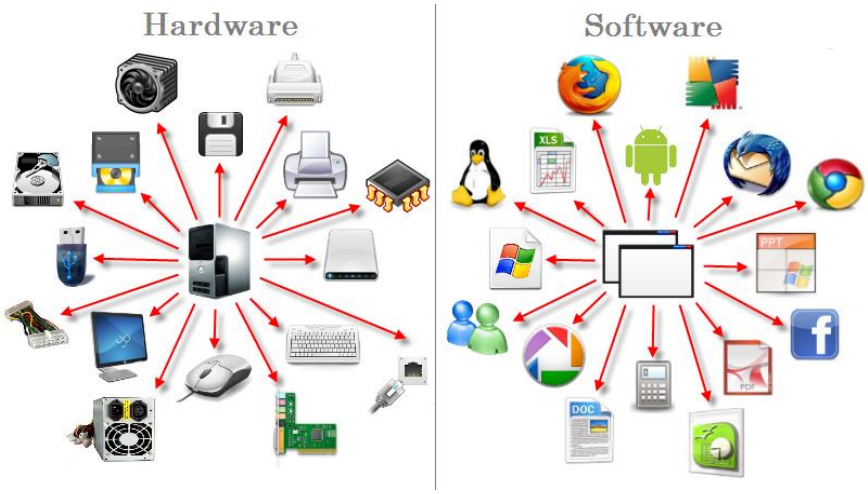
Manual_Programacion_estructurada

Vol_01

"Fundamentos de Programación"



La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora, utilizando subrutinas y tres estructuras: secuencia, selección (*if* y *switch*) e iteración (*for* y *while*),

El teorema del programa estructurado proporciona la base teórica de la programación estructurada. Señala que tres maneras de combinar programas son suficientes para expresar cualquier función computable: secuencia, selección e iteración.
Esta observación no se originó con el movimiento de la programación estructurada. Estas estructuras son suficientes para describir el ciclo de instrucción de una unidad central de procesamiento, así como el funcionamiento de una máquina de Turing. Por lo tanto un procesador siempre está ejecutando un "programa estructurado" en este sentido, incluso si las instrucciones que lee de la memoria no son parte de un programa estructurado.

Estos temas fueron abordados durante la década de 1960 y principio de los años 1970, con importantes contribuciones de Dijkstra, Robert W. Floyd, Tony Hoarey y David Gries.

- Ingeniería de Software es el estudio de los principios y metodologías para el desarrollo y mantenimiento de sistemas software (Zelkovitz, 1978).
- Ingeniería de Software es la aplicación práctica del conocimiento científico al diseño y construcción de programas de computadora y a la documentación asociada requerida para desarrollar, operar y mantenerlos. Se conoce también como desarrollo de software o producción de software (Bohem, 1976).
- La ingeniería de Software trata del establecimiento de los principios y métodos de la ingeniería a fin de obtener software de modo rentable, que sea fiable y trabaje en máquinas reales (Bauer, 1972).
- La ingeniería de Software es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación, y mantenimiento del software (Actual).

Ingeniería de Software:

Es la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software.

Software de Sistema:

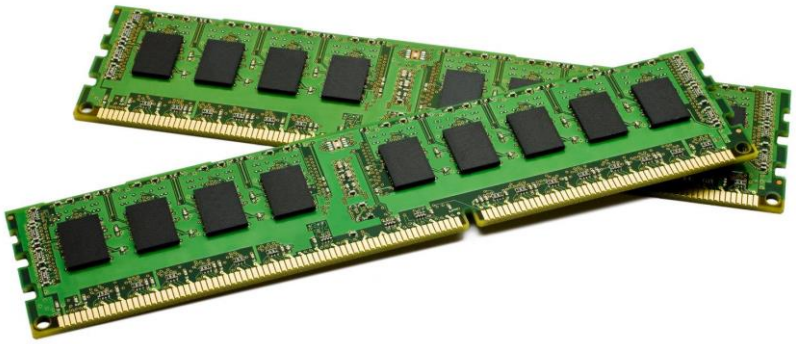
Medio para un Fin (sirve para controlar e interactuar con el sistema operativo).

Software de Aplicación:

Es una herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos logrando con eficacia un mejor control del proceso.

RAM

La memoria de acceso aleatorio (*Random Access Memory*, RAM)
En la RAM se cargan todas las instrucciones que ejecutan la (Unidad Central de Procesamiento, Procesador) y otras unidades del hardware.



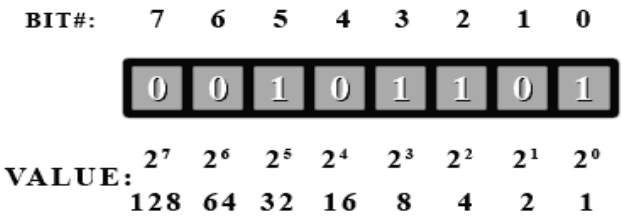
- Es la memoria Principal.
- Funciona como celdas de memoria.

Lectura y Escritura

Son las operaciones que se pueden realizar sobre la celda de memoria.

La Lectura se se trae una copia del dato almacenado.
La Escritura sobrescribe el dato previamente almacenado en la celda.

Sistemas numéricos

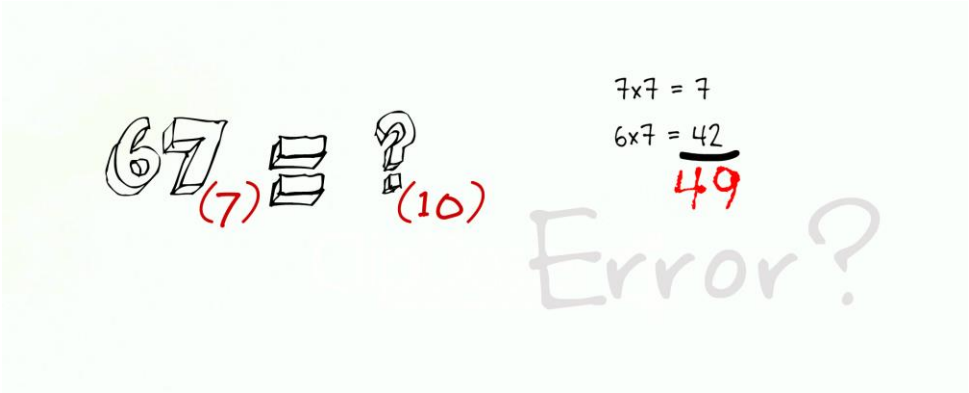


Un Dígito (10)
0 … 9
Dos Dígitos (5)
0 … 44
Tres Dígitos (2)
0 … 111

bit: dígito binario.
byte: conjunto de 8 bits.

bit: 8 = -3 ….. 4
byte: 256 = -128 ….. 127

- ASCII (Código Estándar Estadounidense para el Intercambio de Información) - 8 bits.
- UNICODE (es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas, además de textos clásicos de lenguas muertas.) - 16 bits.



Fórmula para conocer la cantidad de combinaciones, donde “n” es el número de bits.

Overflow

Se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria.

Cuando un Tipo de Dato Byte (Java), se le asigna el valor 128 se genera un overflow.

+
0 ….. 256
- y +
-128 ….. 127

Data size table

- 1,024 (e.g. one Kilobyte = 1,024 bytes).

Bit	= 1 bit
Byte	= 8 bits
Kilobyte	= 1024 bytes
Megabyte	= 1024 kilobytes
Gigabyte	= 1024 megabytes
Terabyte	= 1024 gigabytes
Petabyte	= 1,048,576 gigabytes
Exabyte	= 1,073,741,824 gigabytes
Zettabyte	= 1,099,511,627,776 gigabytes

Binario	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Internet: Red mundial de computadoras que maneja muchos estándares, como:
TCP-IP (transmission control protocol-internet protocol),
http (hyper text transfer protocol),
ftp (file transfer protocol), etc.

Compilador: Es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación. Agarra todo el código y lo traduce a código máquina, línea por línea y lo envía al procesador.

Intérprete: es un programa informático capaz de analizar y ejecutar otros programas. Los intérpretes se diferencian de los compiladores o de los ensambladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria.

Variables

Una Variable funciona como una celda o espacio de memoria, que poseen un conjunto de características.

- Posee un Nombre.
- Se refiere a un Tipo de Dato.
- Almacena un Dato.
- Tiene una Dirección.

En Java no se puede manipular la dirección, pero es controlada por el sistema.

- Inicializar una variable: Asignarle un valor por primera vez.
- Literal: Es un valor fijo que nunca cambia.
- Acumulador: Es una variable que cuando modifica su valor lo hace incrementando o decrementando en una cantidad variable.
- Contador: Es una variable que cuando modifica su valor lo hace incrementando o decrementando constantemente.

x = a + 50000
variables literal

Tipos de Dato

- entero (int, byte, long)
- real (double, float)
- texto (String)
- carácter (char)
- lógico (boolean)

Símbolos para describir las variables

- n: Nombre de la variable
- t : Tipo de dato
- v: valor inicial
- u: unidades de los datos
- c: valor constante
- d: dominio (conjunto estricto de valores que puede tomar la variable)

Algoritmos y Diagramas de Flujo

Un algoritmo es un conjunto de instrucciones para ejecutar una tarea. [Brookshear 1997].

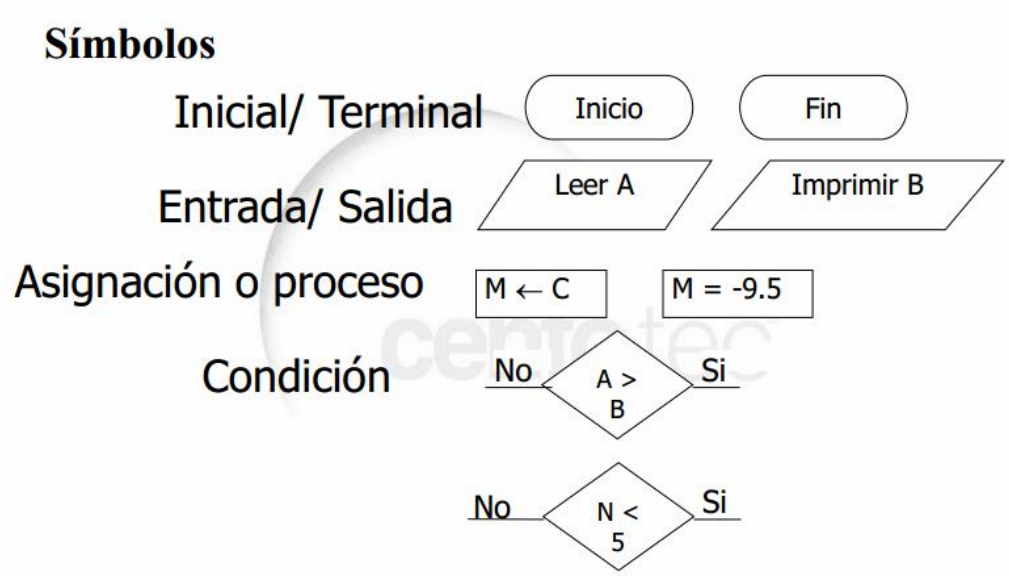
Es una manera formal y sistemática de representar la descripción de un proceso.

En términos más precisos: un algoritmo es un conjunto finito de instrucciones ejecutables, no ambiguas, que dirige una actividad que termina. [Brookshear 1997].

Representaciones de un algoritmo

- Pseudocódigo.
- Diagrama de flujo.

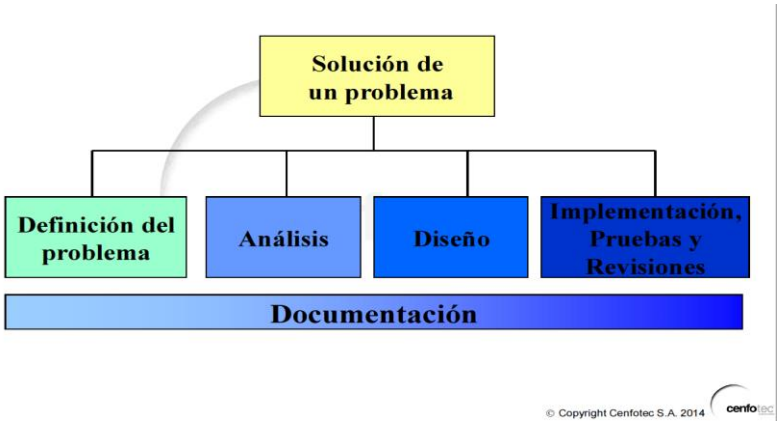
“Siempre tiene que haber una representación para poder usar o ejecutar un algoritmo”



Tipos de Errores

- Error Lógico: No da error al compilarlo, pero el resultado no es correcto.
- Error de Compilación: Se originan por errores de sintaxis. No se apega a las reglas del lenguaje de programación.
- Error de Ejecución: El programa se cae en medio de la ejecución, alerta al programador.

Proceso estructurado



Solución de un problema

Definición: debe constar de una descripción del problema bien definido.

Ejm: “Hallar la paz del Mundo” (no es un problema bien definido).

- Descripción de una Variable (n: variable, t: tipoDato, v: valorInicial, c: valorConstante)

Análisis: consiste en entender el problema y estudiarlo para encontrar en él los conceptos importantes de estos conceptos.

Pasos para analizar el problema:

1. Comprender la situación
2. Definir insumos o datos de entrada
3. Definir los productos esperados o “SALIDAS” del programa
4. Especificar el proceso a seguir para pasar de las “ENTRADAS” a obtener las “SALIDAS”

Diseño: establecer una solución basada en los conceptos descubiertos durante el “ANÁLISIS”, se estructura una secuencia lógica de pasos que la computadora va a ejecutar, (deberá apegarse a las reglas de programación).

- Pseudocódigo
- Diagrama de flujo

Implementación: pasar el algoritmo de la solución a un lenguaje de programación para que sea ejecutado por la computadora.

Elegir un Lenguaje:

- C
- C++
- Basic
- Cobol
- Java
- Visual Basic

Lista de chequeo y Pruebas: probar varios escenarios para asegurar el correcto funcionamiento del sistema, (tomar en cuenta si se requieren cambios o ajustes posteriores, “MANTENIMIENTO”)

Estructuras de Control

- Estructuras secuenciales
- Estructuras condicionales
- Estructuras iterativas

Lenguajes de Programación

Lenguaje de Bajo Nivel (Lenguaje de máquina)
Lenguaje de Alto Nivel

Pseudocódigo

Está diseñado para la lectura humana en lugar de la lectura mediante máquina, y con independencia de cualquier otro lenguaje de programación.
el pseudocódigo omite detalles que no son esenciales para la comprensión humana del algoritmo, tales como declaraciones de variables, código específico del sistema y algunas “subrutinas”.

Diagramas de flujo

Es una representación gráfica de un proceso. Cada paso del proceso es representado por un símbolo diferente que contiene una breve descripción de la etapa de proceso. Los símbolos gráficos del flujo del proceso están unidos entre sí con flechas que indican la dirección de flujo del proceso.

Beneficios del diagrama de flujo.

- Facilita la obtención de una visión transparente del proceso, mejorando su comprensión.
- Permiten definir los límites de un proceso.
- Proporciona un método de comunicación más eficaz, al introducir un lenguaje común.
- Estimula el pensamiento analítico en el momento de estudiar un proceso.

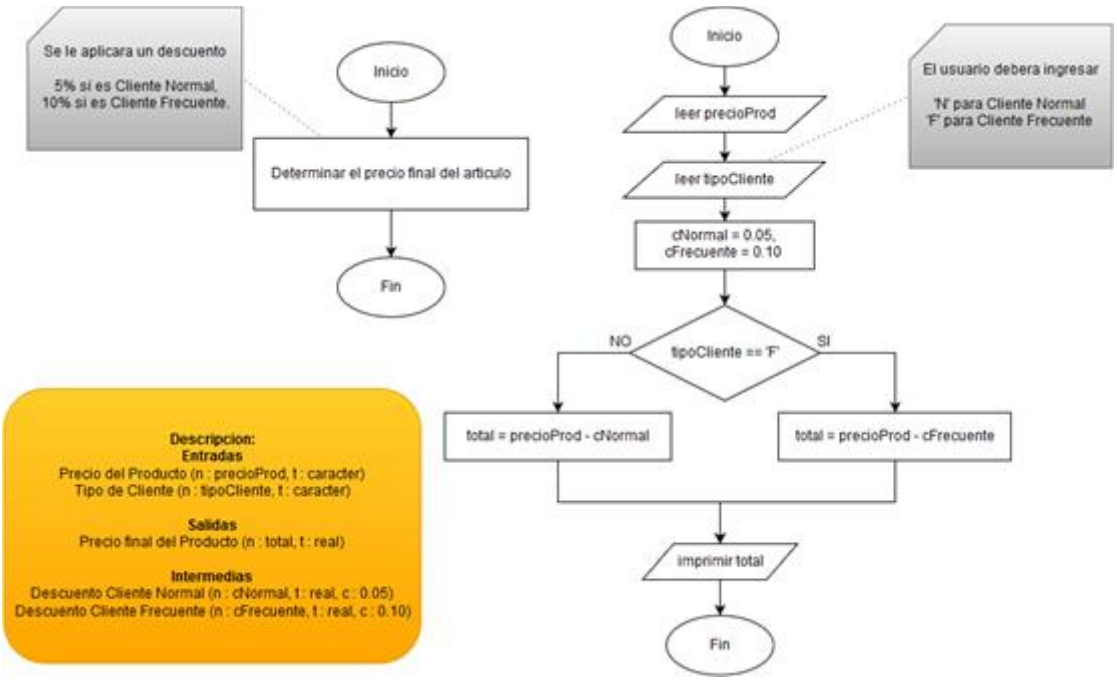
Control de Calidad Simple

- Cumple con los requerimientos?
- Está libre de defectos?
- Funciona correctamente?
- Es eficiente?

Ejm Proceso Estructurado

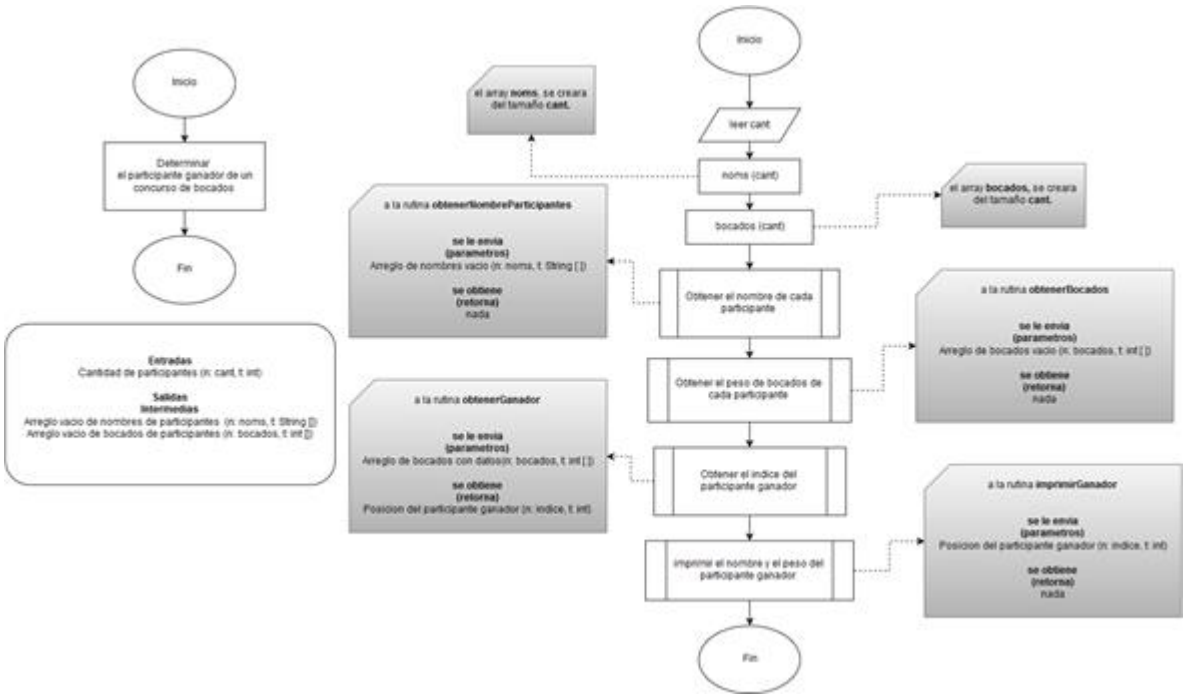
Ejm, Diagrama de flujo #1.

Descrip. | Entradas, salidas e intermedias. | Diagrama Solución |.



Ejm, Diagrama de flujo #2.

Descrip. | Entradas, salidas e intermedias. | Diagrama Solución |.

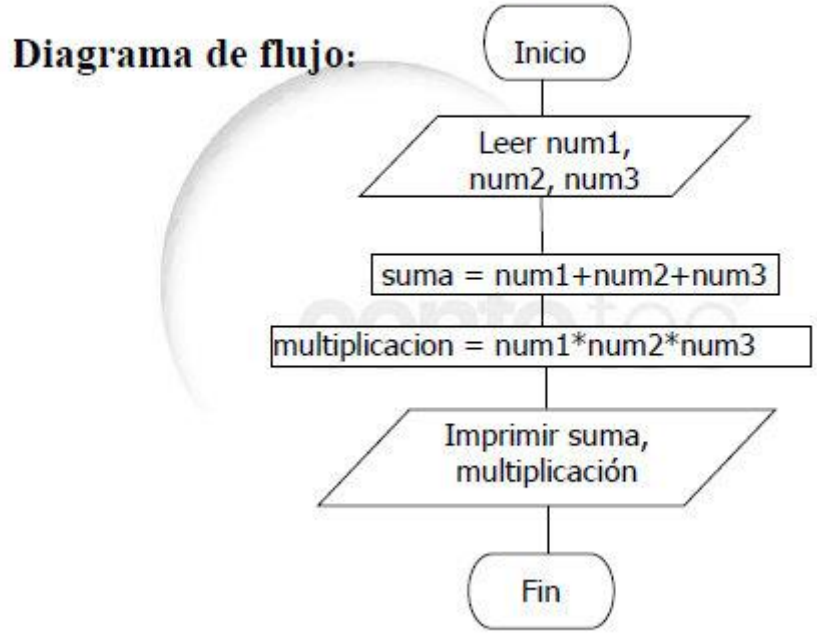


Estructura Secuencial

"Conjunto de instrucciones apiladas para realizar una tarea"

Diseño de la solución o Algoritmo :

Diagrama de flujo:



Java ejm:

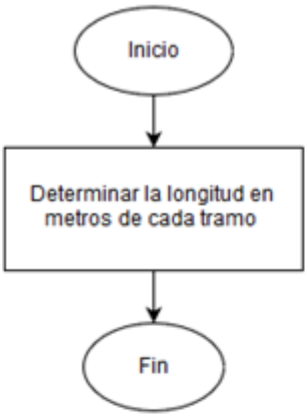
```
int num1 = 12;
int num2 = 34;
int num3 = 5;

int suma;
int multiplicacion;

String msj;

suma = num1 + num2 + num3;
multiplicación = num1 * num2 * num3;

msj = "La suma de los números es: " + suma +
        "\n La multiplicación de todos los números es: " + multiplicación;

out.print(msj);
```

Estructura Secuencial

Ejm, Diagrama de Flujo #3.

| Diagrama Solución |.

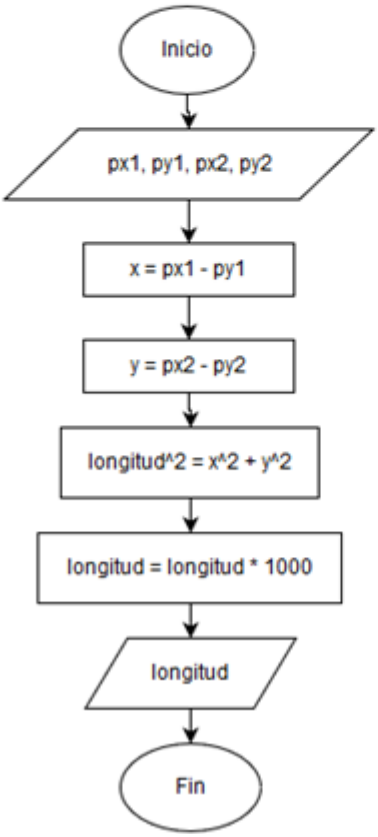
Determinar la longitud en metros de cada tramo.



Descrip
Entradas
(parametros)
Coordenada x1 (n: px1, t:real)
Coordenada y1 (n: py1, t:real)
Coordenada x2 (n: px2, t:real)
Coordenada y2 (n: py2, t:real)

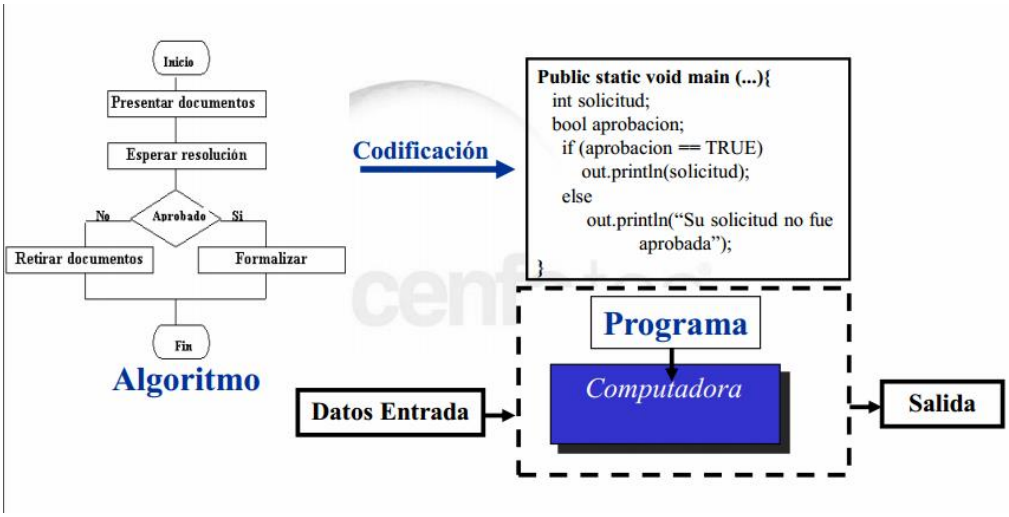
Salidas
(retorna)
Longitud de cada tramo en metros (n: longitud, t:real)

Intermedias
Cateto 1 (n:x, t:real)
Cateto 2 (n:y, t:real)



Estructura de Selección

"Analizar para determinar"



La estructura de selección, es una estructura lógica que permite controlar la ejecución de aquellas acciones que requieren de ciertas condiciones para su realización.

Se organizan por

- ★ Apilamiento: Es cuando se ejecuta después
- ★ Anidamiento: Cuando se ejecutan dentro de otra ramificación.

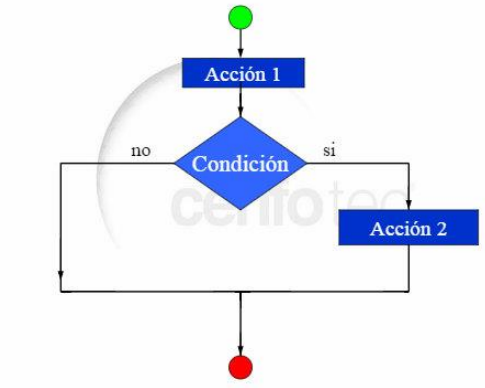
ejm: Anidamiento.

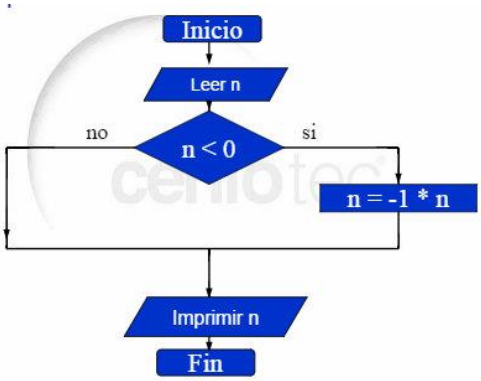
```
if ( condición ) {

} else {
   if ( condición  ) {

   } else {

   }
}
```

Se clasifican en:

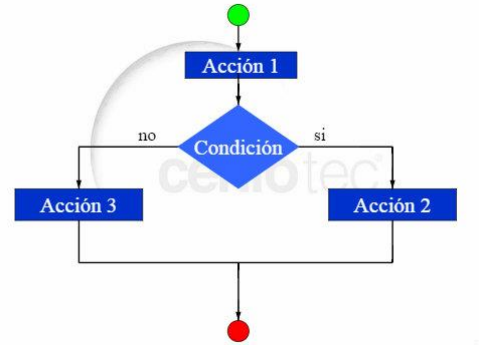
- Simples: Es aquella que con dos ramificaciones, solo hay acción en una de las ramificaciones.
- Dobles: Hay acciones en ambas ramificaciones. Se necesita una expresión lógica.
- Múltiples: Tiene varias ramificaciones, siempre hay acción en sus ramificaciones.

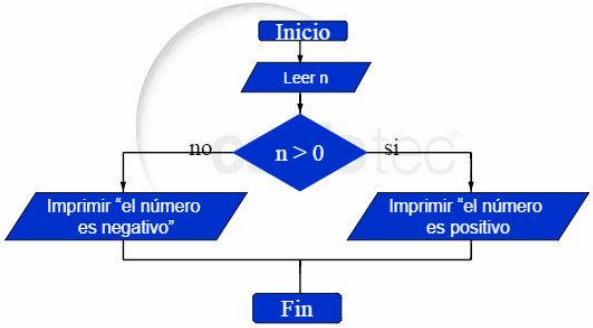
❖ Estructura de selección simple.



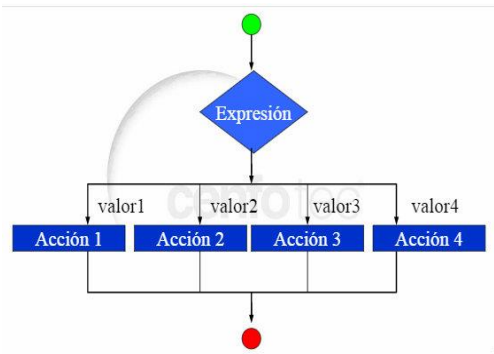


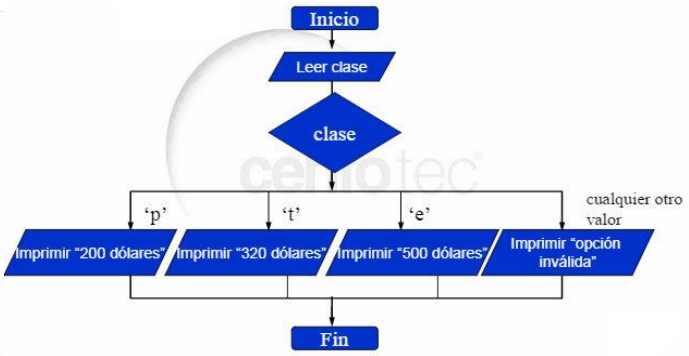
❖ Estructura de Selección Doble.





❖ Estructuras de selección múltiple.





Formas de Ordenamiento:

- Anidamiento.
- Apilamiento.

Expresiones lógicas se pueden clasificar en:

Las expresiones lógicas sirven para plantear condiciones o comparaciones y dan como resultado un valor booleano (verdadero o falso), es decir se cumple o no la condición. Se pueden clasificar en simples y complejas. Las expresiones simples se forman relacionando operandos, variables y/o constantes mediante operadores relacionales.

- Simples: Se forman a partir de operadores relacionales.
- Complejas: Permiten representar situaciones complejas.

Operadores relacionales (simples)

:

< , > , <= , >= , =, <>

= Para asignar

== Para comparar

7 / 2 N mod = 1

< , > , <= , >= , =, <> Ejemplos: X = 1 , horas >= 40, N > 0 , N mod 2=0 N <> Z

A fin de no confundir el operador de asignación "=" con el operador relacional =, utilizaremos el símbolo "==" para representar el operador relacional "=", para evitar confusiones. Por lo que nuestros ejemplos quedarían así: Ejemplos: X == 1 , horas >= 40, N > 0 , N mod 2==0, N <> Z.

Operadores lógicos (complejas)

Permiten representar situaciones complejas.
Se utilizan para crear expresiones que permiten crear operaciones.
y (and), o (or), not

X > 2 y X < 8 , sexo == 'F' y edad > 20 , J == 's' o J == 'S', not (M == 5), not (S>0)

Operadores Lógicos "or" & "not"

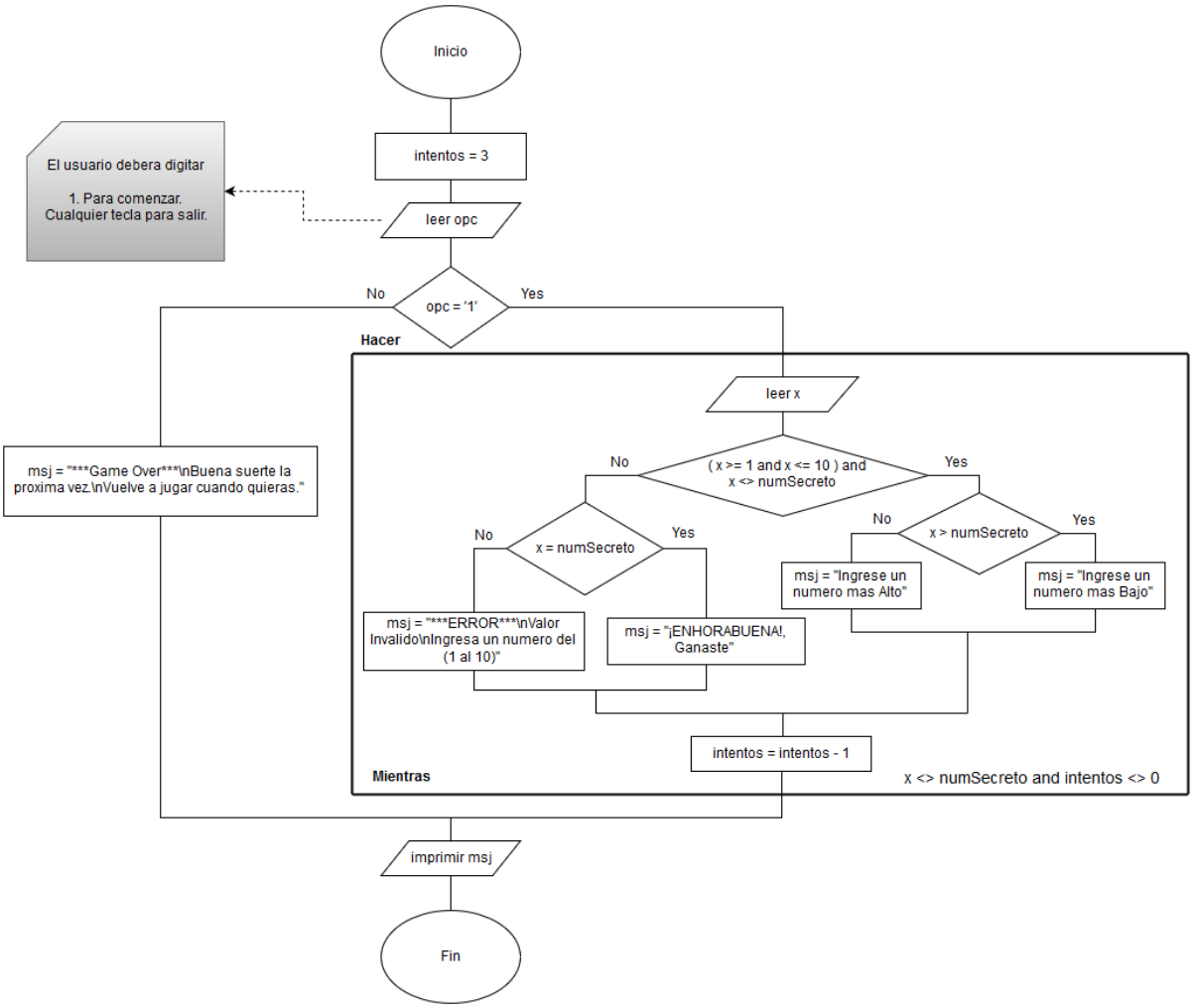
A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

A	NOT A
0	1
1	0

and, (&&)
El resultado es verdadero sólo si ambos enunciados son verdaderos.
or, (| |)
El resultado es verdadero siempre, menos cuando ambos enunciados son falsos.
not (!)
Inversor

Ejm Diagrama de flujo #1.
Diagrama Solución.
Adivinar el número secreto.



Estructuras de Iteración

"Estructuras de control"

Se utiliza cuando se requiere llevar a cabo un proceso de repetición para obtener un resultado que no podemos conseguir de una vez, es decir, que se debe conocer gradualmente, los resultados intermedios que nos llevan conseguir la tarea final.
Partes que componen una estructura de iteración:

- Cuerpo del ciclo (el conjunto de pasos que llevan a concluir una subtarea).
- Control del ciclo (permite que el ciclo itere el número de veces correcto).

El control de ciclo se compone de:

Inicialización (i = 0) dar un valor al contador
Evaluación (i < 5) determinar si se entra al ciclo
Modificación (i = i + 1) altera el valor incrementando o decrementando

Contador: es una variable que se incrementa o decrementa en cada iteración (lleva el control del número de repeticiones).

Las estructuras de iteración se clasifican en 2 ciclos:

Condicionales e Incondicionales:

- Los incondicionales, (con contador)
    - Se sabe que se ejecuta 'N' veces.

Para aumentar el contador en Java.

```
i = i + 1
i = i - 1
i ++  ( Solidario )
++ i  ( Egoísta )
i --  ( Postdecremento )
-- i  ( Predecremento )
```

- Los condicionales, (con centinela o bandera)

Centinela o Bandera: cuando aparece, obliga a que termine el ciclo. Puede obtenerse al ser introducido por el usuario en un proceso de lectura o por algún procesamiento dentro del cuerpo del ciclo, no se conoce el número de veces que debe ejecutar.

En Java.

- Ciclo while
- Ciclo do_while
- Ciclo for

Planeación de un ciclo o estructura de control:

- Paso 1. Reconocimiento de la necesidad de un ciclo.
  - El propósito de un ciclo es proporcionar repetición, un programa que deba repetir uno o más pasos consiguiendo un resultado gradualmente, incluirá un ciclo para conseguirlo.

- Paso 2. Diseño del cuerpo del ciclo: identificar las acciones que se repiten.
  - Las acciones deben repetirse para lograr la solución, según sea el problema a resolver.

- Paso 3.Diseño del control del ciclo: identificar la inicialización y la condición del ciclo.
  - Cuando se conoce el número de veces que se ejecutará el ciclo.
  - Cuando el usuario introduce un valor terminal o especial.
  - Cuando dentro del ciclo se da una condición que obliga a que la condición que controla el ciclo sea falsa.

- Paso 4. Revisión del control del ciclo: completar las acciones que se repiten, la modificación y la inicializaciones que no se hayan tenido en cuenta.
  - No quedar fuera por uno
  - No quedar afuera por medio.
  - No escribir ciclos finitos.

Ejm, Rutinas en Java #1

Convertir un número decimal a binario.

```
public static int num;

public static String convertirNumeroEnBinario( ) {

        int mod;
        int x;
        String resul;
        String cadena = "";
        String cadena2 = "";

        x = num;

        for(boolean salir = false; salir != true; ){

                mod = x % 2;
                cadena += mod;
                x = x / 2;

                if(x <= 0){
                        salir = true;
                }
        }

        for(int i = cadena.length()-1 ; i >= 0 ; i--){
                cadena2 += cadena.charAt(i);
        }

        resul = " El número " + num + ", en binario es: " + cadena2;

        return resul;

}
```

Ejm, Rutinas en Java #2.

Analizar para determinar si un número es “Número de Armstrong”.

```
import java.io.*;

 public static int num;

 public static String analizarSiEsNumeroArmstrong( ) {

        int x, digito;
        int cifra = 0;
        String cantDigitos = "";
        String resul = "";

        cantDigitos += num;

        x = num;

        while(x > 0){
                digito = x % 10;
                cifra += (int)Math.pow(digito, cantDigitos.length( ));
                x = x/10;
        }

        resul = " El numero " + num;

        if(num == cifra){
                resul += (" SI cumple la Ley de Armstrong");

        }else{
                resul += (" NO cumple la ley de Armstrong");

        }

        return resul;

}
```

Ejm, Rutinas en Java #3.
Mostrar fecha y hora actual.

```
import java.util.Calendar;
import java.util.GregorianCalendar;

public static String obtenerFecha( ) {

        Calendar obj = new GregorianCalendar();
        String date;
        String meses[] = {"Enero", "Febrero", "Marzo", "Abri", "Mayo", "Junio",
              "Julio" + "Agosto", "Setiembre", "Octubre", "Noviembre", "Diciembre"};
        String dias[] = {"Domingo", "Lunes", "Martes", "Miercoles",
                  "Jueves", "Viernes", "Sabado"};

        int seg = obj.get(Calendar.SECOND);
        int min = obj.get(Calendar.MINUTE);
        int hora = obj.get(Calendar.HOUR_OF_DAY);
        int dia = obj.get(Calendar.DAY_OF_MONTH);
        int diaSemana = obj.get(Calendar.DAY_OF_WEEK);
        int mes = obj.get(Calendar.MONTH);
        int año = obj.get(Calendar.YEAR);

        date = "Hoy es: " + dias[diaSemana-1] +
                  " " + dia + " " + meses[mes-1] + " " + año +
                  "\nHora: " +   hora + ":" + min + ":" + seg;

        return date;

}
```

Clase String

"Cadena de caracteres"

Son tipos de datos definidos por el usuario que describen, se pueden considerar como un identificador que nos permite manipular un objeto.

Los atributos o características del tipo.

- Nombre
- Apellido
- Edad
- Fecha de Nacimiento

En Java.

```
String nombre, nombreMinus, nombreMayus, longitudNom, letra1;


nombreMayus = nombre.toUpperCase();
nombreMinus = nombre.toLowerCase();

longitudNom = nombre.length;

letra1 = nombre.charAt(0);
```

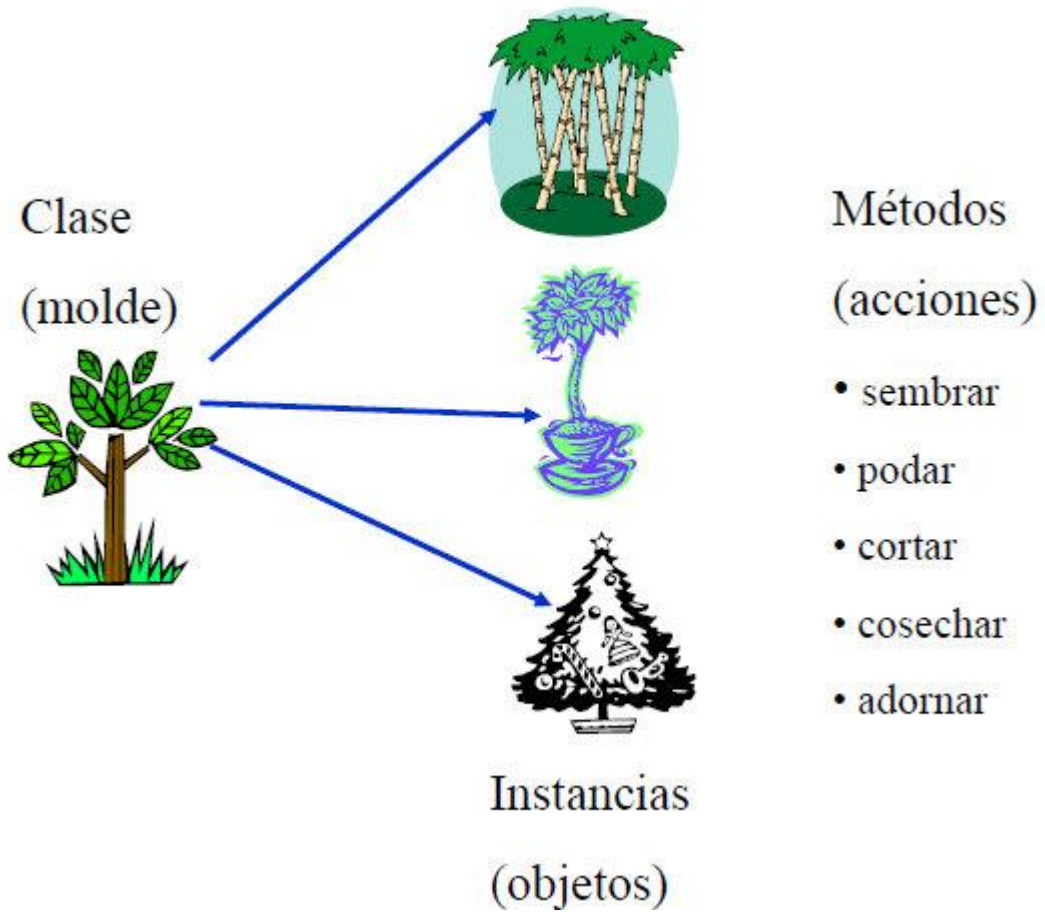
Métodos para manipular los objetos de tipo String.

- length
- toLowerCase
- toUpperCase
- trim
- charAt
- indexOf
- replace
- substring

Nota:

Para comparar datos de tipo String (cadena de caracteres), se debe utilizar el método (.equals) que compara el contenido de la variable.

```
boolean x;

If( cadena.equals(cadena2) ) {
    x = true;
}else{
    x = false;
}
```



Rutinas

"Delegar Responsabilidades (modularizar)"

1	2	3
tipoRetorna java -> double	nombre calcularIndice	(parámetro) { double pa, double pb, double pc{
   }		

Se compone de:

- Encabezado.
- Cuerpo.

Los parámetros pueden o no existir, si no existen, los () de los parámetros van vacíos.

Si los parámetros son varios, deben ir separados por (coma)'.
Los parámetros son los tipos de datos que recibe la Rutina.

El nombre de la rutina debería indicarse de tal forma que permita comprender qué tarea lleva a cabo esa rutina. Es común, utilizar verbos en infinitivo acompañado de un sustantivo.

- El Cliente no sabe lo que pasa en el Servidor.
- El Servidor no sabe lo que pasa en el Cliente.
- Las Rutinas está encapsulada fuera del main pero dentro de la clase Principal.

Parámetros:

- Parámetros Actuales: en el Cliente.
- Parámetros Formales: en el Servidor.

Los Parámetros Actuales deben coincidir con los formales principalmente en:

1. En la cantidad.
2. En los tipos de datos.
3. En el orden en que los tipos deben ser enviados.

Mapa de Comunicación entre Rutinas

"Diagrama de comunicación"

Las Rutinas se clasifican en:

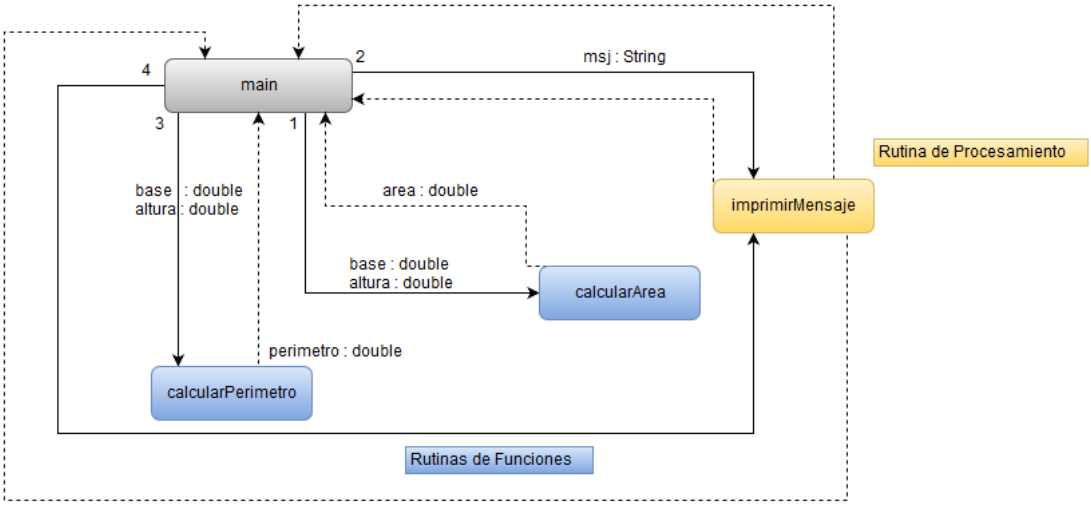
- De Función: Devuelve o retornan un solo valor.
- De Procesamiento: No devuelven nada.

Las rutinas pueden actuar como Cliente o como Servidor.
Cuando se hace una llamada a una rutina (se requiere algo), la rutina que efectúa la llamada se convierte en "Cliente" y la rutina que procesa la información que requiere el "Cliente" se dice que actúa como "Servidor".

ejm, Rutinas Cliente/Servidor (parámetros actuales/reales y parámetros formales) #1.

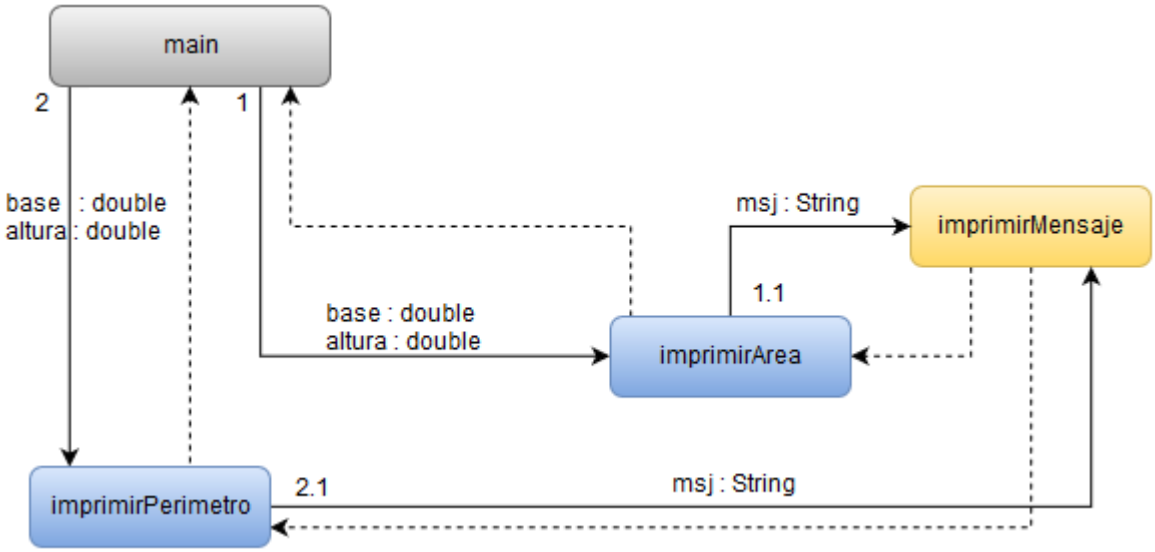
```
// El main actúa como cliente de calcularIndice       (comentario)
// calcularIndice actúa como servidor del main       (comentario)

public static void main {

        int indice;

        indice = calcularIndice(parametrosActuales); // Cliente

}

static Double calcularIndice (parametrosFormales) {  // Servidor

         if(x > ??){
            indice = -1;
         }

         return indice;
}
```

ejm, Mapa de Comunicación entre Rutinas #1.



- ➔ El main actúa como Cliente de calcularArea,
  calcularArea actúa como Servidor del Main.
- ➔ El main actúa como Cliente de imprimirMensaje,
  imprimirMensaje actúa como Servidor del main.

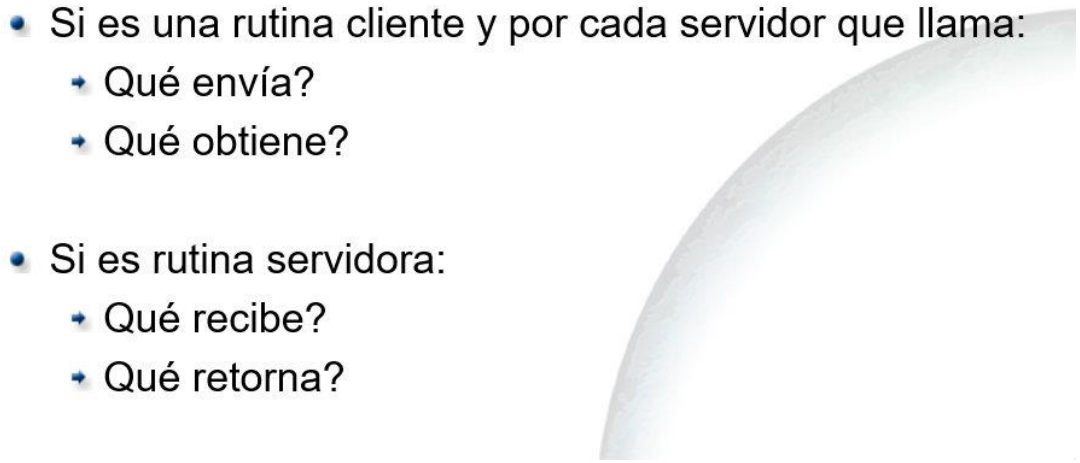
ejm, Mapa de Comunicación entre Rutinas #2:



Llamado de rutinas de forma apilada.

- ➔ El main actúa como Cliente de imprimirArea,
  imprimirArea actúa como Servidor del main.

➔ imprimirArea actúa como Cliente de imprimirMensaje,
imprimirMensaje actúa como Servidor de imprimirArea.



Variables Globales & Locales.

ámbito global: es conocida por todas las rutinas, se agrega la palabra (static), además debe estar fuera de toda rutina pero dentro del class.

```
static Double precio = 20.25;
```

ámbito local: es conocida sólo dentro de una rutina o estructura de control que lo compone.

```
int num1 = 50;
```

Variables de Valor Almacena el dato al que se refiere la variable. ● byte ● int ● long ● double ● float ● char	Variables de Referencia Almacena la dirección al que se refiere la variable. Para comparar direcciones se usa (==). Para comparar contenido se utiliza (.equals). ● String ● Arreglos ● Rutinas ● Double ● Byte

Arreglos o Vectores

"Variable de Amor"

Estructura que almacena datos, todos del mismo tipo. Cada dato se accede por medio de un índice cuyo rango se encuentra entre 0 y la cantidad de elementos -1. Una vez que un arreglo se a creado, no se puede modificar su tamaño.

- 1er Paso.
  Declarar el arreglo.
  Tipo de dato que almacena [ ] nombre de la variable.
  int [ ] datos;

- 2do Paso
  Crear o construir el arreglo.
  datos = new tipo [cant]

- 3er Paso
  Asignar valor (Inicializar las posiciones del arreglo).
  datos [0] = 1;
  datos [1] = 1;
  datos [2] = 1;

Las 3 partes importantes de un arreglo son:

- El nombre
- El tipo [ ]
- El índice

El índice va de 0 al tamaño del arreglo -1.
Si se conoce los datos que lleva un arreglo se puede declarar, construir e inicializar en la misma línea.

ejm en Java:

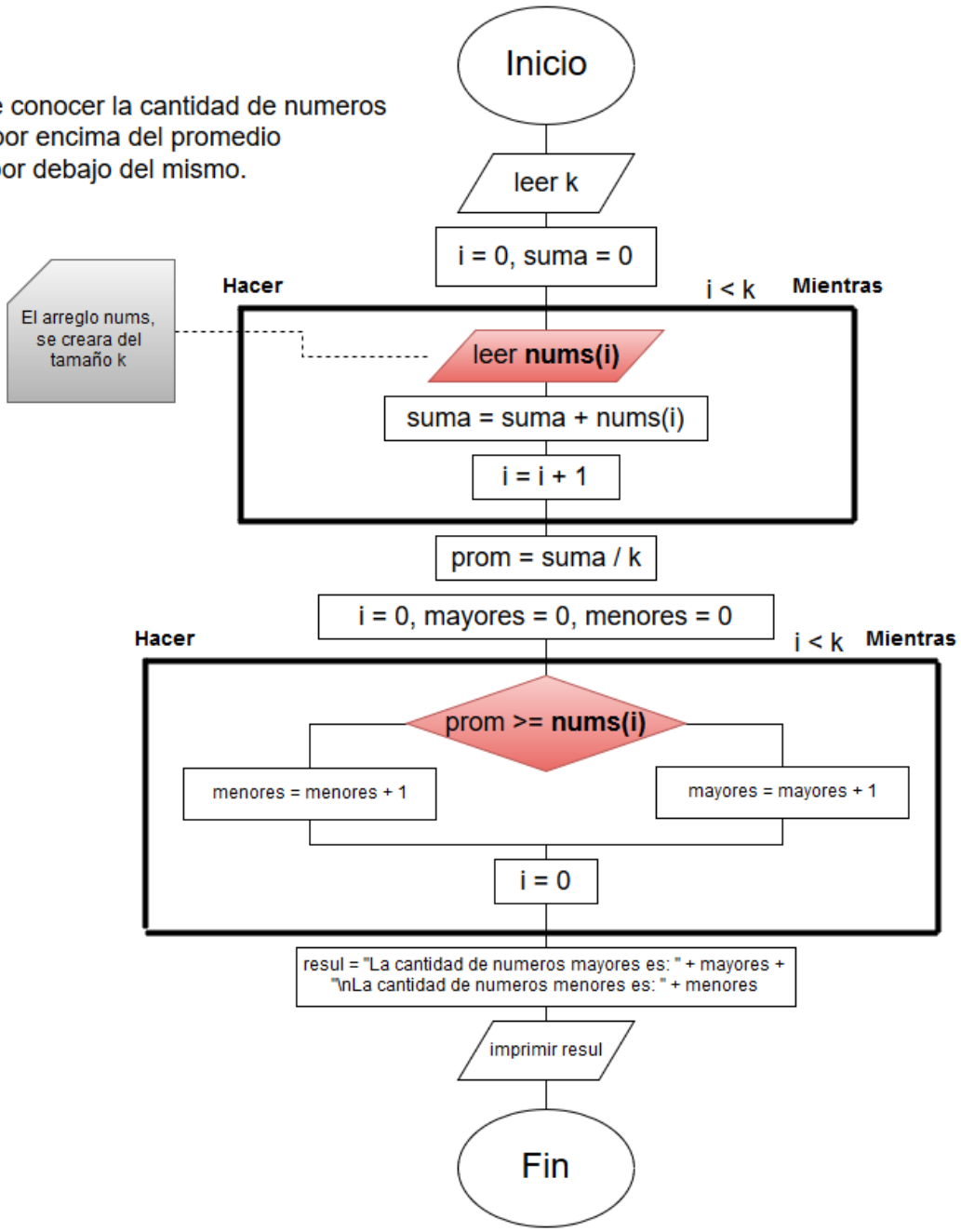
```
int [ ] nums = {1, 5, 8};
```

```
for(int i=o,  i<nums.length,  i++,){
   nums[i]=1;
```

```
}
```

Ejm, Arreglos en Java #1.

Se requiere conocer la cantidad de numeros que estan por encima del promedio y cuantos por debajo del mismo.



Notas:

- Java inicializa los valores de un arreglo por default.
  Si es numérico asigna (0).
  Si es de texto asigna (null).
  Si es booleano asigna (false).

- Error null? = apunta a una dirección inválida.

La restricción que debe cumplir una variable que sirve para controlar el ciclo pero también controla el índice del arreglo.

- Controlar el número de veces correcto.
- Inicializar datos desde 0 hasta el tamaño del arreglo -1.

Conviene inicializar los valores de un arreglo utilizando una estructura de repetición o iteración.

Java ejm:

```
for (int i = 0; i < 3; i++) {
        datos [ i ] = (i + 1);
}
```

Tabla de valores en un arreglo.

datos	char [ ]	arreglo
i	int	índice
datos [i]	char	valor

Nota importante:
Cuando un arreglo es enviado por parámetro, envía su dirección, eso quiere decir que si la rutina servidora modifica su valor, también se ve afectado el valor del arreglo en la rutina cliente.

Ejm, Ejercicios en Java.

obtenerMayor

```
int mayor = pvals[0];

for(int i = 1; i < pvals.length; i++){
        if(pvals[i] > mayor){
                mayor = pvals[i];
        }
}
```

obtenerIndiceMayor

```
int indiceMayor = 0;

for(int i = 1; i < pvals.length; i++){
        if(pvals[i] > pvals[indiceMayor]){
                indiceMayor = i;
        }
}
```

Ejm, Direcciones de Memoria.

int[ ][ ]datos;
datos = new int [3] [3]

	200	int[ ][ ]datos
Dirección del arreglo		
200	400	
	600	
	800	
dirección de la posición [0]		
400	0	datos[0]
	0	
	0	
dirección de la posición [1]		
600	0	datos[1]
	0	
	0	
dirección de la posición [2]		
800	0	datos[2]
	0	
	0	

Arreglos multidimensionales || Matrices

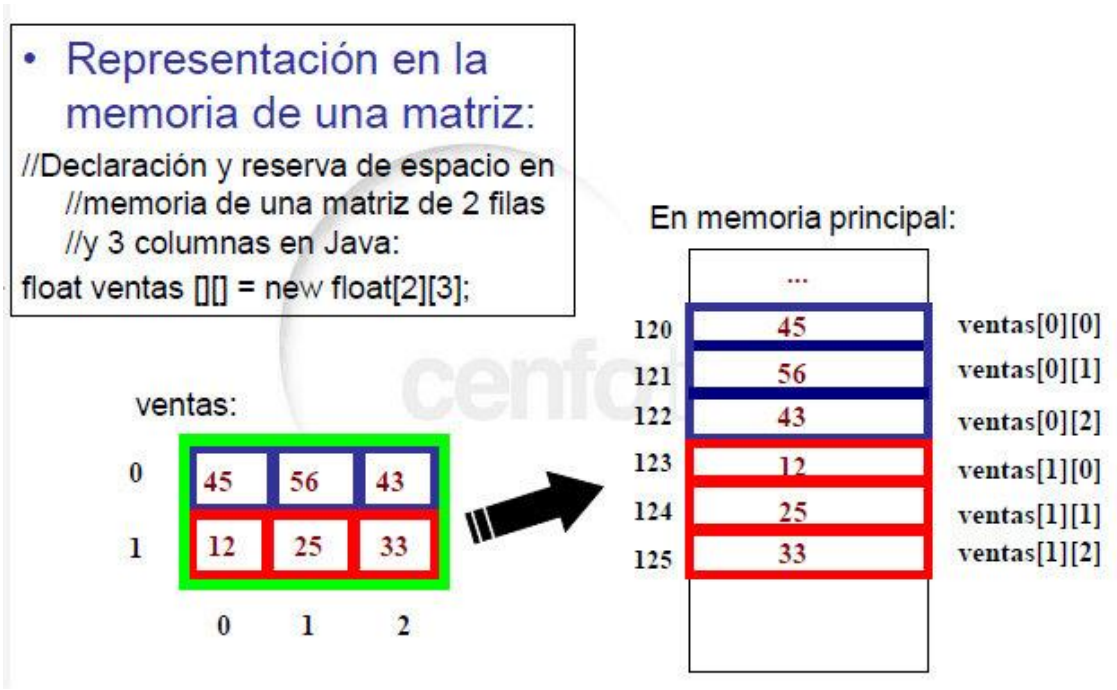
"Arreglos de arreglos || vectores de vectores"

```
float [ ][ ] ventas = new float [4][5];
```

Declaración de una matriz y reserva de espacio en memoria con 4 filas y 5 columnas.

Para declarar, construir e inicializar en una misma línea.

```
float [ ][ ] ventas = {2.3, 4.6, 0.5, 2.8, 1.1}, {2.3, 4.6, 0.5, 2.8, 1.1},
                      {2.3, 4.6, 0.5, 2.8, 1.1}, {2.3, 4.6, 0.5, 2.8, 1.1};
```



Arreglo de arreglos || vector de vectores.
Cada fila es un arreglo del arreglo contenedor

Nota:

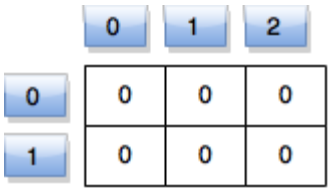
Las matrices se pueden ver como una tabla o como un arreglo de arreglos.
Una matriz debe tener la misma cantidad de columnas.

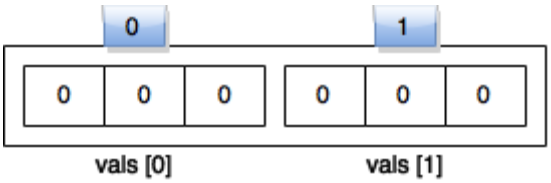
Matriz como tabla.	Matriz como arreglo de arreglos.
10 15 5 20 11 10 8 25 30	{ 10 15 5 }, { 20 11 30 }, { 8 25 30 }

Java ejm:

```
for (int rows = 0; rows < 2 || ; rows++) {
    for (int cols = 0; cols < 3 || ; cols++) {
        datos [ i ][ j ] = (valor);
    }
}
```

Int [ ][ ] vals = new int [2][3];





Variable	Tipo de Dato	Concepto	Tamaño
vals	int[ ][ ]	nombre	
i	int	índice (fila)	0... vals.length**-1**
j	int	índice (columna)	0... vals.length**-1**
vals[i]	int[ ]	índice (arreglo/fila)	
vals[i] j]	int	dato en la pos (“i”, “j”)	
vals.length	int	matriz (tamaño) / num filas	
vals[i].length	int	arreglo (tamaño) c/u	

Programación en Capas

"Capa de Presentación, capa de Negocio"

La programación en capas se utiliza para separar responsabilidades para estructurar de mejor manera las funciones y optimizar
el flujo del programa analizando de manera apropiada los errores en áreas específicas

Se compone de dos clases:

- Capa de Presentación
- Capa de Negocio
- Capa de Datos

La Capa de Presentación es la que debe llevar el control de todas las salidas y entradas del sistema, es decir el usuario envía los datos directamente a la capa de presentación.

La Capa de Negocio se encarga de la lógica del sistema.

Paso de parámetros por valor:

- Cuando el parámetro actual le envía al parámetro formal una **copia de su contenido.**
- Si la rutina modifica los parámetros formales no afecta al parámetro actual.

Paso de parámetros por referencia:

- Pasar una copia de la **dirección** del arreglo.
- Si la rutina modifica datos en los parámetros formales, **se modifican en los parámetros actuales,** (Pero si vuelve a construir el arreglo dentro de la rutina, **"nombre = new tipoDato[tamaño]",** Los parámetros formales no modifican los parámetros actuales), esto se debe a que al volver a construir un arreglo en una rutina, ese arreglo ocupará una dirección distinta.

"Java no tiene la posibilidad de enviar por parámetro un arreglo. (**pero lo simula**)".

Para devolver un dato inválido es recomendable utilizar:

- No estoy apuntando a ningún índice valido **-1**
- No estoy apuntando a ningún valor valido **null**

El **estado** de una aplicación se refiere a los valores asociados a cada una de las **variables** en **un instante determinado**.

Estado esencial:

Grupo de valores que necesitan ser compartidos para realizar la funcionalidad
Ejm: las mismas valores de notas se envían a cada rutina para q elaboren los diferentes procesos, como **sumarNotas, sacarPromedios, aprovadosReprovados.**

Estado de apoyo:

Solo necesitan ser conocidos localmente por el procesamiento que se lleva a cabo.

Capa de Presentación

```
import java.util.*;
import java.io.*;


public class IUEspect {

	static BufferedReader in = new
BufferedReader(new InputStreamReader(System.in));
	static PrintStream out = System.out;

   public static void main(String[] args)throws
java.io.IOException{

	int opc;
		boolean noExit = true;

		do{
			showMenu();
			opc = leerOpc();
			noSalir = runAction(opc);

		}while (noExit);

	}

	// code…
	// code…
	// code…


}
```

Capa de Negocio

```
import java.util.Calendar;
import java.util.GregorianCalendar;



public class Rutinas {

	private static String[][][]salasXDia= null;
	private static String[][]regis= new
String[250][10];

	public static String[][] obtenerSalaDia(int px) {

		return salasXDia [px];
	}

	public static String[][] obtenerRegistros() {

		return regis;
	}

	// code…
	// code…
	// code…


}
```

Ejm, Capa de Presentación - Rutinas en Java #1.

```
	static void mostrarMenu(){

		out.println("1.  Registrar espectaculo.");
		out.println("2.  Reservar asientos.");
		out.println("3.  Probar sistema.");
		out.println("4.  Reiniciar sistema.");
		out.println("11. Salir");
	}
```

```
	static int leerOpcion()throws java.io.IOException{

		int opcion;

		out.print("Seleccione su opci\u00a2n: ");
		opcion = Integer.parseInt(in.readLine());
		out.println();

		return opcion;
	}

	static boolean ejecutarAccion(int popcion)throws java.io.IOException{

		boolean noSalir= true;

		if (popcion >= 2 && popcion <= 7 && RutinasEspect.obtenerEstadoArreglosTridi() == null){

			out.println("No se ha registrado ningun espectaculo");
			return noSalir;
		}

		switch(popcion){

			case 1: // Registrar la informacion del espectaculo.

				int reg= RutinasEspect.obtenerCantRegis();

				if(reg > 0){
					out.println("No puede registrar otro espectaculo");
				}else{
					registrarEspectaculo();
				}
				break;

			case 2: // Reservar uno o varios asientos en un horario.
				break;

			case 3: // Probar sistema.

			probarSistema();
				break;

			case 4: // Reiniciar sistema.

				out.println(RutinasEspect.reinciiarSistema());
				break;

			case 11: //Salir de la aplicacion

				noSalir = false;
				break;

			default: //Cualquier otro valor dado por el usuario se considera inválido

				out.println("Opcion inv\u00a0lida");
				out.println();
				break;
		}

		return noSalir;
	}
```

Ejm, Capa de Negocio - Rutinas en Java #2.

```
public static boolean verificarFechaEspect(String[]nuevaFecha){

	boolean resul;
	Date f1, f2;

	f1= convertirStringAFecha(espect[2]);
```

```
        f2= convertirStringAFecha(nuevaFecha[1]);

        resul = f2.after(f1);

        return resul;

}

 public static String obtenerFechaEnString(Date pfecha){

                SimpleDateFormat sdf= new SimpleDateFormat("dd/MM/yyyy");

                return sdf.format(pfecha);
}

  public static Date obtenerFecha(int[] pdatosFecha){

      GregorianCalendar  calendario=
        newGregorianCalendar(pdatosFecha[2],pdatosFecha[1],pdatosFecha[0]);

        Date fecha= calendario.getTime();

        return fecha;
 }

  public static int[] obtenerDiaMesAnio(String pfecha){

                int[] datosFecha = new int[3];

                datosFecha[0]= Integer.parseInt(pfecha.substring(0,2)); //dias de la fecha
                datosFecha[1]= Integer.parseInt(pfecha.substring(3,5)); //mes de la fecha
                datosFecha[2]= Integer.parseInt(pfecha.substring(6,10)); //anio de la fecha

                return datosFecha;

 }

   static int obtenerRangeDiasEspect() {

                Date f1= convertirStringAFecha(espect[1]);
                Date f2= convertirStringAFecha(espect[2]);

                long numDias =(f2.getTime()-f1.getTime())/86400000;

                int x= (int)numDias +1;

                return x;

  }

    public static String[][] obtenerRegistros() {

                return regis;
  }
```

Metodos de ordenamiento

METODO BURBUJA

```
public static void burbuja(int[]matrix){
        int temp;
        for(int i=1;i < matrix.length;i++){
            for (int j=0 ; j < matrix.length- 1; j++){
                if (matrix[j] > matrix[j+1]){
                    temp = matrix[j];
                    matrix[j] = matrix[j+1];
                    matrix[j+1] = temp;
                }
            }
        }
    }
```

METODO INSERCION

```
public static void Insercion (int[] vector) {
      for (int i=1; i < vector.length; i++) {
         int aux = vector[i];
         int j;
         for (j=i-1; j > =0 && vector[j] > aux; j--){
              vector[j+1] = vector[j];
          }
         vector[j+1] = aux;
      }
   }
```

METODO SELECCION

```
public static void Seleccion(int[]matrix){
        int i, j, k, p, buffer, limit = matrix.length-1;
        for(k = 0; k < limit; k++){
            p = k;
            for(i = k+1; i < = limit; i++){
                if(matrix[i] < matrix[p]) p = i;
                if(p != k){
                    buffer = matrix[p];
                    matrix[p] = matrix[k];
                    matrix[k] = buffer;
                }
            }
        }
    }
```

METODO SHELL SHORT

```
public static void shellSort(int[] matrix) {
    for ( int increment = matrix.length / 2;increment > 0;
          increment = (increment == 2 ? 1 : (int) Math.round(increment /
2.2))) {
        for (int i = increment; i < matrix.length; i++) {
            for (int j = i; j > = increment && matrix[j - increment] >
matrix[j]; j -= increment) {
                int temp = matrix[j];
                matrix[j] = matrix[j - increment];
                matrix[j - increment] = temp;
            }
        }
    }
}
```

ORDENAMIENTO POR MEZCLAS (MERGE)

El algoritmo de ordenamiento por mezcla (Merge) se divide en dos procesos, primero se divide en partes iguales la lista:

```
public static void mergesort(int[ ] matrix, int init, int n){
    int n1;
    int n2;
    if (n > 1){
        n1 = n / 2;
        n2 = n - n1;
        mergesort(matrix, init, n1);
        mergesort(matrix, init + n1, n2);
        merge(matrix, init, n1, n2);
   }
}
```

Y el algoritmo que nos permite mezclar los elementos según corresponda:

```
private static void merge(int[ ] matrix, int init, int n1, int n2){
    int[ ] buffer = new int[n1+n2];
    int temp = 0;
    int temp1 = 0;
    int temp2 = 0;
    int i;
    while ((temp1 < n1) && (temp2 < n2)){
        if (matrix[init + temp1] < matrix[init + n1 + temp2]){
            buffer[temp++] = matrix[init + (temp1++)];
        }else{
            buffer[temp++] = matrix[init + n1 + (temp2++)];
        }
    }
    while (temp1 < n1){
        buffer[temp++] = matrix[init + (temp1++)];
    }
    while (temp2 < n2){
        buffer[temp++] = matrix[init + n1 + (temp2++)];
    }
    for (i = 0; i < n1+n2; i++){
        matrix[init + i] = buffer[i];
    }
```

```
}
```

ORDENAMIENTO RAPIDO

```
public static void Rapido(int matrix[], int a, int b){
    matrix = new int[matrix.length];
    int buf;
    int from = a;
    int to = b;
    int pivot = matrix[(from+to)/2];
    do{
        while(matrix[from] < pivot){
            from++;
        }
        while(matrix[to] > pivot){
            to--;
        }
        if(from < = to){
            buf = matrix[from];
            matrix[from] = matrix[to];
            matrix[to] = buf;
            from++; to--;
        }
    }while(from < = to);
    if(a < to){
        Rapido(matrix, a, to);
    }
    if(from < b){
        Rapido(matrix, from, b);
    }
}
```

Referencias

Profesor de Fundamentos de Programacion 1
Antonio Luna
Universidad Cenfotec

Semana 1

Referencia1
Las operaciones que se pueden realizar sobre las celdas de memoria son lectura y escritura es decir, yo puedo escribir o leer de ella. La lectura se trae una copia del dato almacenado en la celda en cambio, la escritura sobrescribe el valor almacenado en la celda de memoria.

Referencia2
Variable
Es una área de memoria que posee un conjunto de características, tiene un nombre, un valor, posee una dirección y tiene un tipo de dato.

Referencia3
Para poder usar una variable, primero se debe declarar y eso consiste en darle el tipo de dato, para ello debemos decir su “tipo” y su “nombre”.

Referencia4
No se puede usar utilizar para lectura una variable que no se ha inicializado.
Recordar - Hay verdades esenciales y verdades accidentales, el trabajo de un ingeniero es trabajar siempre bajo las verdades esenciales.
Recordar - El “overflow” o desbordamiento de buffer sucede cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto (buffer): Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original, que probablemente
pertenecían a datos o código almacenados en memoria. Esto constituye un fallo de programación.

Semana 2

Referencia1
No se puede usar para lectura una variable no inicializada.
Recordar;

1. Acumulador
Variable que cuando modifica su valor lo hace incrementando o decrementando en una cantidad variable.

2. Contador
Variable que cuando modifica su valor lo hace incrementando o decrementando en una cantidad constante.

Recordar
Concatenar es un el proceso de unir caracteres o cadenas de caracteres con otros tipos de dato utilizando el comando String
.
Recordar

ambigüedad se presta para que las ideas se puedan interpretar de diferentes maneras.

Recordar
Un traductor se en encarga de traducir las instrucciones del lenguaje de alto nivel a lenguaje máquina o binario.

Recordar - Debemos seguir el proceso de seguir un programa estructurado, Definición del problema, (análisis, diseño e implementación pruebas y revisiones).

Referencia2
Los nombres de las variables deben ser significativos es decir que nos ayuden a entender lo que la variable almacena, inicia con letras minúscula no lleva espacio en blanco en medio del nombre y cuando hay cambio de palabra se coloca en mayúscula.

Semana 3

Referencia1
Conviene hacer explícitos todos aquellos conceptos que son de valor en e dominio del problema y dejarlos representados en la solución.

Referencia2
Conviene no usar literales en las operaciones y es preferible asociarlas a identificadores que se convierten en constantes.
Recordar - Las intermedias son variables que no son “entradas” o “salidas”.

Semana 4

Referencia1
Cuando se necesita realizar un anidamiento es conveniente hacerlo en la ramificación del no.

Referencia2
La selección múltiple sólo permite cantidades exactas.

Referencia3
Siempre que tengamos estructuras de control anidándolas o apilándolas debemos de ser capaces de identificar para cada una su entrada y su respectiva salida.
Recordar - El cuerpo son las tareas a repetir y el central la cantidad de iteraciones.
Recordar - Inicialización, modificación, evaluación.

Semana5

Referencia1
Siempre que trabajemos con una selección simple especifiquemos la acción o acciones en la ramificación
de sí.

Referencia2
Siempre que necesitamos identificar en una estructura de iteración quién controla el ciclo el primer lugar que debemos hubiera es el que corresponde a la evaluación y confirmar con la inicialización y la modificación a la variable que controla el ciclo.

Referencia3
Cuando la variable que controla el acto participa del cuerpo del ciclo se debe tener mucho cuidado con el lugar donde se realiza la modificación con la inicialización y con la evaluación.

Referecia4
Siempre que un concepto requiera ser procesado dentro de una estructura de iteración en la que va cambio de valor con cada iteración, utilice la palabra “cada” para describir su función.
Recordar - Las variantes de evaluación del ciclo; evaluación al inicio y evaluación al final.
Recordar - Evaluación al final entra al menos una vez al ciclo y debe haber un dato como mínimo a ser procesado.