física
10000010	0x00A01
00110011	0x00A02
01110111	0x00A03

Por la complejidad inherente de manejar las direcciones de la memoria RAM, el programador usa en su lugar identificadores para almacenar los datos del programa en la memoria RAM sin preocuparse de las direcciones en hexadecimal. Así, cuando hablamos de una *variable* nos referimos a un identificador que nos permite almacenar un valor en la memoria RAM, el cual que puede cambiar durante la ejecución del algoritmo o programa. Por otro lado, cuando hablamos de una *constante* nos referimos a un identificador que nos permite almacenar en la RAM un valor que permanece constante durante la ejecución del algoritmo o programa.

Importante:

El identificador de una *variable* debe cumplir lo siguiente:

- Comienzan con una letra en minúscula o el carácter raya abajo (_)
- Hay distinción entre mayúsculas y minúsculas. Es decir, `Suma`, `suma` y `SUMA` no son el mismo identificador
- No están permitidos espacios entre caracteres y otros caracteres especiales

El identificador de una *constante* debe cumplir lo siguiente:

- Todos los caracteres deben escribirse en mayúscula
- Se usa el carácter raya abajo (_) para separar palabras

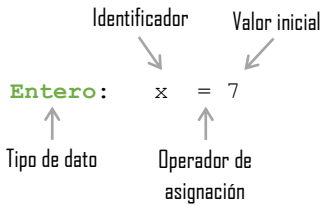
Antes de usar una variable o una constante debemos declararla, es decir, antes de usarlas debemos indicarle a la máquina que separe un espacio en memoria para almacenar un valor en estas. El espacio que se le asigna a una variable o constante depende de su tipo de dato. Veamos cuales son los tipos de datos comúnmente usados en los algoritmos

- **Tipos de datos numéricos:** existen dos tipos de datos numéricos: enteros y reales.
  - *Entero*: una variable declarada como Entero sólo puede almacenar números enteros, es decir, números que no tienen decimales. Este tipo de variables suele ocupar 32 bits de la memoria RAM.
  - *Real*: una variable declarada como Real puede almacenar número con coma flotante, incluyendo números enteros. Este tipo de variables tiene un rango mayor que las variables de tipo Entero puesto que suelen ocupar 64 bits.
- **Tipo de dato** *Lógico* **o** *Booleano*: las variables y constantes con este tipo de dato suelen ocupar 1 bit en la memoria RAM puesto que son variables que toman uno de dos valores: `true` o `false`.
- **Tipo de dato** *Carácter*: las variables y constantes con este tipo de dato suelen ocupar 8 bits en la memoria RAM. Un carácter es cualquier número, letra o símbolo. Los valores de tipo Carácter se encierran en comillas simples, por ejemplo: 'A', '3', '#'.
- **Tipo de dato** *Cadena* **o** *Texto*: las variables y constantes de tipo Cadena no tienen un tamaño fijo definido en la memoria RAM, esto porque tienen una cantidad indeterminada de caracteres. Los valores de tipo Cadena se encierran en comillas dobles, por ejemplo: "Esto es una cadena", "Hola", "El resultado es: ".

Una vez identificados los tipos de datos, podemos definir una forma general para declarar variables y constantes en pseudocódigo:

```
Tipo_de_dato: identificador = valor_inicial
```

Un ejemplo particular de esta forma general es la declaración de una variable entera a la que llamaremos `x` y que tiene valor inicial `7`:



4.3.3 OPERADORES

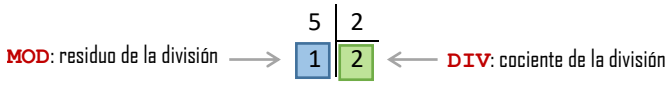
Un algoritmo se construye a partir de expresiones y las expresiones están definidas sobre operandos y operadores. En un algoritmo existen 5 tipos de operadores: aritméticos, relacionales, lógicos, de asignación y de entrada/salida.

- Los *Operadores Aritméticos:* se usan para construir expresiones aritméticas que operan datos de tipo numérico. El resultado de estos operadores también es de tipo numérico. En seudocódigo se usan los siguientes operadores aritméticos:

Operador	Símbolo	Ejemplo	Resultado
Potencia	`**`, `^`	`2 ^ 3`	8
Multiplicación	`*`	`4 * 3`	12
División	`/`	`5 / 2`	2.5
División entera	`DIV`	`5 DIV 2`	2
Residuo	`MOD`	`5 MOD 2`	1

Suma	+	6 + 4	10
Resta	-	3 - 15	-12

Nota: los operadores MOD y DIV se usan para calcular el residuo y el cociente de una división entera. Por otro lado, es importante que tenga presente que el operador / calcula una división de tipo real, es decir que su resultado puede ser un número con decimales. Por el contrario, el operador DIV calcula una división entera y en consecuencia su resultado siempre será un número entero. Veamos cómo se calcula el DIV y el MOD:



Los operadores aritméticos tienen un orden de aplicación (o *precedencia*):

- Primero se resuelven las operaciones en paréntesis
- Después se resuelven las potencias
- A continuación, se resuelven las multiplicaciones y divisiones en orden de aparición de izquierda a derecha
- Finalmente se resuelven las sumas y las restas

Nota: el módulo y el cociente tienen la misma precedencia de las multiplicaciones y divisiones.

- Los *Operadores Relacionales* nos permiten comparar dos valores, considerando que el resultado es un valor lógico (true o false). En seudocódigo los operadores relacionales son:

Operador	Símbolo	Ejemplo	Resultado
Igual que: ¿Son iguales los valores?	==	"Hola"=="hola"	false
Diferente: ¿Son distintos los valores?	<>, !=	7 <> -7.5	true
Menor que: ¿Es un valor menor que el otro?	<	4 < -1	false
Mayor que: ¿Es un valor mayor que el otro?	>	4 > -1	true
Menor o igual que: ¿Es un valor menor o igual que el otro?	<=	6 <= 6	true
Mayor o igual que: ¿Es un valor mayor o igual que el otro?	>=	6 >= 15	false

Nota: los operadores igual == y != nos permiten comparar bien sea valores numéricos entre sí, cadenas de texto entre sí o valores lógicos entre sí. Por otro lado, los operadores <, >, <= y >= sólo se usan para comparar valores numéricos.

El orden de aplicación de los operadores relacionales es el siguiente:

- Primero se evalúan los paréntesis
- Después se evalúan los operadores: <, >, <= y >=
- Después se evalúan los operadores: == y !=

- Los *Operadores Lógicos* se utilizan para construir expresiones lógicas cuyos operandos son valores lógicos. Como resultado, este tipo de operadores genera un valor lógico. En seudocódigo se usan los tres operadores lógicos siguientes:

Operador	Símbolo	Ejemplo	Resultado
Negación	not, !	not true	false
Conjunción	and, &&	true and false	false
Disyunción	or, ||	true or true	true

El orden de precedencia de estos operadores es el siguiente:

- Primero se resuelven las operaciones en paréntesis
- Después se resuelven las negaciones (`not`)
- A continuación, se evalúa el operador `and`
- Por último, se evalúa el operador `or`

Cómo una expresión puede contener al mismo tiempo operadores de cualquiera de los tipos, se debe considerar la precedencia de todos los operadores: operaciones entre paréntesis, operadores aritméticos, operadores relacionales y al final los operadores lógicos. Esta precedencia se resume en la siguiente tabla.

Operador	Tipo
( )	Paréntesis
^	Potenciación
* / DIV MOD	Multiplicativos
+ -	Aditivos
< <= > >=	Relacional
== !=	Igualdad
not	Lógico
and	Lógico
or	Lógico
= += -= *= /= %=	Asignación

- Los *Operadores de Lectura y Escritura* son los operadores que permiten a un programa interactuar con los usuarios que lo utilizan. El operador `Escribir` (también conocido como Mostrar o Imprimir) se utiliza para mostrar un mensaje en la pantalla del computador. Por otro lado, el operador `Leer` permite a los usuarios ingresar valores que serán almacenados en una variable. Veamos un ejemplo, el siguiente fragmento de seudocódigo pide al usuario su nombre, lo almacena en una variable llamada `nombre` y posteriormente lo saluda:

```
Cadena: nombre

Escribir("Cuál es tu nombre?")
Leer(nombre)

Escribir("Hola ", nombre)
Escribir("Espero que tengas un fantástico día")
```

En la instrucción `Escribir`, el texto entre las comillas dobles (" ") es el texto que se muestra al usuario, tal cual como lo ponemos en las comillas. Por otro lado, el texto por fuera de las comillas puede ser una variable o una operación encerrada en paréntesis y unida al texto usando una coma (**,**), la cual va por fuera de las comillas.

- Finalmente, están los *Operadores de Asignación*, los cuales permiten guardar (o reemplazar) un valor de una variable. El operador de asignación se representa por el símbolo igual (=) e indica que en la variable que está a la izquierda del operador, se le asigna el valor que está a la derecha de este.

Veamos un ejemplo, creamos tres variables enteras: x, y e z. Los valores de x e y se los pedimos al usuario. Posteriormente en la variable z ASIGNAMOS el resultado de sumar x e y. Finalmente mostramos el resultado de la suma en la pantalla:

```
//Se declaran tres variables enteras
Entero: x, y, z

//Se piden dos valores al usuario y se almacenan en las variables x e y
Escribir ("Ingrese 2 números enteros: ")
Leer (x, y)

//Se suman los dos valores y el resultados se le ASIGNA a la variable z
z = x + y

//Se muestra el resultado al usuario
Escribir ("El resultado de la suma es: ", z)
```

Esta es la instrucción que almacena en z el resultado de sumar x e y

Nota: Tenga presente que una instrucción que tenga la forma `x+y=z` **NO ES CORRECTA** en un algoritmo, esto porque el operador de asignación siempre espera que a la izquierda del mismo haya una sola variable y no una operación como es este caso.

4.3.4 EXPRESIONES

Una expresión se compone de operadores y operandos. Como vimos los operadores nos permiten realizar operaciones aritméticas entre los operandos, los cuales pueden valores específicos, variables o constantes. En general, existen dos tipos de expresiones las aritméticas y las booleanas (o lógicas). Las expresiones aritméticas son aquellas que tras su ejecución el resultado es un valor numérico, mientras que el resultado de las expresiones booleanas es un valor lógico.

Veamos un ejemplo, cuando transformamos una fórmula matemática obtenemos una expresión aritmética. Un caso particular es la transformación de la ecuación $x = \frac{-b+\sqrt{b^2-4ac}}{2a}$. Al transformar esta fórmula matemática en una expresión aritmética obtenemos:

```
x = ((-1*b)+(b^2-4*a*c)^(1/2))/(2*a)
```

Un ejemplo que combina los dos tipos de expresiones es el siguiente:

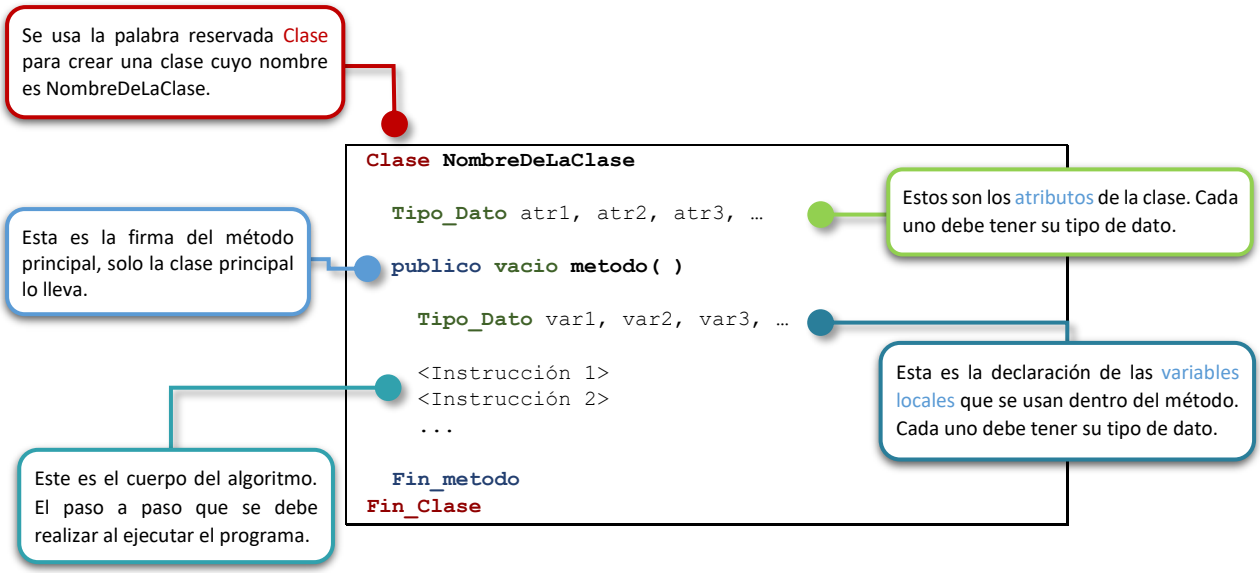
```
Entero: x=25, y=13, z
Lógico: p, q

z = x + y DIV 2 * 3 - (23 MOD 20 + 5 ^ 2)
p = x > z OR y + 20 < z
q = p AND x MOD 7 < y - 2
```

En este ejemplo, el cálculo de `z` es una expresión aritmética cuyo resultado es `25`, mientras que el cálculo de `p` y `q` cuyos valores es `false`, obedecen al resultado de expresiones booleanas.

4.3.5 ESTRUCTURA DE UN ALGORITMO EN SEUDOCÓDIGO

En este curso usamos una estructura general que está estrechamente relacionada con la programación orientada por objetos, miremos esa estructura:



- Todo algoritmo debe ser creado en una o varias clases. Para definir una clase se utiliza la palabra reservada `clase`, seguida del nombre que le daremos a esta. Ese nombre sigue una regla básica: la primera letra de cada palabra inicia en mayúscula: **N**ombre**D**e**L**a**C**lase.
- Si la clase tiene atributos, estos deben ser declarados debajo del nombre de la clase. Como los atributos son variables, estos tienen que definirse indicando su tipo de dato (`Tipo_Dato`) el cual determina el tipo de valores que podrá tomar el atributo. El identificador de cada atributo debe seguir la siguiente regla: la primera palabra inicia con minúscula, pero las demás palabras inician con mayúscula, por ejemplo: **e**ste**E**s**U**n**A**tributo.
- Posterior a los atributos están los métodos, de la clase en los cuales se escribe el algoritmo que un objeto de la clase cuando se invoca el método. Todo método tiene una marca de visibilidad (`publico`, `privado` o `protegido`) seguido del tipo de datos que retorna el método (de momento `vacio` para indicar que el método no retorna nada a quien lo invoca), después sigue el nombre del método, el cual debe iniciar con un verbo en infinitivo en minúscula. Las siguientes palabras que complementan el verbo inician en mayúscula, por ejemplo: **mostrarI**nforme**D**e**V**entas.
- Las instrucciones dentro del método definen el paso a paso que se ejecuta el invocarse el método, es decir en el método se escribe el algoritmo que da solución a la funcionalidad para la que se creó el método.

En Resumen …

El seudocódigo es un seudo lenguaje que tiene su propia sintaxis para la escritura de algoritmos. Un algoritmo puede tener:

- **Comentarios**, que son instrucciones que ayudan a explicar lo que hace el algoritmo
- **Variables y Constantes**, las cuales se usan para almacenar valores en la memoria RAM a través de un identificador. Variables y Constantes tienen asociado un tipo de dato que puede ser Entero, Real, Cadena, Lógico o Carácter. El operador de asignación se usa para asignar valores a las constantes y constantes.
- **Expresiones** construidas a partir de los **Operadores** para definir las instrucciones del algoritmo. Estos operadores tienen una precedencia: aritméticos, relacionales, lógicos y por último los de asignación.
- Todo algoritmo por desarrollar debe seguir una estructura basada en el desarrollo de clases.

5. PROCEDIMIENTO

Considere el siguiente enunciado a partir del cual se ejemplificarán los elementos antes descritos.

Ejemplo: Algoritmos Secuenciales en Seudocódigo

La empresa "***Huevos a Precio de Huevo***" comercializa 4 tipos de huevos: A, AA, AAA y Jumbo. El precio de cada tipo de huevo está definido sobre el valor de los huevos tipo AA. Haga un algoritmo que pida el valor de un huevo AA y con base en este indique el precio de cada tipo teniendo en cuenta las siguientes reglas:

- Los huevos A equivalen al valor de un huevo AA menos: la raíz cuadrada del precio de un huevo AA sumado con el resultado de multiplicar 3 por el módulo entre el precio de un huevo AA y 100.
- El precio de los huevos AAA equivale un huevo AA más la raíz tercera del precio de un huevo tipo A, más un cuarto del valor de un huevo AA, sumado con el resultado de la división entera entre el valor del huevo AA con 10.
- El precio de un huevo Jumbo equivale al precio de un huevo AAA más la raíz cuarta de un huevo AA elevado a la potencia 5. Al resultado se le suma cuatro quintos del valor de un huevo A.

5.1 IDENTIFICAR LOS REQUISITOS FUNCIONALES:

Al analizar este problema encontramos que existe un solo requerimiento funcional:

- Calcular el precio de los huevos A, AAA y Jumbo con base en el precio de los huevos AA.

Al expandir el requerimiento en el formato de descripción de requerimientos, tenemos.

Nombre	**RF-1. Calcular y mostrar el precio de los huevos A, AAA y Jumbo**
Resumen	A partir del precio de un huevo AA el sistema debe calcular el precio de los huevos A, AAA y Jumbo. Para ello se deben seguir las reglas definidas en la definición del problema
Entradas	El precio de un huevo AA
Resultados	Se muestra en la pantalla el valor de los huevos A, AA, AAA y Jumbo

5.2 DEFINIR LAS REGLAS DE NEGOCIO

Procedemos ahora a identificar las reglas de negocio para cada uno de los requerimientos.

Reglas Asociadas a:	**Descripción de la Regla de Negocio:**	**Tipo de Regla:**
Calcular y mostrar el precio de los huevos A, AAA y Jumbo	Para el calculo del precio de los huevos se parte del precio de un huevo AA, el cual debe ser un valor real positivo.	Estructura y dominio de datos
	La fórmula para calcular el precio de un huevo A es: huevoA = $\sqrt{huevoAA} + 3 * (huevoAA\ MOD\ 100)$	Cálculo
	La fórmula para calcular el precio de un huevo AAA es: HuevoAAA = $huevoAA + \sqrt[3]{huevoA} + \frac{1}{4}huevoAA + (huevoAA\ DIV\ 10)$	
	La fórmula para calcular el precio de un huevo Jumbo es: huevoJumbo = $huevoAAA + \sqrt[4]{huevoAA^5} + \frac{4}{5}huevoA$	

5.3 IDENTIFICAR LAS ENTIDADES DEL MUNDO DEL PROBLEMA

En este problema sencillo podemos usar una sola clase, la clase principal, a la cual denominaremos `PrecioDeHuevo`. Esta clase no tiene atributos y sólo tiene un método principal en el que desarrollaremos el algoritmo completo:

PrecioDeHuevo
principal()

5.4 SEUDOCÓDIGO DE LA CLASE

El paso siguiente en el diseño es el desarrollo de los algoritmos de la clase. En nuestro caso el algoritmo es sencillo y se presenta a continuación:

```
clase PrecioDeHuevo

  //Esta clase no tiene atributos

  publico vacio principal()
    Real: huevoA, huevoAA, huevoAAA, huevoJumbo

    Escribir("Ingrese el precio de los huevos tipo AA: ")
    Leer(huevosAA)

    //Calculamos el precio de los huevos tipo A
    huevoA = huevoAA^(1/2) + 3 * (huevoAA MOD 100)

    //Calculamos el precio de los huevos tipo AAA
    huevoAAA = huevoAA + huevoA^(1/3) + (1/4)*huevoAA + (huevoAA DIV 10)

    //Calculamos el precio de los huevos tipo Jumbo
    huevoJumbo = huevoAAA+ huevoAA^(5/4) + (4/5)*huevoA

    //Mostramos los precios de los huevos
    Escribir("Los huevos tipo A cuestan ", huevoA, " pesos")
    Escribir("Los huevos tipo AA cuestan ", huevoAA, " pesos")
    Escribir("Los huevos tipo AAA cuestan ", huevoAAA, " pesos")
    Escribir("Los huevos tipo Jumbo cuestan ", huevoJumbo, " pesos")

  Fin_metodo
Fin_clase
```

5.5 DESARROLLO EN PROYECTO INTEGRADOR

Ahora debes poner en práctica lo descrito en esta guía de trabajo. Para ello, selecciona uno de los casos del proyecto integrador con tu equipo de trabajo y hagan lo siguiente:

1. Definir y detallar los requisitos funcionales
2. Definir las reglas de negocio y clasifíquelas por su tipo
3. Identifique las entidades del mundo del problema y sus atributos y descríbalos
4. Construir el modelo de mundo del problema o diagrama de clases
5. Escriba los algoritmos en seudocódigo para las clases identificadas en el diagrama de clases

6. RÚBRICA DE EVALUACIÓN

Resultado de aprendizaje	Indicador de logro	Rango de valoración
Diseño de una solución algorítmica completa y su software, fundamentado en el paradigma	Identifica los requisitos funcionales de la situación problemática	0.0 – 0.5
	Detalla de forma completa los requisitos funcionales	0.0 – 1.0

objetual y herramientas de programación básicas.	Identifica y detalla las reglas de negocio del sistema, clasificándolas de forma adecuada	0.0 – 1.0
	Identifica las entidades del modelo del mundo del problema, las describe y construye el diagrama de clases	0.0 – 1.0
	Desarrolla los algoritmos de las clases asociadas al modelo del mundo para solucionar el problema	0.0 – 1.5

7. REFERENCIAS BIBLIOGRÁFICAS

Pressman, R. S. (2010). Ingeniería del Software-Un enfoque práctico. 7ª Edición McGraw-Hil.

Villalobos, J. A., & Casallas, R. (2006). Fundamentos de programación: aprendizaje activo basado en casos: un enfoque moderno usando Java, UML, Objetos y Eclipse (No. Sirsi) i9789702608462).

Elaborado Por:	Carlos Andrés Mera Banguero
Versión:	1.0
Fecha	Agosto de 2022
Aprobado Por:	Comité Curricular Departamento de Sistemas de Información Acta 02 del 7 de febrero de 2023