Resolución de Problemas y Algoritmos

# Estructuras Alternativas

Facultad de Informática
Universidad Nacional del Comahue

2021

# Temario

# Temario

# Introducción

- Las alternativas son partes de nuestra vida cotidiana. Ej. 'Si llueve voy al cine, si no llueve voy a la plaza', o una alternativa mas trascendental '¿que carrera voy a seguir?!'
- Las estructuras alternativas son una estructura de control útil cuando nos encontramos con problemas que requieren *ejecutar una o más instrucciones según diferentes casos*.
- En otras palabras, las estructuras alternativas bifurcan o dirigen la ejecución de un programa hacia un grupo de sentencias u otro dependiendo de una condición.

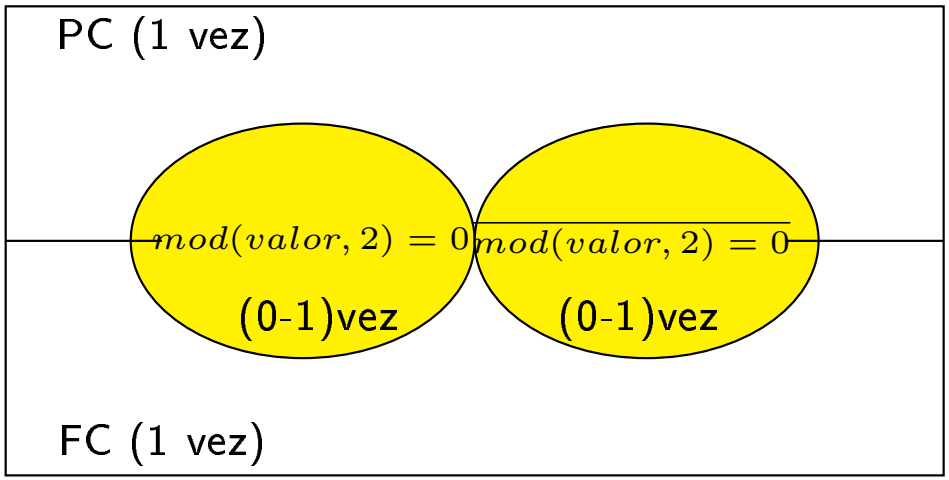
# Temario

# Alternativas en etapa de Análisis

- Un conjunto de datos (o una de sus partes) de estructura alternativa es un conjunto de datos en el cual se encuentran uno o varios subconjuntos, cuya presencia es aleatoria (condicional), con la salvedad de que la presencia de uno excluye la de todos los demás (Warnier) [1], ó
- La presencia aleatoria de un conjunto se representa por la etiqueta 'o - 1 veces'

# Alternativas en etapa de Análisis

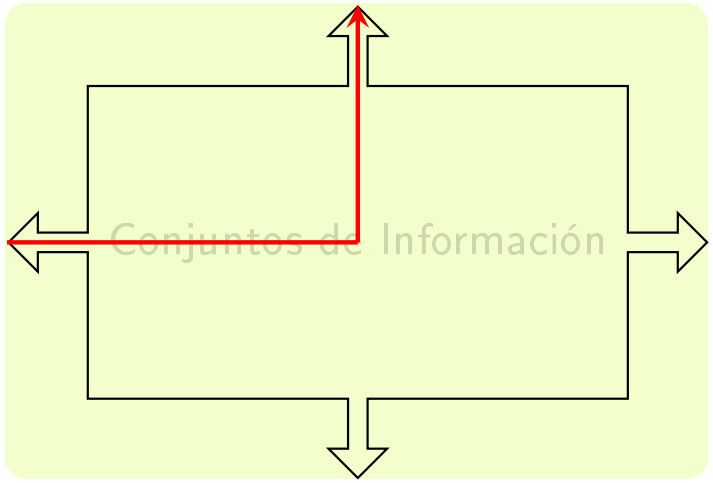
- Las distintas condiciones que incluye una alternativa pueden estar explícitas en el mismo problema que debemos resolver por ej: 'dado un entero ingresado por el usuario si el valor del entero es par incrementarlo, si es impar decrementarlo',




- O pueden ser parte de la solución matemática de un problema. Ejemplo al resolver las raíces de un función cuadrática.

# Alternativas en etapa de Análisis

- Un conjunto de datos tiene estructura alternativa si comprende al menos un subconjunto presente 0 o 1 vez. La presencia de cada uno debe ser exclusiva de la de todos los demás, para que el conjunto sea llamado de estructura alternativa. [1]




1. La organización del programa viene determinada por la organización de los datos de entrada.
2. Regla: A estructura alternativa de datos de entrada, estructura alternativa del programa.

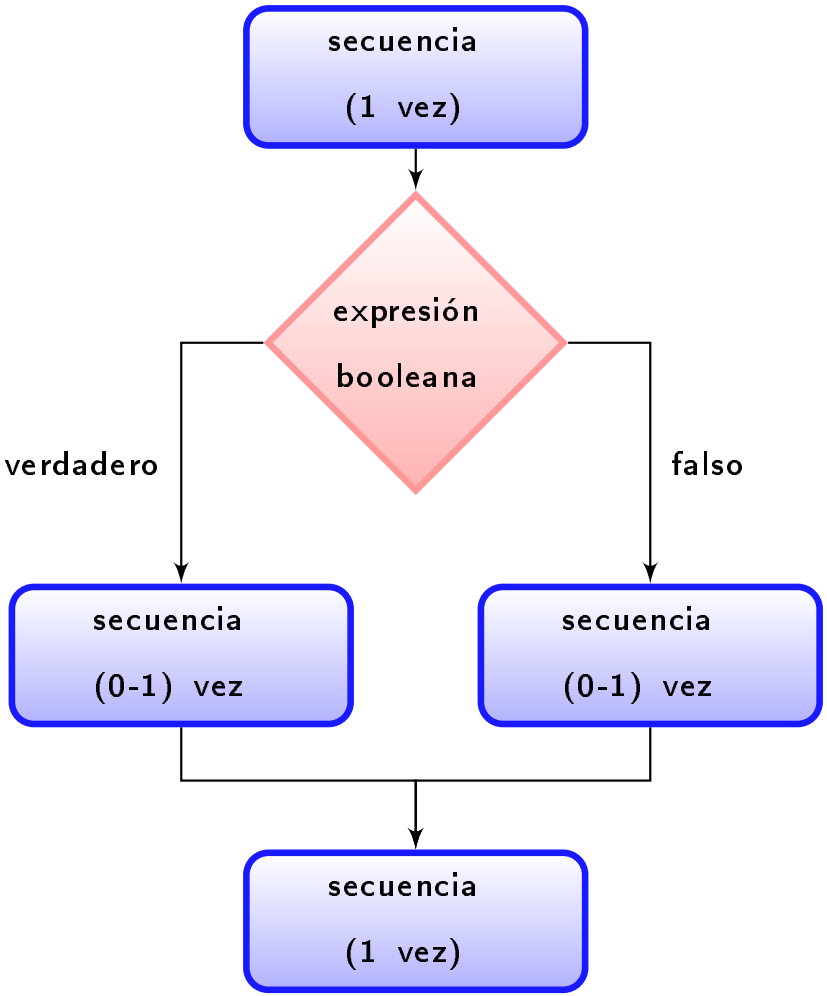
# Temario

- Veremos dos representaciones gráficas en la etapa de diseño: organigramas y pseudocódigo. La primera es una representación del tipo Diagrama.

- Un organigrama es una representación semigráfica de un algoritmo y facilita la visión descriptiva de la ejecución del programa.
- Los organigramas, muestran una versión clara de flujo de control del algoritmo.
- Un organigrama es útil para comprender los componentes principales de una alternativa. A veces pueden resultar engorrosos de graficar (en el apunte tenemos un ejemplo).

# Alternativas en la etapa de Diseño.

## Organigramas

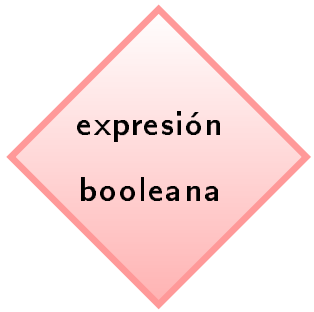



- Un subconjunto del programa de estructura alternativa comprende siempre dos o más subconjuntos, ejecutados 0-1 veces excluyéndose unos a otros. Estos subconjuntos están precedidos obligatoriamante de un subconjunto principio, a ejecutar una vez en el conjunto, y seguido de un subconjunto fin, a ejecutar una vez en el conjunto.
- Nos permite representar el flujo de control.

# Alternativas en la etapa de Diseño

## Organigramas

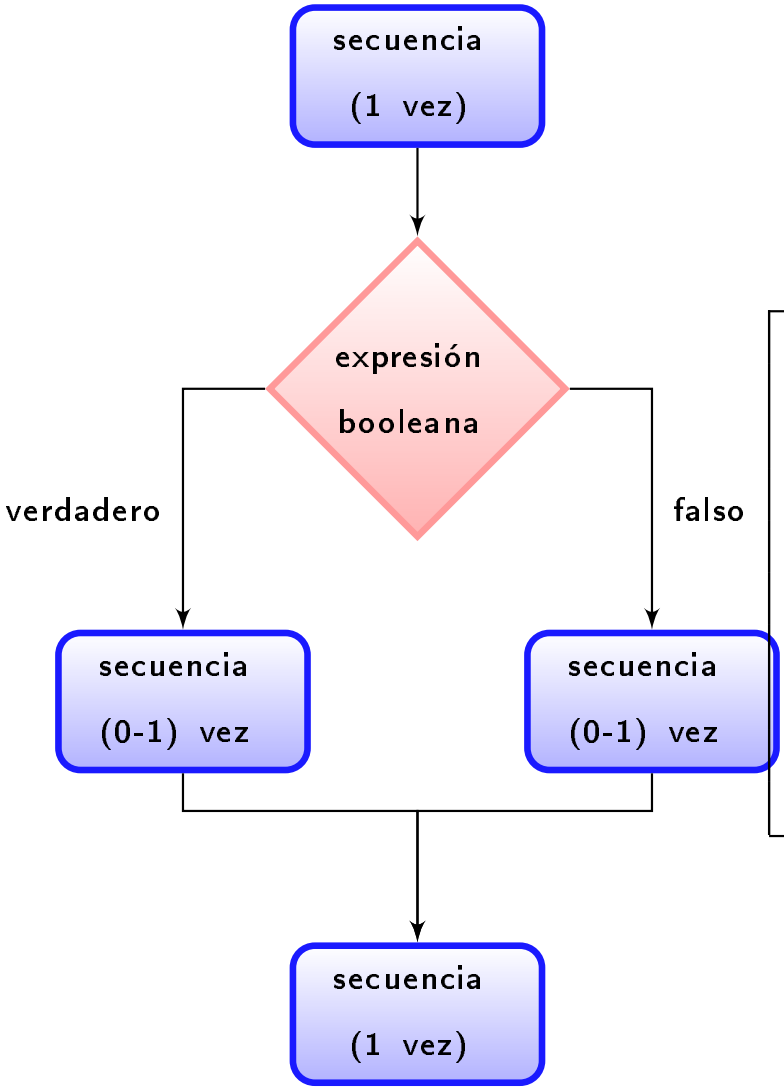
- Una expresión booleana permite determinar la condición que seleccionará una secuencia de sentencias u otra. La expresión booleana se muestra a partir de un rombo en el organigrama.




  - Si la expresión booleana se evalúa a verdadero se ejecuta la secuencia de sentencias de la izquierda (graficado con un rectángulo azul de puntas redondeadas).
  - Si la expresión booleana se evalúa a falso se ejecuta la secuencia de sentencias de la derecha.

# Alternativas en la etapa de Diseño

## Organigramas y Pseudocódigo




```
MÓDULO ....
        |   ....
        |   SI <condicion > ENTONCES
        |           | < Hacer algo si condición es True > )
        |   SINO
        |           | < Hacer algo si la condición es false > )
        |   FIN SI
        |   ...
FIN MÓDULO ....
```

# Alternativas en la etapa de Diseño

## Pseudocódigo (alternativas Simples)

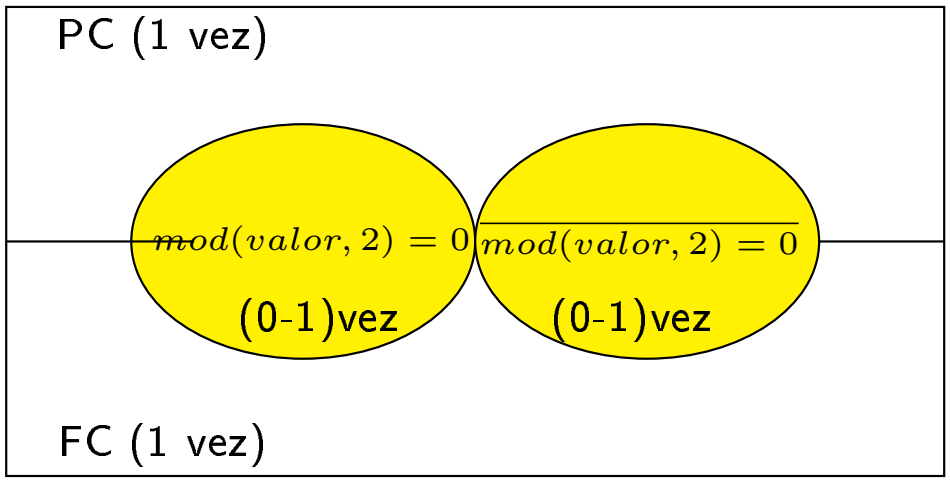
```
MÓDULO ....
        |
        | ....
        | SI <condicion > ENTONCES
        |         | < Hacer algo si la condición es True > )
        | FIN SI
        | ...

FIN MÓDULO ....
```

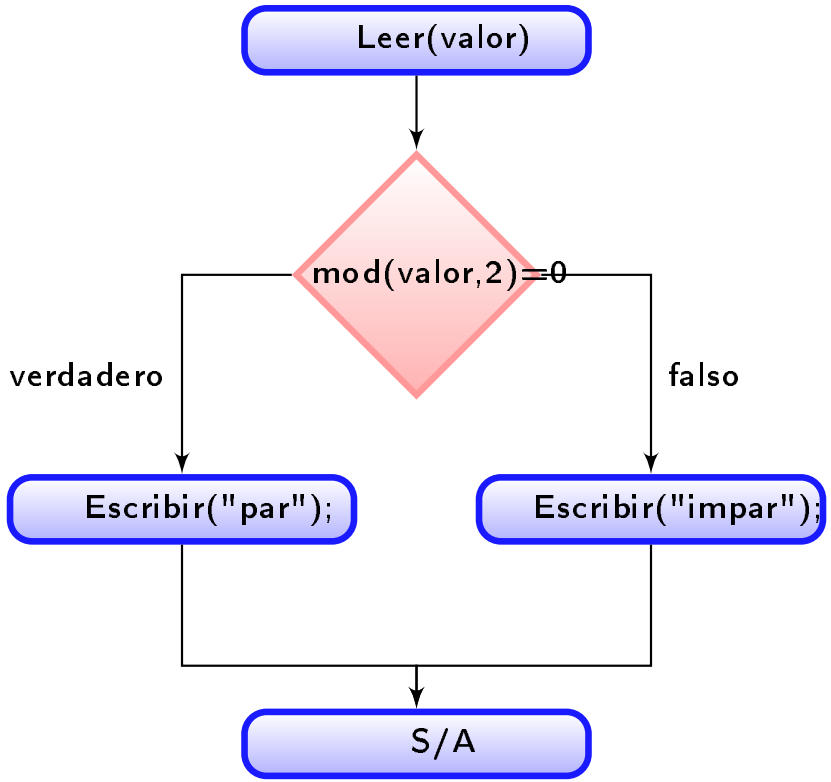
# Aternativas en la etapa de Diseño. Ejemplo.

**Ejemplo 1**: Determinar si un número entero ingresado por el usuario es par o impar

Datos de Entrada { Valor:numero

Dato de Salida { cartel "es par"
cartel "es impar"



# Aternativas en la etapa de Diseño. Ejemplo.

```
ALGORITMO esPar() RETORNA ∅
        | (* Determina si un entero es par *)
        | ENTERO x
        | ESCRIBIR("Ingrese un numero entero")
        | LEER(x)
        | SI ( x % 2 == 0 ) ENTONCES
        |         | ESCRIBIR(x + "Es un numero par")
        | ELSE
        |         | ESCRIBIR(x + "Es un numero impar")
        | FIN SI

FIN ALGORITMO esPar
```

# Temario

# Alternativas en Java. Ejemplo

- La sentencia if-else de Java permite ejecutar distintos conjuntos de sentencias según algún criterio.
- La sintáxis de la sentencia If-else es:

```
1  if ( condición )
2    Secuencia de código a ejecutar si la condición es cierta
3  else
4  Secuencia de código a ejecutar si la condición es falsa
```

- La parte del else es opcional, y una secuencia de código puede ser simplemente la sentencia vacía (cuya sintaxis es el punto y coma ";") para representar que en ese caso no se ha de ejecutar nada.
- Ejemplo: Supongamos que un programa debe realizar diferentes acciones dependiendo del valor de una variable. El programa puede realizar esto usando la sentencia If-else.

# Alternativas en Java. Ejemplo.

Código JAVA correspondiente al **Ejemplo 1** (El ejemplo solicitaba determinar si un número entero era *par* o *impar*)

```
1  public class comparo
2  {
3  public static void main(String[] args)
4  {
5  // definición de variables
6  int valor;
7  Scanner sc = Scanner(System.in);
8  // ingreso de datos
9  System.out.println ("Ingrese un numero entero");
10 valor = sc.nextInt();
11 //cálculo y salida
12 if (valor % 2 == 0)
13         System.out.println (valor+"es un número PAR");
14 else
15         System.out.println (valor+"es un número IMPAR");
16 }
17 }
```

# Alternativas en Java. Ejemplo.

## Las convenciones de codificación en Java [3] de SUN recomiendan

- Utilizar cuatro espacios como unidad de indentación.
- Siempre utilizar llaves en los bloques de sentencias correspondientes al if o al else.

```
1  if (condition) //EVITELO! OMITE LAS LLAVES {}!
2  statement;
```

# Alternativas en Java. Calidad.

## Las convenciones de codificación en Java [3] de SUN recomiendan

- Las sentencias if-else deberían tener la siguiente forma:

```
1
2   if (condition) {
3       statements;
4   }
5
6   if (condition) {
7       statements;
8   } else {
9       statements;
10  }
```

## Importante!

- En NetBeans la combinación de teclas Alt+Mayúscula+F permite adaptar el código del programa a un formato que utiliza estas convenciones.
- Si no recordamos la combinación, la opción se puede seleccionar desde el menú: Source->Format.

# Temario

# Alternativas en Etapa de Revisión.

- Revisar cuidadosamente si la condición booleana es correcta.
- Revisar si las variable booleanas utilizadas en la condición han sido inicializadas previamente.
- Revisar si efectivamente estamos implementando una disyunción exclusiva con la condición boolena (i.e. $\oplus$).
- Realizar diferentes trazas del programa para verificar que la condición booleana de la alternativa puede asumir los valores verdadero o falso, y el programa se comporta como esperamos.
- Verificar que la secuencia de sentencias de cada bloque, correspondiente al if y al then son los adecuados.
- Verificar que el programa luego de la ejecución de la alternativa utiliza el producto resultante de haber ejecutado esta estructura.
- Importante! Simplificar lo máximo posible la condición booleana. Mas adelante veremos algunas técnicas utilizando otros diagramas.

# Temario

# Alternativas Anidadas

Se pueden anidar expresiones If-else, para poder implementar aquellos casos con múltiples acciones.

## Importante sobre Calidad!

- Al anidar alternativas no olvidemos la indentación ya que es más clara para el lector y el programador.
- Recordemos que dificilmente el software es mantenido durante toda su vida por el autor original [3]
- Las sentencias if-else anidadas deberían tener la siguiente forma:

```
1  if (condition) {
2       statements;
3  } else if (condition) {
4            statements;
5          } else if (condition) {
6               statements;
7               }
```

# Alternativas Anidadas. Ejemplo

**Ejemplo 1:** Supongamos que se desea escribir un programa que según el contenido de una variable llamada *valor*, asigne una letra a una variable llamada clasificación. Si el valor está en el rango 100-91 la letra es A, B si el rango es de 90-81, C para 80-71 y F si no es ninguno de los anteriores:

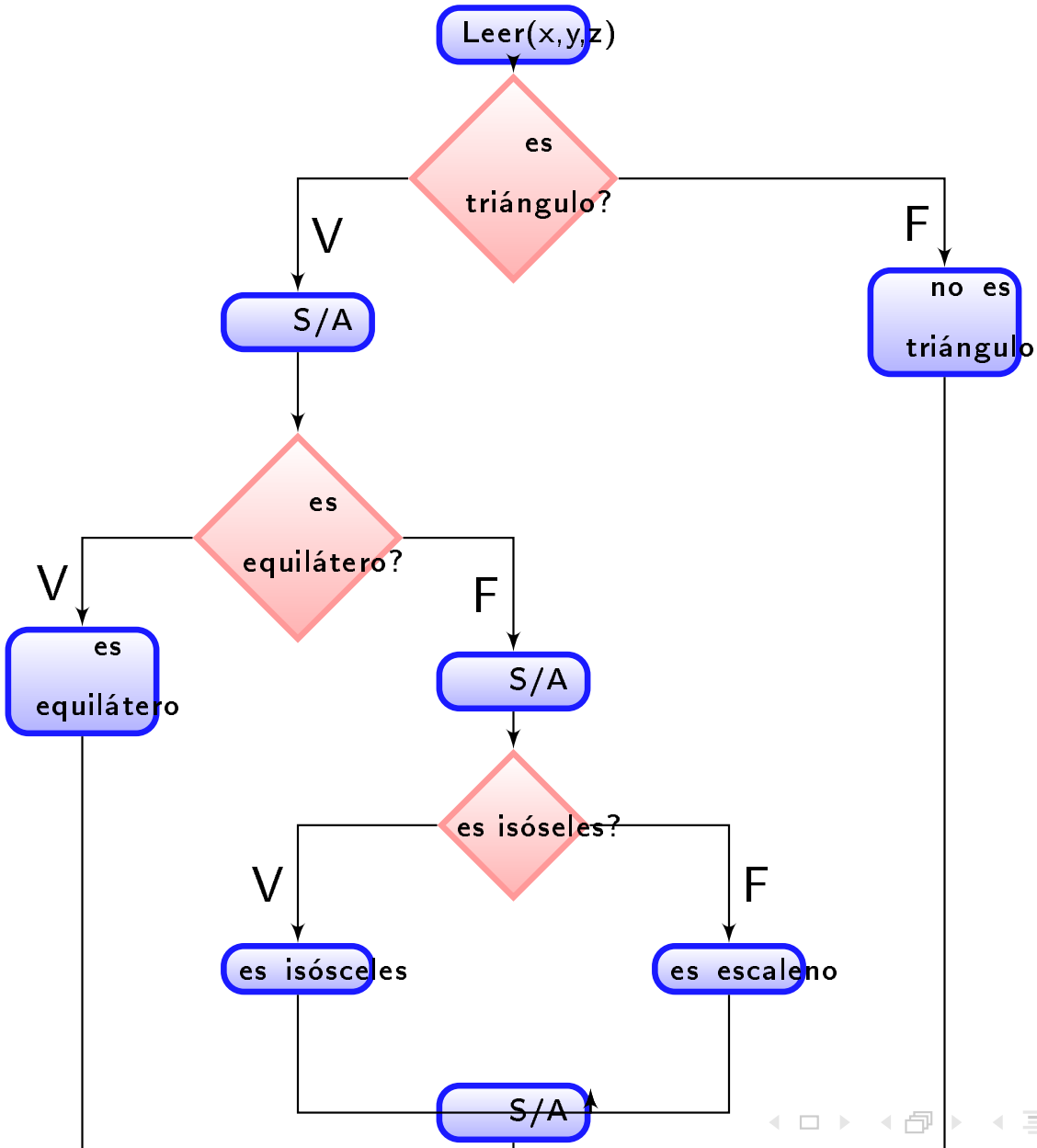
```
1  int valor;
2  char clasificacion;
3  Scanner sc = new Scanner(System.in);
4  valor = sc.nextInt();
5  if (valor > 90 && valor < 101){
6      clasificacion= A ;
7  } else if (valor > 80  && valor < 91) {
8      clasificacion= B ;
9      }
10     else if (valor > 70  && valor < 81) {
11         clasificacion= C ;
12         } else {
13         clasificacion= F ;
14         }
```

# Alternativas Anidadas. Ejemplo

**Ejemplo 2:**

- Diseñe un programa que lea tres longitudes y determine si forman o no un triángulo.
- Si es un triángulo determine de qué tipo de triángulo se trata entre: equilátero (si tiene tres lados iguales), isósceles (si tiene dos lados iguales) o escaleno (si tiene tres lados desiguales).
- Considere que para formar un triángulo se requiere que: 'el lado mayor sea menor que la suma de los otros dos lados'.

# Alternativas Anidadas. Ejemplo



# Alternativas Anidadas. Ejemplo

```
 1  public class Triangulo
 2  {
 3  public static void main(String[] args)
 4  {
 5  int x, y, z
 6  Scanner sc = new Scanner(System.in);
 7  System.out.println("Ingrese las longitudes");
 8  x = sc.nextInt();
 9  y = sc.nextInt();
10  z = sc.nextInt();
11
12  if ((x<y+z) || (y<x+z) || (z<x+y)) {
13     if ((x == y) && (y == z)) {
14       System.out.println("Es un triangulo equilatero");
15       } else if ((x == y) || (x == z) || (y == z)){
16            System.out.println("Es un triangulo isosceles");
17            } else {
18            System.out.println("Es un triangulo escaleno");
19          }
20       } else {
21       System.out.println("No es un triangulo");
22       }
```

# Temario

# Sentencia Condicional SWITCH

- Mediante la sentencia switch se puede seleccionar entre varias sentencias según el valor de cierta expresión.
- La forma general de switch es al siguiente:

```
1  switch ( expresionMultivalor ) {
2  case valor1 : conjuntoDeSentencias; break;
3  case valor2 : conjuntoDeSentencias; break;
4  case valor3: conjuntoDeSentencias; break;
5  default: conjuntoDeSentencias; break;
6  }
```

- La sentencia switch evalúa la *ExpresionMultivalor* y ejecuta el *ConjuntoDeSentencias* que aparece junto a la cláusula case cuyo valor corresponda con el valor de la *ExpresionMultivalor*.
- Cada sentencia case debe ser única y el *valor* que evalúa debe ser del mismo tipo que el devuelto por la *ExpresionMultivalor* de la sentencia switch.

# Sentencia Condicional SWITCH

- Las sentencias break que aparecen tras cada *ConjuntoDeSentencias* provocan que el control salga del switch y continúe con la siguiente instrucción al switch.
- Las sentencias break son necesarias porque sin ellas se ejecutarían secuencialmente las sentencias case siguientes.
- Existen ciertas situaciones en las que se desea ejecutar secuencialmente algunas o todas las sentencias case, para lo que habrá que eliminar algunos break.
- Finalmente, se puede usar la sentencia default para manejar los valores que no son explícitamente contemplados por alguna de las sentencias case. Su uso es altamente recomendado.

# Sentencia Condicional SWITCH

Por ejemplo, supongamos un programa con una variable entera meses cuyo valor indica el mes actual, y se desea imprimir el nombre del mes en que estemos. Se puede utilizar la sentencia switch para realizar esta operación:

```
1  int meses;
2  switch ( meses ){
3  case 1: System.out.println( "Enero" ); break;
4  case 2: System.out.println( "Febrero" ); break;
5  case 3: System.out.println( "Marzo" ); break;
6  //Demas meses
7  // . . .
8  case 12: System.out.println( "Diciembre" ); break;
9  default: System.out.println( "Mes no valido" ); break;
10 }
```

# Sentencia Condicional SWITCH

1. Por supuesto, se puede implementar esta estructura como una sentencia if else if: int meses;

```
1  if ( meses == 1 ) {
2  System.out.println( "Enero" );}
3  else
4  if ( meses == 2 ) {
5  System.out.println( "Febrero" );
6  }
7  // Y así para los demás meses
```

2. El decidir si usar la sentencia if o switch depende del criterio de cada caso. Se puede decidir cuál usar basándonos en la legibilidad, aunque se recomienda utilizar switch para sentencias con más de tres o cuatro posibilidades.

# Lectura Adicional


Jean-Dominique Warnier.
*Síntesis de Programación Lógica. Los Tratamientos y sus Datos.*
Editores Técnicos Asociados, S.A.
Barcelona

Apuntes de Alternativas.
Apunte sobre Tipos Primitivos
*Facultad de Informática*, Universidad del Comahue, 2021.

Sun
*Code Convention for the JAVA Programming Language*
Sun Microsystems, Inc. 1997

James Gosling, Bill Joy, Guy Steele, Gilad Bracha.
*The Java Language Specification Third Edition.*
Addison-Wesley, 1996-2005.
http://java.sun.com/docs/books/jls/