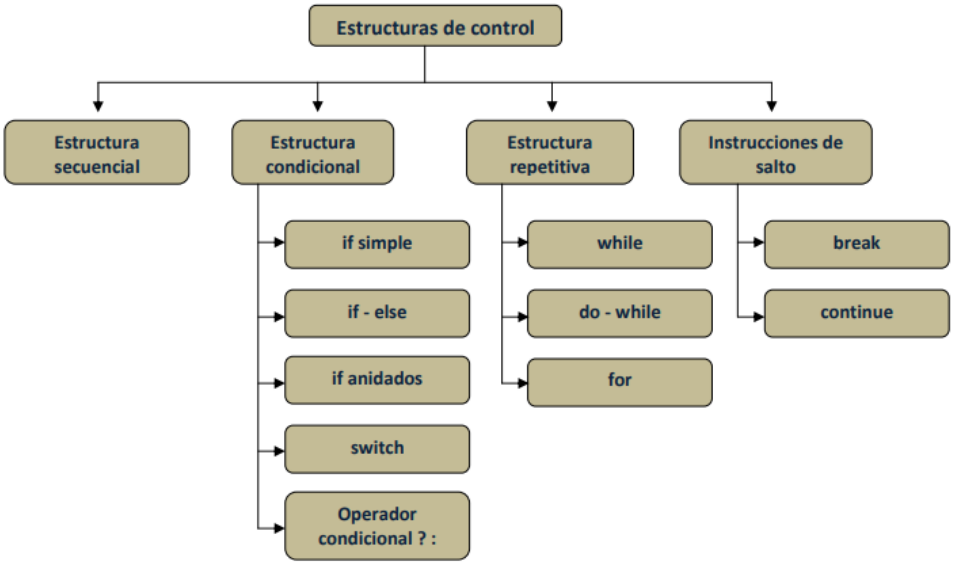
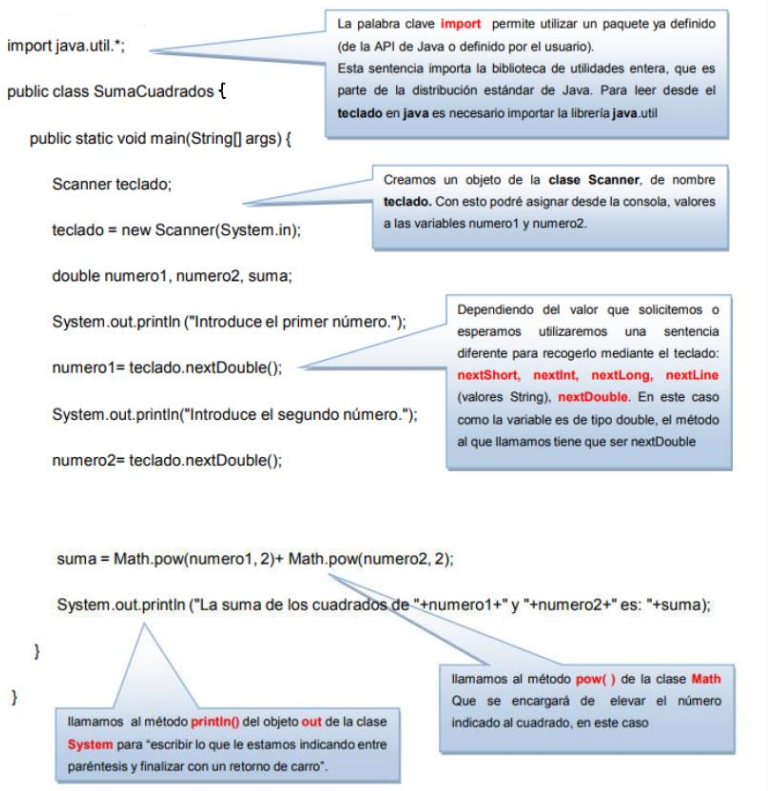
ESTRUCTURAS DE CONTROL EN JAVA

Los programas contienen instrucciones que se ejecutan una a continuación de la otra, siguiendo el orden/secuencia en la que las hemos escrito. Desde siempre, en todos los lenguajes y ya desde los tiempos de la programación estructurada, en muchas ocasiones es necesario romper esa secuencia de ejecución "lineal" descendente para crear bifurcaciones en los programas que permitan varios caminos posibles **(Estructuras condicionales).** También es necesario a menudo hacer que una serie de instrucciones se repitan un número determinado de veces **(Estructuras repetitivas o bucles).**



Por defecto, las instrucciones de un programa se ejecutan en **orden secuencial**, una detrás de otra, según aparecen escritas dentro del programa. Las instrucciones se agrupan en bloques de código entre corchetes **{}**



En la figura vemos un ejemplo de programa secuencial para una suma de cuadrados: **numero1^2 + numero 2^2**
Es importante fijarse sobre todo en el uso del paquete **java.util** de la librería de JAVA **(API),** en cómo se usa una librería **(import),** y también en cómo se utilizan las clases **Scanner, Math** y sus métodos asociados.
No importa sino entendemos por ahora algunas cosas, pues comenzaremos desde ya a hacer nuestros propios programas sencillos para ir aprendiendo poco a poco. Y lo que es más importante, los editaremos y probaremos uno a uno, al principio en entornos sencillos como **BlueJ**, y según avance el curso en IDES profesionales como **Netbeans y Eclipse.**

Ejemplo (1)

```
/*
  Programa que pide un nombre (string) por teclado y lo muestra por pantalla formando un mensaje.
*/
        import java.util.Scanner;  // importa las clase Scanner del paquete java.util
        public class EjemSecuencial1 {
                public static void main(String[] args) {
                Scanner sc=new Scanner(System.in);
                System.out.println("Hola.");
                System.out.println("¿Cómo te llamas?");

                String miNombre=sc.nextLine();
                System.out.println("Así que te llamas "+ miNombre);
                sc.close();
            }
        }
```

Ejemplo (2) de programa Java con estructura secuencial: programa que lee dos números por teclado y los muestra por pantalla. De nuevo hay un único bloque de instrucciones contenidas entre las llaves de apertura y cierre del método main. Las instrucciones se ejecutan en el mismo orden que aparecen escritas.

```
/*
  Programa que lee dos números por teclado y los muestra por pantalla.
*/
import java.util.*;  // importa las clases del paquete java.util, entre ellas la clase Scanner
public class EjemSecuencial2 {
   public static void main(String[] args){
     //declaración de variables
      int n1, n2;
      Scanner sc = new Scanner(System.in);
      //lee el primer número por teclado
      System.out.println("Introduce un número entero: ");
      n1 = sc.nextInt();
      //lee el segundo número por teclado
      System.out.println("Introduce otro número entero: ");
      n2 = sc.nextInt();
      //mostrar resultado
      System.out.println("Ha introducido los números: " + n1 + " y " + n2);
   }
}
```

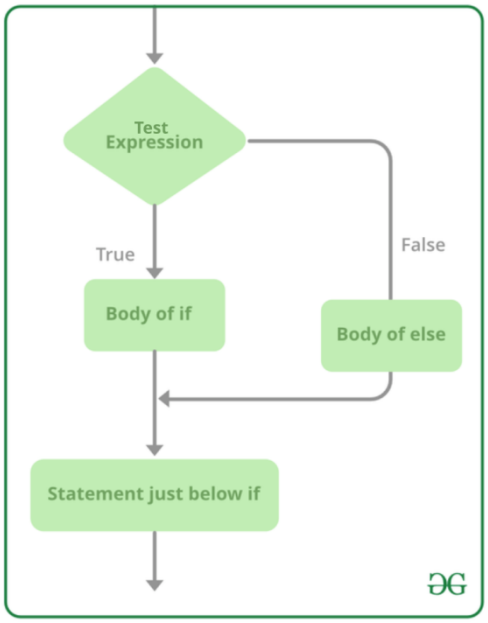
Ejemplo (3) de programa Java con estructura secuencial: programa que lee por teclado dos valores de tipo double y a continuación muestra su suma, resta y multiplicación. De nuevo el programa también está formado por un único bloque de instrucciones que se ejecutan secuencialmente dentro del método main.

```
/*
 Mostrar la suma, resta y multiplicación de dos valores de tipo double que se introducen por teclado.
 */
import java.util.*;
public class EjemSecuencial3{
   public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      double numero1, numero2;
      System.out.println("Introduce el primer número:");
      numero1 = sc.nextDouble();
      System.out.println("Introduce el segundo número:");
      numero2 = sc.nextDouble();
      System.out.println("Los números introducidos son: " + numero1 + "  " + numero2);
      System.out.println("La suma de: " + numero1 + " + " + numero2 + " = " + (numero1+numero2));
      System.out.println("La resta de: " + numero1 + " - " + numero2 + " = " + (numero1-numero2));
      System.out.println("La multiplicación de: " + numero1 + " * " + numero2 + " = " + numero1*numero2);
    }
}
```

ESTRUCTURAS CONDICIONALES.

Permiten modificar el orden de ejecución de las instrucciones del programa creando bifurcaciones o caminos alternativos de ejecución, que dependerán del cumplimiento o no de una **CONDICIÓN**. De ahí el nombre de **estructuras condicionales**. La condición que se comprueba para decidir si se toma un camino u otro debe ser una expresión condicional booleana, cuyo resultado será true/false.

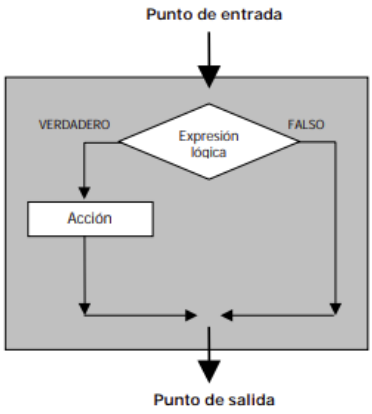
De modo genérico, diremos que el programa llega a un punto en el que se evalúa una condición booleana, y en ese punto se bifurca, tomando un camino u otro según sea el resultado de evaluar la condición.



Nos podemos encontramos varias **alternativas**:

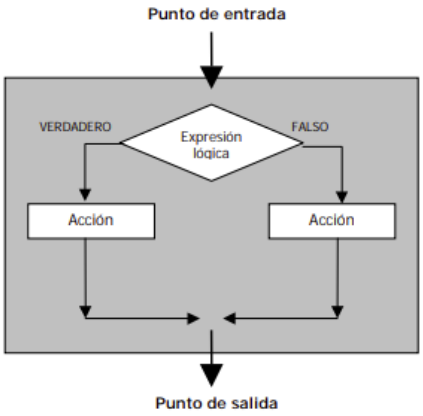
CONDICIONAL SIMPLE – if

```
if (condicion) {
 // Bloque de sentencias – condición true
}
```



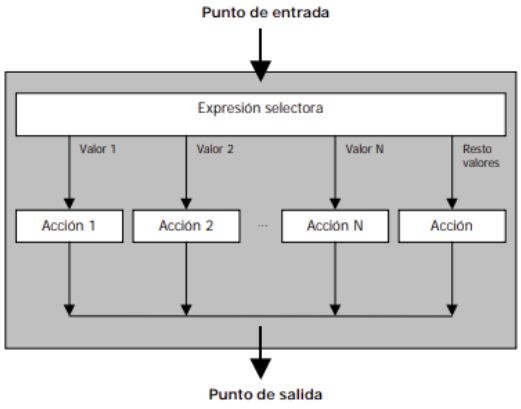
CONDICIONAL DOBLE – if … else

```
if (condicion) {
 //Bloque de sentencias - condición true
 }else {
 //Bloque de sentencias - condición false
 }
```



CONDICIONAL MÚLTIPLE – switch

```
switch (variable) {
   case valor1:
        // Bloque de sentencias 1
        break;
   case valor2:
        // Bloque de sentencias 2
        break;
   case valor3:
        // Bloque de sentencias 3
        break;
   default:
        // Bloque de sentencias
   }
```



Ejemplo (4) – Estructura condicional SIMPLE: Programa que pide que se introduzca una nota por teclado y muestra mensajes en caso de Aprobado.

```
/*
  Programa que pide una nota por teclado y muestra  mensaje en caso de Aprobado
 */
import java.util.*;
public class CondicionalSimple1 {
	public static void main( String[] args ){
		Scanner sc = new Scanner( System.in );
		System.out.print("Cuál es la Nota: ");
		int nota = sc.nextInt();
		if (nota >= 5){			// inicio de la condición
			System.out.println("Enhorabuena, has aprobado!!");
		}				// fin de la condición
		System.out.println("Hasta Pronto!");
	}
}
```

Si un bloque de instrucciones contiene una sola instrucción no sería necesario escribir las llaves { }, aunque para evitar confusiones se recomienda escribir las llaves siempre.

```
import java.util.*;
public class CondicionalSimple1 {
	public static void main( String[] args ){
		Scanner sc = new Scanner( System.in );
		System.out.print("Cuál es la Nota: ");
		int nota = sc.nextInt();
		if (nota >= 5)
			System.out.println("Enhorabuena, has aprobado!!");
		System.out.println("Hasta Pronto!");
	}
}
```

No escribir las llaves en estos casos puede conducir a un código confuso de entender, por lo que es siempre recomendable escribir las llaves incluso en los casos en los que no "parezca" necesario. Incluso se podrían haber escrito las 2 instrucciones de salida al mismo nivel de tabulado. No hay errores de compilación, pero el bloque del if **no se ve con claridad, y el código se hace difícil de entender**.

```
	if (nota >= 5)
	System.out.println("Enhorabuena, has aprobado!!");
	System.out.println("Hasta Pronto!");
```

Ejemplo (5) – Estructura condicional DOBLE: Programa que lee una nota por teclado y muestra un mensaje indicando si se ha aprobado o no.

```
/*
 * Programa que pide una nota por teclado y muestra si se ha aprobado o no
 */
```

```
import java.util.*;

public class CondicionalDoble1{
	public static void main( String[] args ){
		Scanner sc = new Scanner(System.in);
		System.out.print("Nota: ");
		int nota = sc.nextInt();
		if (nota >= 5){
			System.out.println("Enhorabuena!!");
			System.out.println("Has aprobado");
		} else {
			System.out.println("Lo siento, has suspendido");
		}
		System.out.println("Hasta Pronto!");
	}
}
```

Ejemplo (6) – Estructura condicional DOBLE: Programa que lee un número entero por teclado y muestra un mensaje indicando si el número es par o impar.

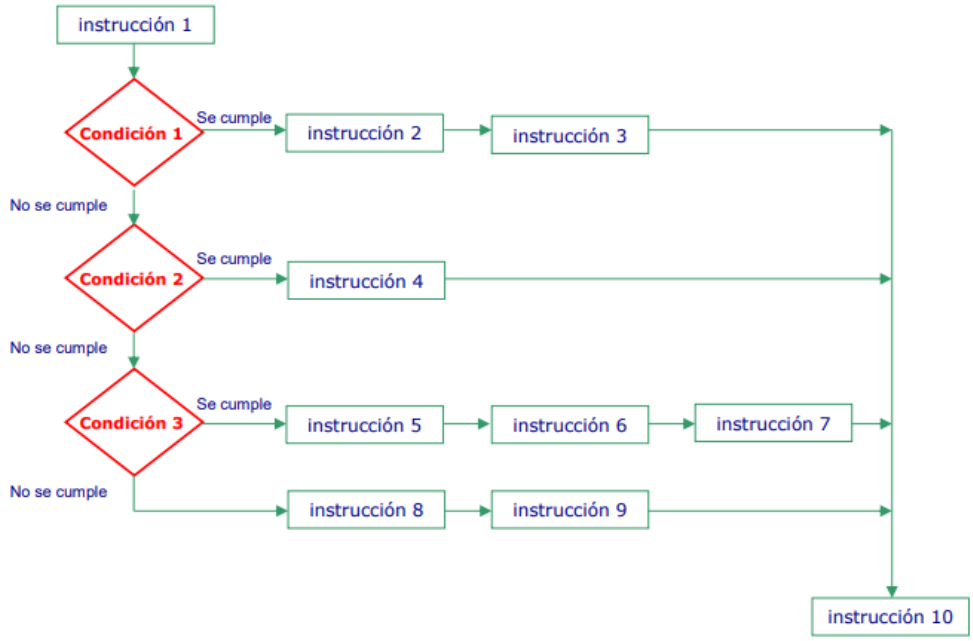
```
/*
 * Programa que pide un número por teclado y muestra si es par o impar
 */
import java.util.*;

public class CondicionalDoble2 {

	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int num;
		System.out.print("Introduzca un número entero: ");
		num = sc.nextInt();
		if (num % 2 == 0) {
			System.out.println("El número " + num+ " ES PAR");
		} else {
			System.out.println("El número " + num+ " ES IMPAR");
		}
	}
}
```

Ejemplo (7) – Estructura condicional ANIDADA: Programa que solicita por teclado una hora (número entero) y muestra un mensaje de saludo por pantalla según la hora introducida corresponda a la mañana, la tarde o la noche.

Un ejemplo de *if* múltiple ANIDADO expresado en diagrama de flujo puede ser este:



Aplicado al ejemplo de la Hora en Java quedaría:

```
//Programa que muestra un saludo distinto según la hora introducida
import java.util.*;
public class CondicionalAnidado1{
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		int hora;
		System.out.print("Introduzca una hora (>= 0 y <= 23): ");
		hora = sc.nextInt();
		if (hora >= 0 && hora < 12) {
			System.out.println("Buenos días");
		} else if (hora < 21) {
			System.out.println("Buenas tardes");
		} else if (hora < 24) {
			System.out.println("Buenas noches");
		} else {
			System.out.println("Hora no válida");
		}
	}
}
```

Ejemplo (8) – Estructura condicional ANIDADA 2: Programa que pide que se introduzca la nota de un alumno en una asignatura, y muestra un texto con la calificación equivalente de la siguiente forma:

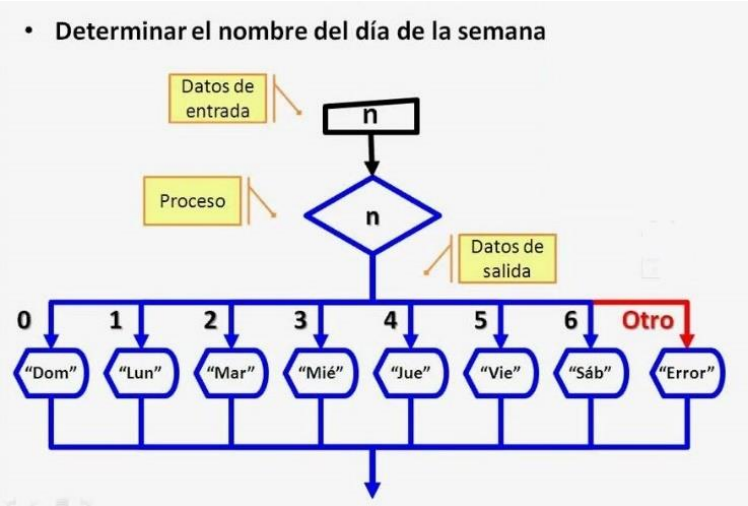
<5	"Suspenso"
>=5 y <6	"Suficiente"
>= 6 y <7	"Bien"
>=7 y <9	"Notable"
>=9 y <10	"Sobresaliente"
= 10	"Matrícula de honor"
<10 ó >10	"Nota no válida"

```
// programa que lee una nota y escribe la calificación correspondiente
```

```
import java.util.*;
public class CondicionalAnidado2 {
   public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Introduzca una nota entre 0 y 10: ");
        double nota = sc.nextDouble();
        System.out.println("La calificación del alumno es ");
        if (nota < 0 || nota > 10) {
              System.out.println("Nota no válida");
        } else if (nota == 10) {
                    System.out.println("Matrícula de Honor");
              } else if (nota >= 9) {
                          System.out.println("Sobresaliente");
                    } else if (nota >= 7) {
                                System.out.println("Notable");
                          } else if (nota >= 6) {
                                      System.out.println("Bien");
                                } else if (nota >= 5) {
                                            System.out.println("Suficiente");
                                      } else {
                                                System.out.println("Suspenso");
                                      }
      }
}
```

Ejemplo (9) – Estructura condicional de Selección MÚLTIPLE : se utiliza para seleccionar una de entre **múltiples** opciones posibles. **Es una alternativa práctica y sencilla a los if .. else anidados**. El flujo de ejecución del programa lo determina el valor de una variable o expresión. El tipo de esta variable o expresión puede ser:

- Tipo primitivo: byte, short, char, int. **(ni float ni double están permitidos)**
- La clase envolvente de los tipos primitivos anteriores: Byte, Short, Character, Integer.
- Tipo String o tipos enumerados (enum).



Se evalúa la expresión **n,** y el flujo del programa toma la "rama" que coincida con el valor de la expresión **n**. Para cada bloque case se ejecutan las instrucciones que contiene el bloque, hasta que se encuentra un ***break*** o hasta el final del ***switch***. El ***break*** produce la **salida** y ejecución a partir de la siguiente línea tras el *switch*. Si el valor de la expresión no coincide con ningún *case* se ejecuta el bloque ***default*** (opcional).

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

```
public class SeleccionMultiple1 {
	public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.print("Introduzca un numero de día de la semana: ");
		int dia = sc.nextInt();
		switch (dia) {
		case 0:
		   System.out.println("DOMINGO");
		    break;
		case 1:
		   System.out.println("LUNES");
		    break;
		case 2:
		   System.out.println("MARTES");
		    break;
		case 3:
		   System.out.println("MIERCOLES");
		    break;
		case 4:
		   System.out.println("JUEVES");
		    break;
		case 5:
		   System.out.println("VIERNES");
		    break;
		case 6:
		   System.out.println("SABADO");
		    break;
		default :
		   System.out.println("DÍA NO VÁLIDO");
	}
   }
}
```

Ejemplo (10). Estructura de selección múltiple (2): Programa que lee una nota y escribe la calificación correspondiente (Nueva versión). La diferencia entre esta versión y la realizada con **IF ANIDADOS** es que la nueva versión con **Switch – Case** tan sólo serviría para notas enteras, sin decimales, pues cómo hemos comentado no se pueden utilizar variables **float/double** para controlar una estructura Switch.

```
import java.util.*;
public class SeleccionMultiple2 {
    public static void main(String[] args) {
		Scanner sc = new Scanner(System.in);
		System.out.println("Introduzca una nota entera entre 0 y 10: ");
		int nota = sc.nextInt();
		System.out.println("La calificación del alumno es ");
		switch (nota) {
		case 0:
		case 1:
		case 2:
		case 3:
		case 4:
		   System.out.println("SUSPENSO");
		    break;
		case 5:
		   System.out.println("SUFICIENTE");
		    break;
```

```
            case 6:
               System.out.println("BIEN");
                break;
            case 7:
            case 8:
               System.out.println("NOTABLE");
                break;
            case 9:
            case 10:
               System.out.println("SOBRESALIENTE");
                break;
            default :
               System.out.println("NOTA NO VÁLIDA");
            }
   }
}
```

Un **BLOQUE *case*** puede estar vacío, contener 1 o varias instrucciones que se ejecuten en secuencia, y también estructuras más complejas como condiciones, bucles o incluso otro ***switch***.

Ejemplo (11). Estructura de selección múltiple (3): programa que pide por teclado dos números enteros y un operador +, -, *, / y muestra el resultado de la operación según el operador introducido. En caso de división, se comprueba si el divisor es cero, en ese caso la división no se puede realizar y se muestra un mensaje.

```
//Programa que pide dos números enteros y un operador y muestra el resultado de la operación

import java.util.*;
public class SeleccionMultiple3 {

        public static void main(String[] args) {
            int num1, num2, resultado = 0 ;
            char operador;
            boolean calculado = true;
            Scanner sc = new Scanner(System.in);
            System.out.print("Teclea el primer operando:");
            num1 = sc.nextInt();
            System.out.print("Teclea el segundo operando:");
            num2 = sc.nextInt();
            System.out.print("¿Qué operador quieres utilizar? (+,-,*,/):");
           operador = sc.next().charAt(0);
```

/* IMPORTANTE – No hay un método en la clase Scanner para leer un char por teclado, así que para leer un char recurrimos a los métodos de lectura de cadenas y tomamos sólo el primer carácter de la cadena leída - charAt(0)
ADEMÁS, en este ejemplo usamos next() en vez de nextLine() por culpa del funcionamiento característico de los nextInt anteriores. nextInt() lee el entero que tecleamos, pero deja en el "buffer" de teclado el carácter de salto de línea "/n" que hemos tecleado al hacer Return. De esta forma, como nextLine lee lo que hay en el buffer de teclado hasta que encuentra un "/n" … se contenta con el "residuo" del anterior nextInt(), carga una cadena vacía y continúa hacía delante sin hacer lo que nosotros esperamos que haga (anómalo). La solución es hacer un nextLine(); después del último nextInt() para consumir ese "/n" que el nextInt() dejó en el buffer. La otra opción es hacer como en este ejemplo, usar un next(), alternativa a nextLine pero que sirve no para recoger por teclado palabras sueltas, no cadenas enteras, pues lo que busca no es un "/n" sino un espacio en blanco para terminar. De ahí que no le afecte el "/n" que dejó en último nextInt() y que en este ejemplo funcione perfectamente */

```
            switch (operador) {
                case '-' :
                    resultado = num1 - num2;
                    break;
                case '+' :
                    resultado = num1 + num2;
                    break;
                case '*' :
                    resultado = num1 * num2;
                    break;
                case '/' :
                    if(num2!=0){
                        resultado = num1 / num2;
                    }else{
                        System.out.println("\nNo se puede dividir por cero");
                        calculado = false;
                    }
                    break;
                default :
                    {
                        System.out.println("\nOperador no valido");
                        calculado = false;
                    }
             }
            if(calculado){
                System.out.println("\nEl resultado es: " + resultado);
            }
        }
}
```

Es **importante** entender que una vez que la ejecución del programa entra en uno de los *case*, el programa sigue ejecutándose desde ese punto hasta que encuentre el primer *break* (estén dentro del mismo *case* o no) o hasta que se llegue al final del *switch* en cuyo caso continuará por la instrucción que se encuentre a continuación del *switch*.

OPERADOR CONDICIONAL TERNARIO ? :

El operador condicional ternario se puede utilizar en sustitución de la instrucción condicional ***if-else***.

expresión1 **?** expresión2 **:** expresión3 **Equivale a:**

```
if (expresión1){
                  expresión2
                } else {
                  expresión3
                }
```

Ejemplo (12): programa que pide por teclado un número entero y muestra si es positivo o negativo. Consideramos el cero como positivo.

```
import java.util.*;
public class OperadorTernario {
  public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      System.out.println("Introduzca numero: ");
      int num = sc.nextInt();
      System.out.println(num >= 0 ? "POSITIVO" : "NEGATIVO");
    //usamos el operador ternario directamente en el método println
   }
}
```

La línea *System.out.println(num >= 0 ? "POSITIVO" : "NEGATIVO");* **es equivalente a:**

```
if(num >= 0){
            System.out.println("POSITIVO");
          }else{
            System.out.println("NEGATIVO");
          }
```

Ejemplo (13): programa que pide por teclado un número entero y muestra si es par o impar.

```
import java.util.*;
public class OperadorTernario2 {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.println("Introduzca numero: ");
    int num = sc.nextInt();
    System.out.println(num%2 == 0 ? "PAR" : "IMPAR");
    //Si el resto (operador %)  de dividir un número entre 2 es 0 entonces el PAR, sino es IMPAR
  }
}
```