Universidad Nacional del Litoral
Facultad de Ingeniería y Ciencias Hídricas
Departamento de Informática

Tecnicatura en Informática Aplicada al Diseño Multimedia y Sitios Web

Elementos de Programación

Unidad 4: Estructuras de Control

Docente: Ing. Risso, Oscar Luis
Año: 2024

Índice

Unidad 4: Estructuras de Control

En este Tema

Estudiaremos aquí la sintaxis de las estructuras de control en Java.

En los primeros diseños de proyectos sólo utilizamos acciones de estructura secuencial (sentencias simples de Java), es decir, los pasos o acciones indicados se ejecutan uno tras otro, a medida que van apareciendo; es decir, secuencialmente.

Pero en el diseño de algoritmos computacionales, generalmente es necesario modificar el orden secuencial de ejecución del conjunto de acciones. Por ejemplo, es común tener que tomar decisiones al efectuar un cálculo: "Si el empleado tiene asistencia perfecta, entonces cobra un premio adicional de X cantidad de pesos, si no el adicional se reduce a menos cantidad de pesos". Otro caso es tener que repetir acciones muchas veces; por ejemplo: "Ingresar los nombres y las 3 calificaciones parciales de 160 alumnos, determinar el promedio de cada uno".

Los lenguajes de programación poseen herramientas para resolver estos casos con facilidad a través de las llamadas estructuras de control.

Es fundamental que practiquemos la sintaxis resolviendo problemas. ¡No hay mejor forma de aprender la sintaxis de un lenguaje de programación que programando!

Objetivos

Que el alumno logre:

Conocer y utilizar las estructuras de control del lenguaje Java. Resolver problemas de complejidad creciente usando el modelo de objetos, y aplicando estructuras de control condicionales e iterativas.

1. Tipos de sentencias en Java

Java posee 3 tipos de sentencias: simples, compuestas y estructuradas. Estudiemos cada una de ellas.

1.1. Sentencias simples

La asignación y las llamadas a métodos o funciones son consideradas en Java como sentencias simples. (Aunque goto es una palabra reservada, actualmente el lenguaje Java no soporta la sentencia goto).

Ejemplos:

```
X = 125; //asigna 125 a la variable X
String texto = "Hola Mundo";//asigna una cadena a texto
Cilindro.Calcular; //llamar al método Calcular del objeto cilindro
System.exit( 0 ); // llama al comando exit para cerrar la aplicación
```

1.2. Sentencias compuestas

En Java, cualquier conjunto de sentencias precedida por el delimitador **{** (llave) y seguidas del delimitador final **}** (llave); constituye una sentencia compuesta.

Ejemplo:

```
{
r = Double.parseDouble(JOptionPane.showInputDialog("Radio"))
y= x*x*x;
t= (char)88;
}
```

IMPORTANTE

Al delimitar una grupo de acciones o sentencias con y se conforma una sentencia compuesta que puede tratarse en el programa como una entidad individual (como una sola sentencia).

1.3. Sentencias Estructuradas

Son sentencias que permiten alterar la secuencia lineal de ejecución en un programa. Con ellas podemos plantear alternativas o iteraciones. Son también conocidas como estructuras de control.

En Java las sentencias estructuradas son:

- Estructuras de control condicional
    1. Decisión simple if
    2. Decisión multiple switch
    3. Decisión in line (?) a>b? “a”:”b”;
- Estructuras de control iterativas
    1. while
    2. do-while
    3. for

A continuación estudiaremos a cada una de ellas.

2. Estructuras de control condicional: If / ?

La estructura condicionar if permite tomar decisiones en un algoritmo o en un programa. La sintaxis que deberemos emplear en Java es la siguiente:

```
1 if <expresión booleana>
2  { sentencias 1, por verdadero }
3  else {sentencias 2, por falso };
```

Esta sentencia indica al compilador que la expresión boolean que colocamos después del If arroja un valor verdadero (True), se debe ejecutar el bloque de sentencias 1; en caso contrario ejecutará el bloque de sentencias 2 .

IMPORTANTE

1. La alternativa por False indicada con else es opcional. Pero la acción por True es obligatoria.
2. ? in line: <expresión booleana> ? Ejecución por verdadero (true): Ejecución por falso (False) a>b? “a es Mayor” : “b es Mayor”;

2.1. Análisis de dos casos de anidamiento empleando If

Observemos detenidamente los dos diagramas que se presentan a continuación y su codificación en Java.

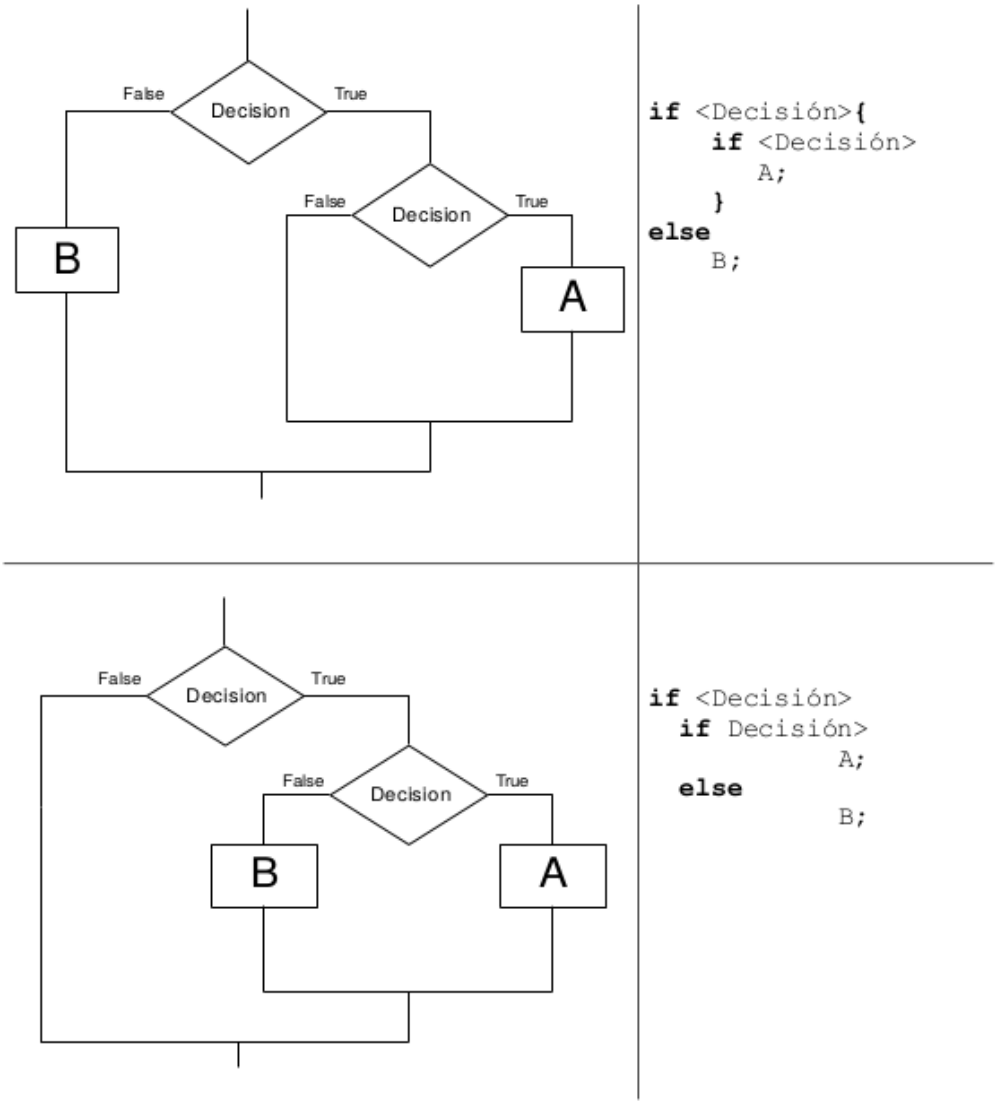


Figura 1: IF

IMPORTANTE

Observe la presencia de los delimitadores en el segmento de código Java del primer ejemplo. Si no estuvieran presentes, el código Java sería idéntico al del segundo ejemplo siendo porciones de algoritmos diferentes. El son necesarios en este caso de anidamiento porque sino el compilador consideraría que el else pertenece al último if (el anidado).

Ejemplo 4.1

Ingresar un número, si es mayor o igual que 6 mostrar el Texto 'Aprobado' en caso contrario 'Insuficiente'. Lo primero será crear un proyecto llamado U4Ejemplo01 y luego un JForm llamado VPEjem01.
Proponemos la siguiente interface:
Paso a Paso

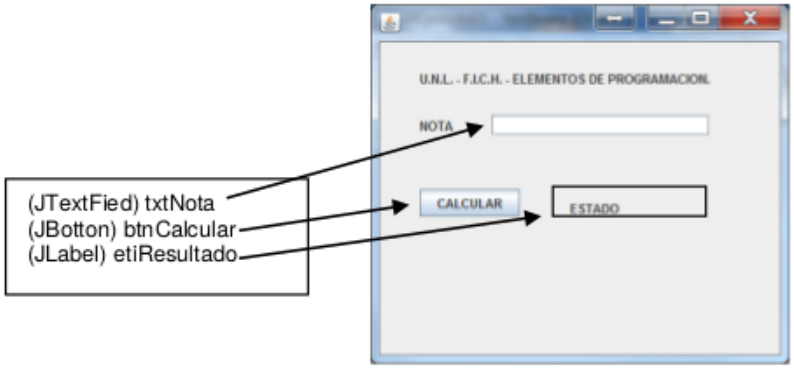


Figura 2: Interface Ejemplo 4.1

Debemos leer un número y mostrar un mensaje en pantalla.

Al hacer clic en el botón Calcular, la lectura la haremos desde una caja de texto (*JTextField*) y asignaremos la lectura a un String que es el tipo de dato que lee un componente JTextField: *String cadena= txtNota.getText()*. El problema es si, además de los dígitos numéricos, hay algún espacio en el texto que intentamos convertir a numero, esto dará error. Así que quitaremos los espacios del String con la siguiente instrucción: *cadena= cadena.replace(,);* Pero necesitaremos una variable numérica tipo entera para almacenar ese dato, que llamaremos nota (*int nota*) y la inicializaremos así *int nota= Integer.parseInt(cadena)*.

Con la instrucción *Integer.parseInt()* lo que hacemos es pasar el String a entero, (no debe tener espacios en blanco). También necesitamos una variable tipo String para mostrar el resultado que llamaremos salida y la declararemos así: *String salida*.

Ahora con una estructura condicional asignaremos el valor de salida:

```
 if (nota >= 6)
  salida = "Aprobo con " + nota;
 else
  salida = "Insuficiente con " + nota;
```

Por último al JLabel que llaremos etiResultado le asignamos nuestro String Salida:

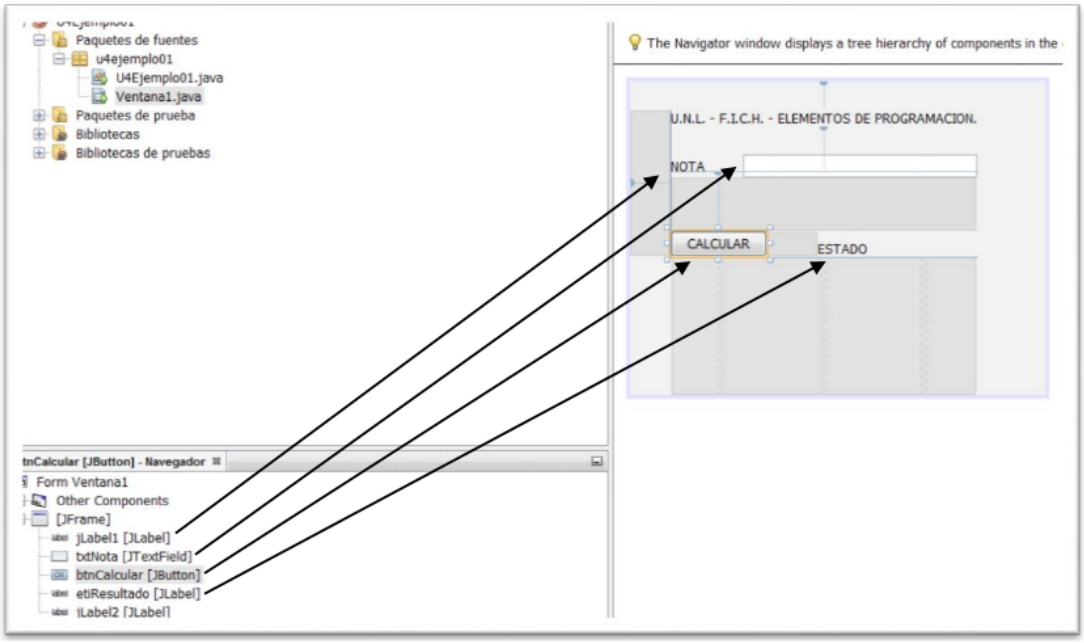
```
etiResultado.setText(salida);
```

Nos queda el siguiente código completo:

Boton Calcular (btnCalcularActionPerformed):

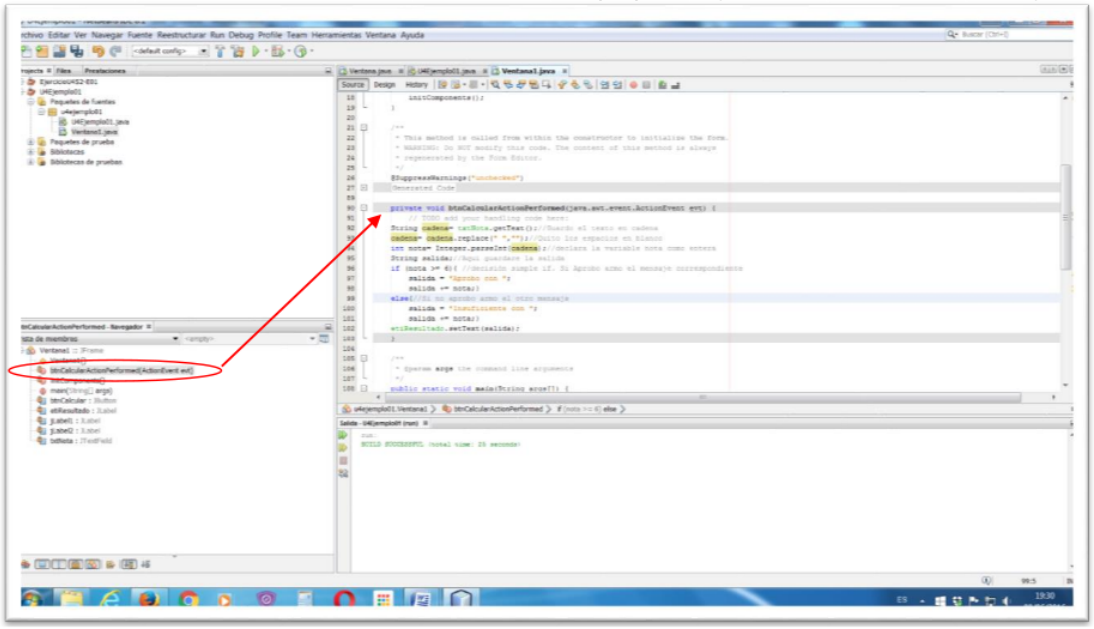
```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
 // TODO add your handling code here:
 String cadena= txtNota.getText();//Guardo el texto en cadena
 cadena= cadena.replace(" ","");//Quito los espacios en blanco
 int nota= Integer.parseInt(cadena);//declara la variable nota como entera
 String salida;//Aqui guardare la salida

 if (nota >= 6) //decisión simple if. Si Aprobo armo el mensaje correspondiente
   salida = "Aprobo con " + nota;
 else  //Si no aprobo armo el otro mensaje
   salida = "Insuficiente con " + nota;

 etiResultado.setText(salida);
}
```

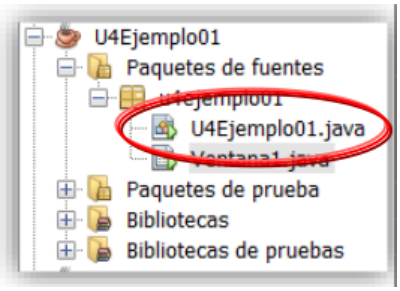
Debería tener en pantalla lo siguiente:



Haciendo doble clic en el boton btnCalcular se accede a la propiedad (btnCalcularActionPerformed)

Recuerde que para que se ejecute la aplicación con F11 (Limpiar construir) y F6 (Run) deberá modificar el main() del paquete de fuentes U4Ejemplo01.java:






```
public class U5Ejemplo01 {
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
  // TODO code application logic here
  VPEjemp01 v= new VPEjemp01();
  v.setVisible(true);
}
```

Ejemplo 4.2

Determine el tipo de raíces de una ecuación cuadrática del tipo $ax^2+bx+c=0$ conociendo los coeficientes a, b y c de la ecuación. Informe el tipo determinado.

Podemos repasar los conceptos teóricos en este sitio:

`https://es.wikipedia.org/wiki/Ecuaci%C3%B3n_de_segundo_grado`

Lo primero será crear un proyecto llamado U4Ejemplo02 y luego un JForm llamado VPEjem02.

Paso a Paso

Las raíces de una ecuación cuadrática dependen del resultado de la formula del discriminante (disc) pueden ser: reales distintas (disc >0), reales iguales (disc $=0$) o complejas (disc <0).

$disc = b*b–4*a*c$

Como para la formula de la resolvente necesitaremos la operación potencia *pow(b,2)* utilizaremos la librería *Math* o bien **b*b** y como tendremos que ingresar datos y escribir resultados por pantalla utilizaremos elementos tipo *JTextField* y *JLabel* respectivamente, también un botón tipo *JButton*.

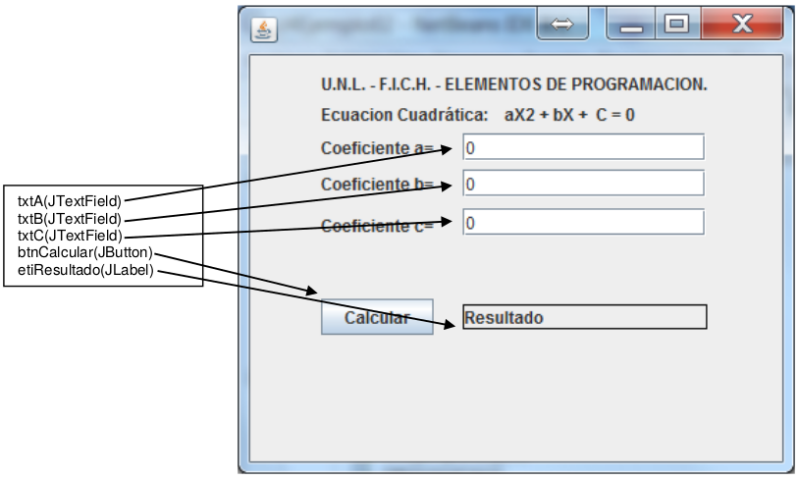
Ya en el desarrollo del programa dentro del *ActionPerformed* en el *JButton*, tendremos que definir tres variables para almacenar los coeficientes a, b y c que pueden contener números con coma.

Para ello utilizaremos el tipo *double*. También nos hará falta una variable para guardar el mensaje a mostrar en pantalla en un *JLabel*, la llamaremos texto y será de tipo *String*.

Recuerde que para que se ejecute la aplicación con F11 (Limpiar construir) y F6 (Run) deberá modificar el main() del paquete de fuentes U4Ejemplo02.java:

```
public class U5Ejemplo02 {
 /**
 * @param args the command line arguments
 */
 public static void main(String[] args) {
   // TODO code application logic here
   VPEjemp02 v= new VPEjemp02();
   v.setVisible(true);
 }
```

Proponemos la siguiente interface:



Con doble clic en el botón Calcular accedemos a la propiedad btnCalcularActionPerformed() y copiamos el siguiente código:

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
 // TODO add your handling code here:
 String texto;//Para la salida del resultado.
 double a,b,c,discriminante;//Para los coeficientes y el Discriminante
 a= Double.parseDouble(txtA.getText());//Guardo el texto Transformado en Duble
 b= Double.parseDouble(txtB.getText());//Guardo el texto Transformado en Duble
 c= Double.parseDouble(txtC.getText());//Guardo el texto Transformado en Duble
 discriminante = Math.pow(b,2) - 4 * a * c;//Calculamos el discriminante
 if (discriminante >= 0)
   if (discriminante >0)
     texto = "Raíces Reales Distintas";
   else
     texto= "Raíces Reales Iguales";
 else
```

```
15        texto = "Raíces Complejas";
16    etiResultado.setText(texto);
```

Observe en este último ejemplo, que una estructura If se anida dentro de otra, pues hay 3 posibilidades (3 tipos de raíces).

Nota

Vea en el código Java, como la indentación (espacios delante de cada línea), si bien es ignorada por el compilador, juega un rol fundamental en nuestra lectura e interpretación del código fuente. La indentación es también una muy buena forma de documentar nuestro código.

3. Estructura de selección múltiple switch

La estructura de selección switch permite seleccionar una opción de ejecución entre múltiples posibilidades. Estas posibilidades se basan en el valor de una variable de control llamada selector de tipo ordinal. La sintaxis Java es la siguiente:

```
switch <Variable de control>
{
 case v1: sentencias;
          break;
 case v2: sentencias;
          break;
 default:
          sentencias;
};
```

El compilador evalúa la variable de control y busca este valor en la lista propuesta, ejecutando la sentencia correspondiente; luego del break salta al final de la estructura. Si el valor de la variable no figura en la lista propuesta se ejecuta la sentencia propuesta después del default. La alternativa default es opcional. Notemos que utilizamos la sentencia break al finalizar cada case. Esto es muy importante, ya que sino la utilizamos, el programa, luego de entrar al case correspondiente seguirá ejecutando secuencialmente todas las sentencias posteriores. Hasta llegar a algún break o al fin de la estructura switch.

Nota

La alternativa default es opcional. La sentencia correspondiente será ejecutada si el valor de la variable de control no coincide con ninguno de los presentes en la lista del case.

Ejemplo 4.3

En base a la calificación obtenida por un alumno (1..10) determine la calificación en letras que le corresponde de acuerdo a la escala establecida por la Universidad Nacional del Litoral de Santa Fe.

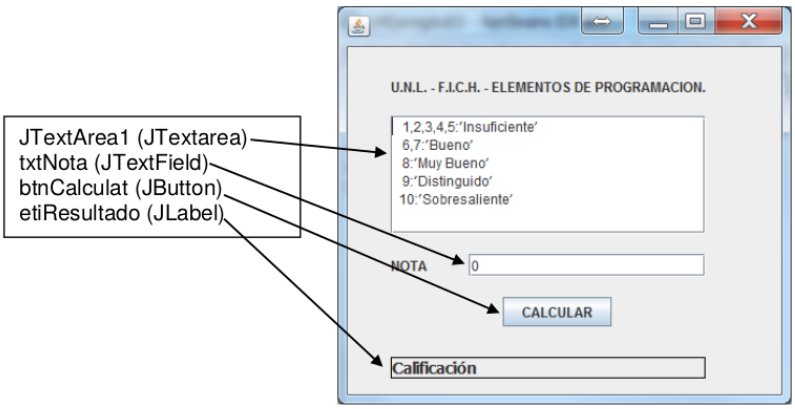
Lo primero será crear un proyecto llamado U4Ejemplo03 y luego un JForm llamado VPEjem03.

Proponemos la siguiente interface:

Paso a Paso

Recuerde que para que se ejecute la aplicación con F11 (Limpiar construir) y F6 (Run) deberá modificar el main() del paquete de fuentes U4Ejemplo03.java:

```
public class U5Ejemplo03 {
 /**
```




```
 * @param args the command line arguments
 */
 public static void main(String[] args) {
 // TODO code application logic here
 VPEjemp03 v= new VPEjemp03();
 v.setVisible(true); }
```

En la propiedad ActionPerformed del JButton debemos declarar una variable tipo entera ("int nota") para ingresar la nota que puede variar de 0 a 10. Luego otra variable tipo String ("String calificación") para asignarle la nota en letras.

```
String calificacion= txtNota.getText();
int nota= Integer.parseInt(calificacion);
```

Luego con la sentencia switch y en función del valor de nota asignamos a la variable calificación la nota en letras.

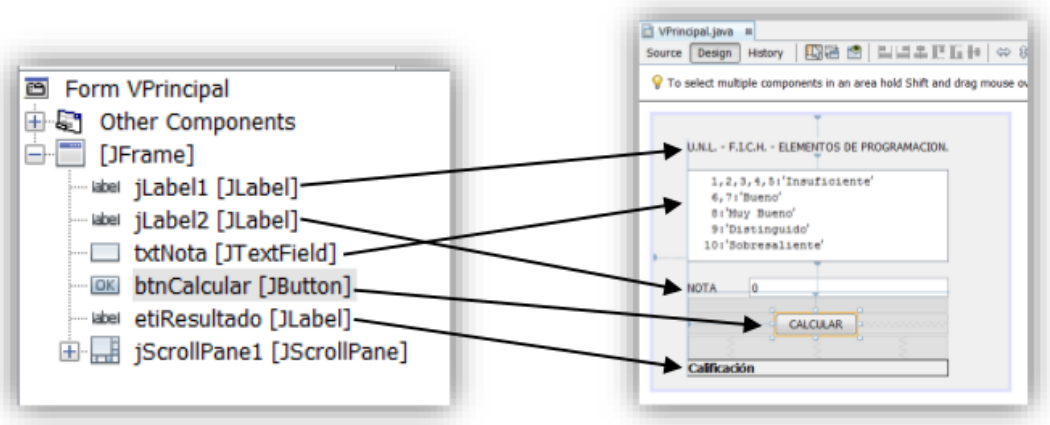
- 1,2,3,4,5: Calificación 'Insuficiente';
- 6,7: Calificación 'Bueno';
- 8: Calificación 'Muy Bueno';
- 9: Calificación 'Distinguido';
- 10: Calificación 'Sobresaliente';

Por último mostramos la nota en con:

```
etiResultado.setText(calificacion);
```

Para codificar hacemos doble clic en el botón calcular y copiamos el siguiente código:

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {

 // TODO add your handling code here:
 String calificacion= txtNota.getText();//Aqui guardare la salida
 int nota=Integer.parseInt(calificacion);//declara la variable nota como entera

 //con la sentencia switch y en función del valor de nota asignamos a la
 //variable calificación la nota en letras.

```




```
10   switch (nota){
11     case 1:
12     case 2:
13     case 3:
14     case 4:
15     case 5:
16       calificacion = "Insuficiente";
17       break;
18     case 6:
19     case 7:
20       calificacion= "Bueno";
21       break;
22     case 8:
23       calificacion= "Muy Bueno";
24       break;
25     case 9:
26       calificacion = "Distinguido";
27       break;
28     case 10:
29       calificacion= "Sobresaliente";
30       break;
31     default:
32       calificacion = "Error "+nota+" no corresponde a una nota";
33    }
34   etiResultado.setText(calificacion);
35  }
```

4. Estructura iterativa while(...)

Esta estructura permite hacer iteraciones (repeticiones) y su sintaxis en Java es muy sencilla:

```
1  while <Expresión Boolean>
2  {
3   sentencia;
4  }
```

El compilador evalúa la expresión boolean (lógica) y mientras arroje un valor true (verdadero) se ejecuta la sentencia; luego se vuelve a evaluar la expresión boolean y así sucesivamente.

IMPORTANTE

Casi siempre <sentencia> será una sentencia compuesta, pues necesitamos de al menos una acción simple para alterar el valor de verdad de la expresión Boolean. De otra forma, si la esta expresión siempre fuera verdadera el while nunca terminaría.

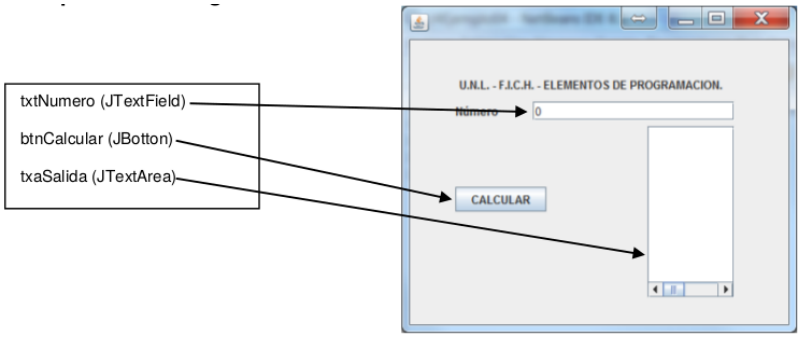
Ejemplo 4.4

Ingrese un número entero y determine los pares menores a dicho número. Muéstrelos ordenados en forma decreciente.

Si el resto de la división (%) por 2 da cero (0) el numero es par.

Lo primero será crear un proyecto llamado U4Ejemplo04 y luego un JForm llamado VPEjem04.

Proponemos la siguiente interface:



Paso a Paso

Recuerde que para que se ejecute la aplicación con F11 (Limpiar construir) y F6 (Run) deberá modificar el main() del paquete de fuentes U4Ejemplo04.java:

```
public class U5Ejemplo04 {
 /**
 * @param args the command line arguments
 */
 public static void main(String[] args) {
 // TODO code application logic here
 VPEjemp04 v= new VPEjemp04();
 v.setVisible(true);
 }
```

Necesitamos ingresar un número y deberemos guardar ese número en una variable tipo entera que definiremos con int numero;. Luego la leeremos desde el JTextField y la asignaremos a la variable numero con la instrucción:

```
//Captura el String del JTextField
String cadena= txtNumero.getText();
//Convierte cadena a entero y asigna
int numero= Integer.parseInt(cadena);
```

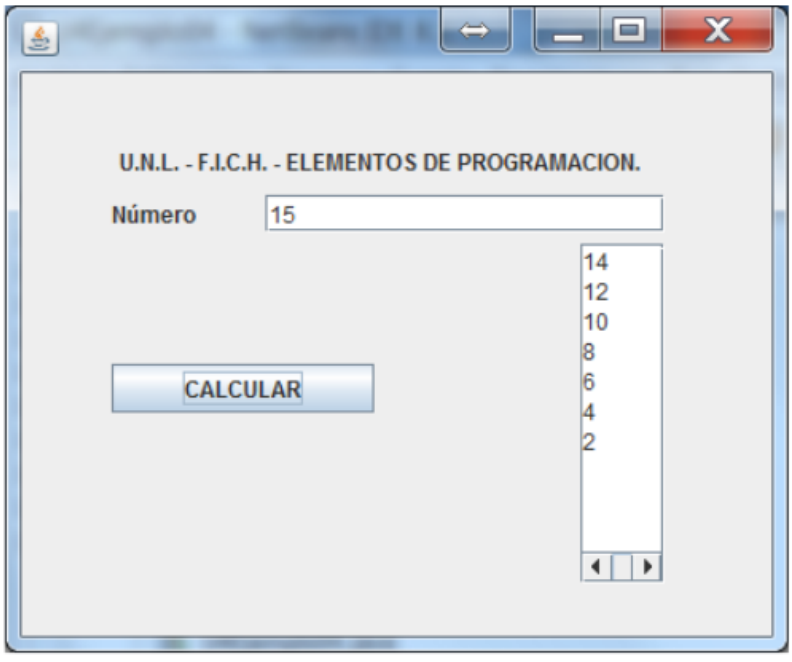
Para mostrar el resultado en pantalla crearemos un String que guardará los números solicitados. String salida=; ahora bien supongamos que nos dan el número impar 7, los pares menores son 6, 4, 2 y si nos dan el número par 8, los pares menores son los mismos. Así que si se ingresa un número impar y le sumamos 1 y luego lo vamos decrementando de 2 en 2 hasta cero.

Obtendremos todos los números pares menores a él, en cambio si el número ingresado es par, solo lo decrementaremos de 2 en 2.-

Esto lo lograremos con el ciclo repetitivo while(numero¿2), la variable de control es numero y debemos decrementarla en el interior del while para que en algún momento la comparación (numero ¿2) de cómo resultado Falso y finalice el while. Dentro del while deberemos ir concatenando los valores de numero para mostrarlos por pantalla una vez finalizado el ciclo while.

Copiamos este código en el ActionPerformed de JButton

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
 // TODO add your handling code here:
 String cadena= txtNumero.getText();//Captura el String del JTextField
 int numero= Integer.parseInt(cadena);//declara la variable numero y asigna
 String salida="";//Para guardar el resultado, se crea vacia con =""
 //Verifico que sea par
 if (numero % 2 != 0){//si no es par
   numero++;//incremento en 1 su valor (numero = numero + 1)
 }
 //Ciclo repetitivo while, se ejecuta mientras numero sea mayor a 2
 while (numero > 2){
   numero -= 2;//decremento en 2 su valor (numero = numero - 2)
   salida+= numero+"\n";//concateno el resultado y agrego un salto de linea
 }
 txaSalida.setText(salida);
}
```



5. Estructura iterativa do – while(. . .)

Esta estructura de control es similar a la sentencia While, pero la evaluación de la condición que determina si se continúa en la iteración se efectúa al final: si dicha condición arroja True continúa la iteración. Su sintaxis en Java es la siguiente:

```
do
  sentencia1;
  sentencia2;
  ..
  ..
  ..
  sentenciaN
while <Expresión Boolean>;
```

Nota

Este ciclo también itera mientras se verifique <Expresion Boolean>, pero a diferencia del ciclo anterior, en este caso la entrada al ciclo no esta condicionada por lo tanto las acciones encerradas entre el do y el while se ejecutaran al menos una vez.

6. Estructura iterativa for(. . .)

Esta estructura ejecuta la sentencia por primera vez con el valor inicial de la variable de control, la cual se incrementa automáticamente en cada iteración. La última ejecución de la sentencia se realiza cuando la variable toma el valor final propuesto. La sintaxis Java es :

```
for (valores de entrada ; condición de terminación ; iteración por ciclo) {
 Acciones;
}
```

- Las tres partes del ciclo se encuentran separadas por ; (punto y coma)
- La primer parte del ciclo especifica valores previo a su inicio.
- La segunda parte indica la condición de terminación para el ciclo, la cual esta directamente relacionada con los valores iniciales.
- Finalmente, la última parte especifica como serán manipulados los valores iniciales en cada iteración del ciclo.

Cada parte del ciclo debe incluir al menos un elemento, a la cual es posible agregar otros elementos a partir de una , (coma).

Nota

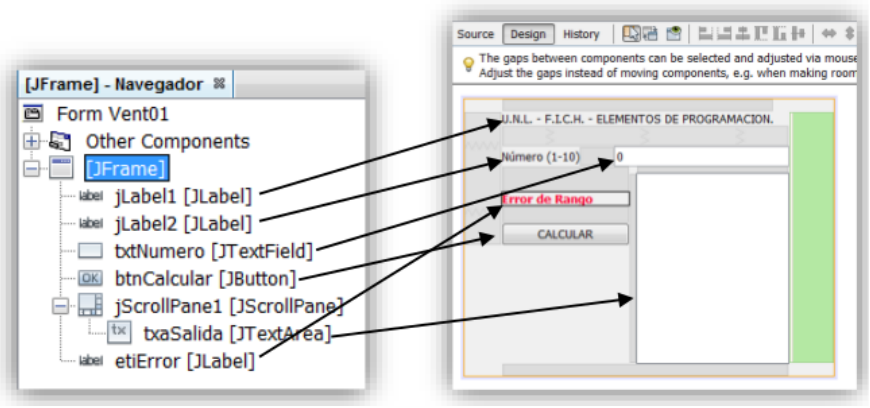
Al igual que en casos anteriores, esta estructura no posee un delimitador propio por lo que tendremos que utilizar sentencias compuestas en muchas ocasiones. El dato de la variable de control de la estructura debe ser de tipo simple ordinal. En todos los casos, el próximo valor de la variable de control se basa en la secuencia del tipo ordinal asociado a dicha variable. Se puede utilizar cualquier incremento para la variable de control, especificado en el 3er. parámetro.

Ejemplo 4.5

Ingrese un número entre 1 y 10. Valide que el número este en ese rango, si no es así reingréselo, a continuación muestre la tabla de multiplicar correspondiente.-

Lo primero será crear un proyecto llamado U4Ejemplo05 y luego un JForm llamado VPEjem05.

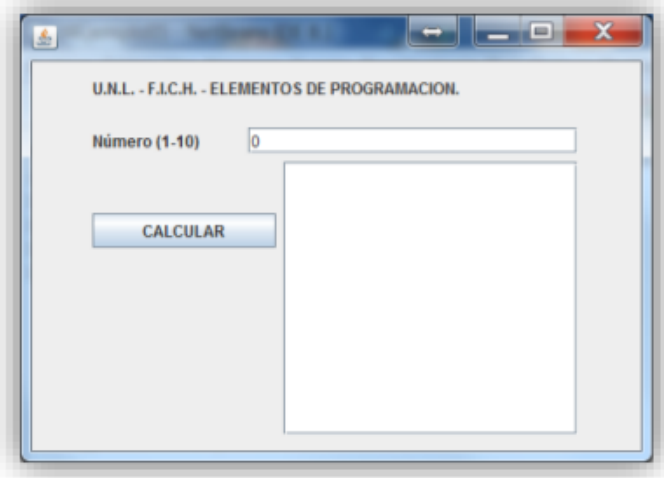
Se sugiere una interface como la siguiente



Como no queremos que el JLabel "etiError" no se vea cuando ejecutemos la aplicación debemos colocar el siguiente código en el "Source", suponiendo que ha creado el JForm con el nombre Vent01.

```
/**
 * Creates new form Vent01
 */
 public Vent01() {
 initComponents();
 etiError.setVisible(false);//Oculta la etiqueta de Error
 }
```

Si la ejecutamos veremos la siguiente ventana de interfaz



Paso a Paso

Deberemos guardar ese número en una variable tipo entera y debemos utilizar una variable para contar de 1 a 10, que definiremos con *int* numero, contador; necesitamos ingresar un número por pantalla, así que lo extraeremos del *JTextField* con la instrucción:

```
numero= Integer.parseInt(txtNumero.getText());
```

Con *if()* validamos que el numero este en el rango de 1 a 10. Aquí pondremos el JLabel "etiError", visible o no, según corresponda.

Para mostrar el resultado en pantalla crearemos un String inicializándolo como vacio (= "") que guardará

los números solicitados. *String salida=*; Por ultimo con un ciclo *for()* en donde vamos concatenando la cadena de salida con los valores de la tabla de multiplicar solicitada.

Copiamos este código en el ActionPerformed del JButton

```
private void btnCalcularActionPerformed(java.awt.event.ActionEvent evt) {
 // TODO add your handling code here:
 String salida="";//Para guardar el resultado, se crea vacia con =""
 //Declaro numero y contador de tipo entero
 int numero, contador;
 numero= Integer.parseInt(txtNumero.getText());//asigna contenido JTextFile
 if ((numero<1)||(numero>10))//valida el rango de 1 a 10.-
   etiError.setVisible(true);//Pone Visible la etiqueta de Error
 else{
   etiError.setVisible(false);//Oculta la etiqueta de Error
   //Ciclo repetitivo for
   for (contador=1; contador<=10; contador++){
     salida+= numero+" * "+contador+" = "+contador*numero+"\n";
   }
 }
 txaSalida.setText(salida);
}
```

Recuerde que para que se ejecute la aplicación con F11 (Limpiar construir) y F6 (Run) deberá modificar el main() del paquete de fuentes U4Ejemplo05.java:

```
public class U5Ejemplo05 {
 /**
 * @param args the command line arguments
 */
 public static void main(String[] args) {
 // TODO code application logic here
 VPEjemp05 v= new VPEjemp05();
 v.setVisible(true); }
```

7. Resumen

1. if

```
 if <expresión boolean>
 { sentencia1;}
  else {sentencia2;}
```

2. switch

```
 switch <Variable de control> {
  case v1: sentencia;
           break;
  case v2: sentencia;
           break;
  default:
           sentencia
 };
```

3. while

```
while <Expresión Boolean> {
 sentencia;
}
```

4. do - while

```
do {
  sentencia1;
  sentencia2;
  ..
  ..
  sentenciaN;
}
while <Expresión Boolean>;
```

5. for

```
for (valores de entrada ; condición de terminación ; iteración por ciclo) {
  Acciones;
}
```