Repetir

En Karel, además de las instrucciones básicas, también existe una instrucción que le dice a Karel que ejecute varias veces una o más instrucciones, y se escribe como a continuación:

PASCAL	JAVA
repetir <numero> veces <instrucción>	iterate(<numero>) <instrucción>

Por ejemplo, para indicarle a Karel que debe caminar cinco posiciones hacia adelante, debemos escribir:

PASCAL	JAVA

repetir 5 veces
    avanza;
	iterate(5)
	    move();

Para especificar si queremos que más de una instrucción se ejecute varias veces, debemos delimitarlas de la siguiente manera:

PASCAL	JAVA

repetir <numero> veces inicio
    <código>
fin;
	iterate(<numero>) {
	    <código>
	}

El *código* que escribamos entre los delimitadores (palabras en rojo) se ejecutara en orden el número de veces que especifiquemos.

Por ejemplo para hacer que Karel construya un cuadrado de zumbadores de 5x5 el código sería:

PASCAL

```
1  iniciar-programa
2      inicia-ejecucion
3          repetir 4 veces inicio
4              repetir 4 veces inicio
5                  avanza;
6                  deja-zumbador;
7              fin;
8              repetir 3 veces
9                  gira-izquierda;
10         fin;
11         apagate;
12     termina-ejecucion
13 finalizar-programa
```

Escrito por Enrique Lira Vargas

JAVA

```
1  class program {
2      program() {
3          iterate(4) {
4              iterate(4) {
5                  move();
6                  putbeeper();
7              }
8              iterate(3)
9                  turnleft();
10         }
11         turnoff();
12     }
13 }
```

Como podemos ver, las líneas 3 y 10 delimitan el fragmento de código que se repetirá 4 veces (una por cada lado del cuadrado). Las líneas 4 y 7 delimitan un fragmento de código que se repetirá cuatro veces, el cual es una secuencia de avanzar y poner un zumbador. Las líneas 8 y 9 le indican a Karel que debe girar tres veces a la izquierda (girar a la derecha).

Al ejecutarse, tiene una sentencia de repetición en la línea 3 que nos indica en este caso particular que haremos cuatro lados. Cada lado se conforma de una sucesión de 4 beepers contiguos, y giramos a Karel para que pueda dibujar el siguiente lado.

Ejercicio. *Ejecuta el código anterior y verifica que en realidad hace un cuadrado de 5x5, no olvides poner zumbadores en la mochila de Karel.*

Pregunta de reflexión. *¿Por qué si cada lado solo se compone de una sucesión de 4 zumbadores contiguos, Karel dibuja un cuadrado de 5x5?*

Los sensores de Kare

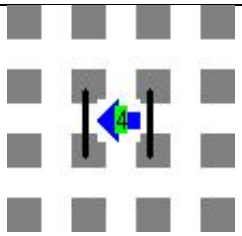
Karel como buen robot, tiene sensores, los cuales le permiten saber cosas respecto a las condiciones en las que se encuentra. Cada uno de estos sensores, puede ser preguntado mediante una **Función Booleana** específica para cada sensor la cual es denominada booleana por la cualidad de que solo puede ser verdadera o falsa. A continuación veremos cada una de ellas:

PASCAL	JAVA
frente-libre	frontIsClear
frente-bloqueado	frontIsBlocked
izquierda-libre	leftIsClear
izquierda-bloqueada	leftIsBlocked
derecha-libre	rightIsClear
derecha-bloqueada	rightIsBlocked
junto-a-zumbador	nextToABeeper
no-junto-a-zumbador	notNextToABeeper
algun-zumbador-en-la-mochila	anyBeepersInBeeperBag
ningun-zumbador-en-la-mochila	noBeepersInBeeperBag
orientado-al-norte	facingNorth
orientado-al-sur	facingSouth
orientado-al-este	facingEast
orientado-al-oeste	facingWest
no-orientado-al-norte	notFacingNorth
no-orientado-al-sur	notFacingSouth
no-orientado-al-este	notFacingEast
no-orientado-al-oeste	notFacingWest

Cada una de estas funciones booleanas pregunta por una cosa en específico y su nombre lo indica, sin embargo la función ***junto-a-zumbador* | *nextToABeeper*** puede ser la más confusa, puesto que solo es verdadera cuando Karel está sobre un zumbador.

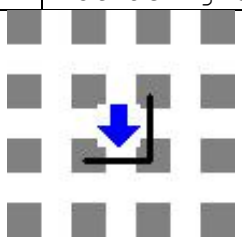
Ahora veamos unos ejemplos de los valores que tomarían en diversas situaciones:

Escrito por Enrique Lira Vargas



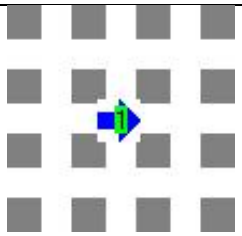
Zumbadores en la mochila: **6**

PASCAL	**JAVA**	**Valor**
frente-libre	frontIsClear	FALSO
frente-bloqueado	frontIsBlocked	VERDADERO
izquierda-libre	leftIsClear	VERDADERO
izquierda-bloqueada	leftIsBlocked	FALSO
derecha-libre	rightIsClear	VERDADERO
derecha-bloqueada	rightIsBlocked	FALSO
junto-a-zumbador	nextToABeeper	VERDADERO
no-junto-a-zumbador	notNextToABeeper	FALSO
algun-zumbador-en-la-mochila	anyBeepersInBeeperBag	VERDADERO
ningun-zumbador-en-la-mochila	noBeepersInBeeperBag	FALSO
orientado-al-norte	facingNorth	FALSO
orientado-al-sur	facingSouth	FALSO
orientado-al-este	facingEast	FALSO
orientado-al-oeste	facingWest	VERDADERO
no-orientado-al-norte	notFacingNorth	VERDADERO
no-orientado-al-sur	notFacingSouth	VERDADERO
no-orientado-al-este	notFacingEast	VERDADERO
no-orientado-al-oeste	notFacingWest	FALSO



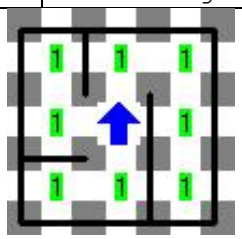
Zumbadores en la mochila: **0**

PASCAL	**JAVA**	**Valor**
frente-libre	frontIsClear	FALSO
frente-bloqueado	frontIsBlocked	VERDADERO
izquierda-libre	leftIsClear	FALSO
izquierda-bloqueada	leftIsBlocked	VERDADERO
derecha-libre	rightIsClear	VERDADERO
derecha-bloqueada	rightIsBlocked	FALSO
junto-a-zumbador	nextToABeeper	FALSO
no-junto-a-zumbador	notNextToABeeper	VERDADERO
algun-zumbador-en-la-mochila	anyBeepersInBeeperBag	FALSO
ningun-zumbador-en-la-mochila	noBeepersInBeeperBag	VERDADERO
orientado-al-norte	facingNorth	FALSO
orientado-al-sur	facingSouth	VERDADERO
orientado-al-este	facingEast	FALSO
orientado-al-oeste	facingWest	FALSO
no-orientado-al-norte	notFacingNorth	VERDADERO
no-orientado-al-sur	notFacingSouth	FALSO
no-orientado-al-este	notFacingEast	VERDADERO
no-orientado-al-oeste	notFacingWest	VERDADERO



Zumbadores en la mochila: **6**

PASCAL	**JAVA**	**Valor**
frente-libre	frontIsClear	VERDADERO
frente-bloqueado	frontIsBlocked	FALSO
izquierda-libre	leftIsClear	VERDADERO
izquierda-bloqueada	leftIsBlocked	FALSO
derecha-libre	rightIsClear	VERDADERO
derecha-bloqueada	rightIsBlocked	FALSO
junto-a-zumbador	nextToABeeper	VERDADERO
no-junto-a-zumbador	notNextToABeeper	FALSO
algun-zumbador-en-la-mochila	anyBeepersInBeeperBag	VERDADERO
ningun-zumbador-en-la-mochila	noBeepersInBeeperBag	FALSO
orientado-al-norte	facingNorth	FALSO
orientado-al-sur	facingSouth	FALSO
orientado-al-este	facingEast	VERDADERO
orientado-al-oeste	facingWest	FALSO
no-orientado-al-norte	notFacingNorth	VERDADERO
no-orientado-al-sur	notFacingSouth	VERDADERO
no-orientado-al-este	notFacingEast	FALSO
no-orientado-al-oeste	notFacingWest	VERDADERO



Zumbadores en la mochila: **0**

PASCAL	**JAVA**	**Valor**
frente-libre	frontIsClear	VERDADERO
frente-bloqueado	frontIsBlocked	FALSO
izquierda-libre	leftIsClear	VERDADERO
izquierda-bloqueada	leftIsBlocked	FALSO
derecha-libre	rightIsClear	FALSO
derecha-bloqueada	rightIsBlocked	VERDADERO
junto-a-zumbador	nextToABeeper	FALSO
no-junto-a-zumbador	notNextToABeeper	VERDADERO
algun-zumbador-en-la-mochila	anyBeepersInBeeperBag	FALSO
ningun-zumbador-en-la-mochila	noBeepersInBeeperBag	VERDADERO
orientado-al-norte	facingNorth	VERDADERO
orientado-al-sur	facingSouth	FALSO
orientado-al-este	facingEast	FALSO
orientado-al-oeste	facingWest	FALSO
no-orientado-al-norte	notFacingNorth	FALSO
no-orientado-al-sur	notFacingSouth	VERDADERO
no-orientado-al-este	notFacingEast	VERDADERO
no-orientado-al-oeste	notFacingWest	VERDADERO

Unión de Funciones Booleanas

Podemos crear nuevas funciones a partir de otras utilizando los siguientes operadores:

NOMBRE	**PASCAL**	**JAVA**
Y lógico (AND)	y	&&
O lógico (OR)	o	||
NO lógico (NOT)	no	!

Los operadores pueden ser utilizados de la siguiente manera:

NOMBRE	**PASCAL**	**JAVA**
Y lógico (AND)	<función 1> **y** <función 2>	<función 1> **&&** <función 2>
O lógico (OR)	<función 1> **o** <función 2>	<función 1> **||** <función 2>
NO lógico (NOT)	**no** <función>	**!** <función>

Las tablas de verdad para los operadores lógicos de Karel son las siguientes.

AND

		PASCAL	**JAVA**
función 1	función 2	<función 1> **y** <función 2>	<función 1> **&&** <función 2>
VERDADERO	VERDADERO	VERDADERO	
VERDADERO	FALSO	FALSO	
FALSO	VERDADERO	FALSO	
FALSO	FALSO	FALSO	

OR

		PASCAL	**JAVA**
función 1	función 2	<función 1> **o** <función 2>	<función 1> **||** <función 2>
VERDADERO	VERDADERO	VERDADERO	
VERDADERO	FALSO	VERDADERO	
FALSO	VERDADERO	VERDADERO	
FALSO	FALSO	FALSO	

NOT

	PASCAL	**JAVA**
función	**no** <función>	**!** <función>
VERDADERO	FALSO	
FALSO	VERDADERO	

Escrito por Enrique Lira Vargas

Instrucción Si

Ahora que ya conocemos la funciones booleanas de Karel, podemos pasar a utilizarlas. La primera forma de utilizarlas y la más simple es por medio de la instrucción **SI** la cual nos permite preguntar por alguna función booleana y si su valor es VERDADERO, ejecuta un fragmento de código, en caso contrario puede ejecutar o no otro fragmento de código, cabe aclarar que el valor de la función booleana depende únicamente del estado en el que se encuentre Karel al momento de preguntar. La forma de escribir esta instrucción es la siguiente:

PASCAL	**JAVA**

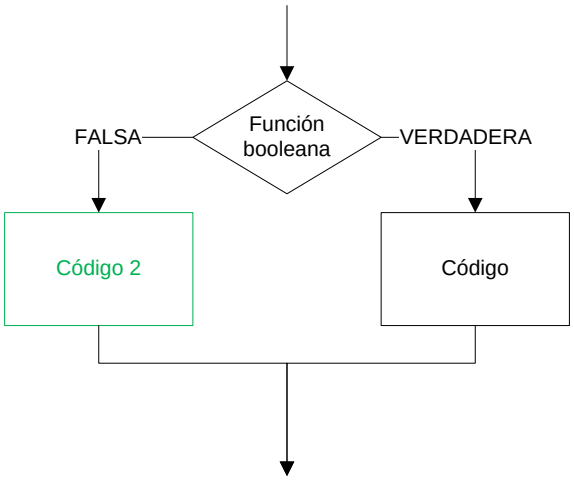
`si <función booleana> entonces inicio`
`  <código>`
`fin sino inicio`
`  <código 2>`
`fin;`
	`if( <función booleana> ){`
	`  <código>`
	`} else {`
	`  <código 2>`
	`}`

La parte en verde es opcional.

La forma en que funciona esta instrucción es la siguiente, primero verifica el valor de la función booleana si el valor es VERDADERO entonces ejecuta el <código> en caso contrario si existe la sección en verde, ejecuta el <código 2>. Si no existe la sección verde, continua con la ejecución del programa con la siguiente instrucción después de la instrucción Si.

Note que si la función booleana es VERDADERA, el <código 2> nunca se ejecutará. De igual modo si la función booleana es FALSA, el <código> nunca se ejecutará.

Visto como un diagrama de flujo, tenemos:



Escrito por Enrique Lira Vargas

Ejemplo:

Hagamos un programa que pregunte si Karel está sobre un zumbador, si es así, que lo recoja. Y si no es así, que coloque un zumbador en esa posición. Después, que Karel avance una casilla hacia adelante.

El código nos quedaría de la siguiente manera:

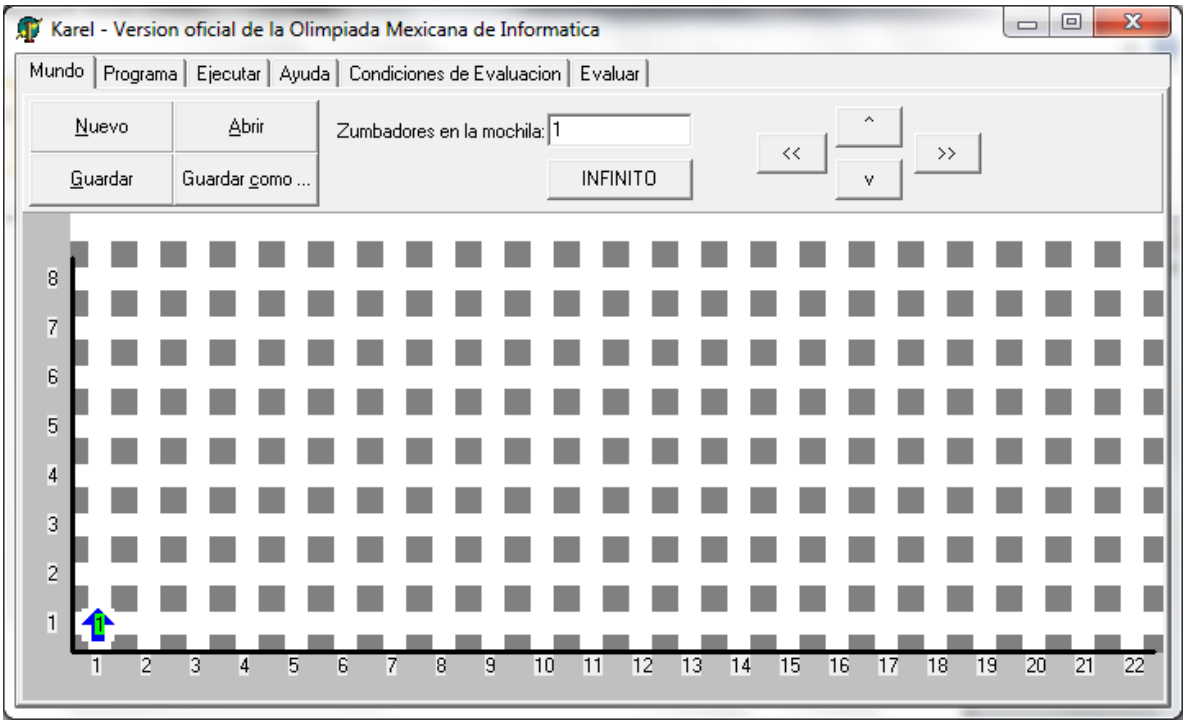
PASCAL

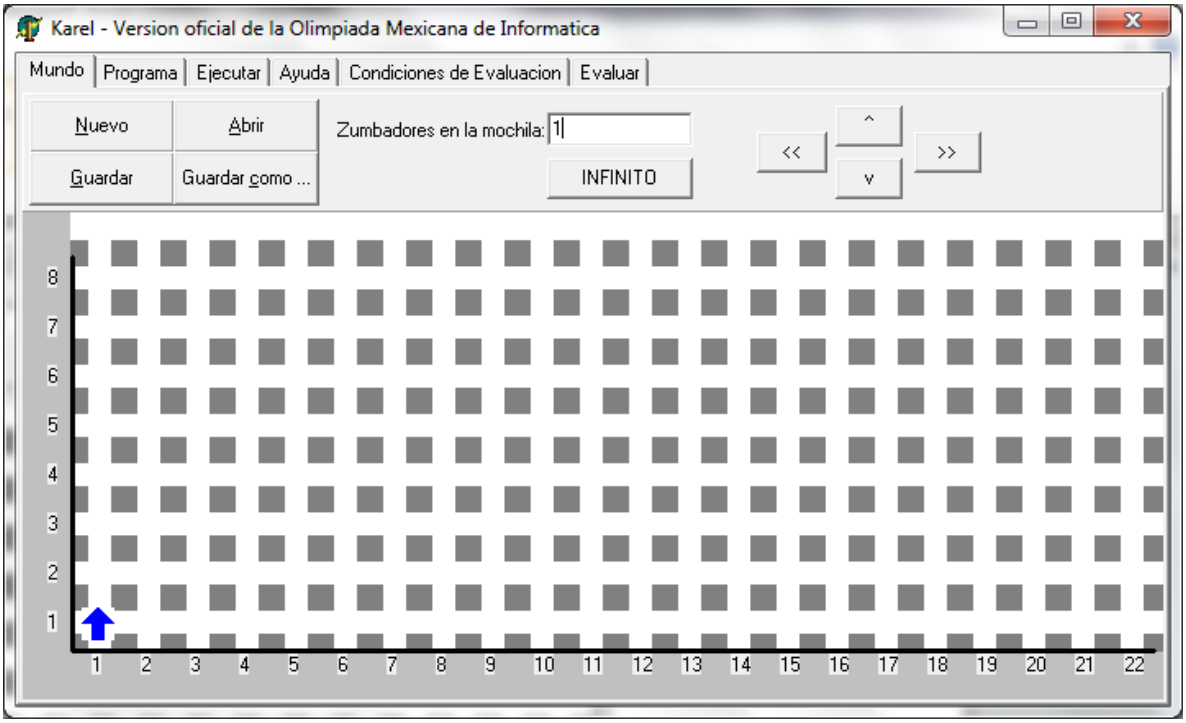
```
1   iniciar-programa
2       inicia-ejecucion
3           si junto-a-zumbador entonces inicio
4               coge-zumbador;
5           fin sino inicio
6               deja-zumbador;
7           fin;
8           avanza;
9           apagate;
10      termina-ejecucion
11  finalizar-programa
12
13
```

JAVA

```
1   class program {
2       program() {
3           if( nextToABeeper ){
4               pickbeeper();
5           } else {
6               putbeeper();
7           }
8           move();
9           turnoff();
10      }
11  }
12
13
```

Probemos el programa con dos mundos distintos:





Ejecuta el programa paso a paso con ambos casos y observa cómo se comporta la instrucción Si.

Ejemplo 2:

Hagamos ahora algo más complicado. Hagamos que Karel avance una casilla hacia adelante, después preguntemos si tiene una pared al frente. Si es así, veamos si su izquierda o su derecha están libres, si es así, rodeamos la pared y dejamos un beeper, en caso contrario apagamos a Karel. Si no tenía una pared al frente, avanzamos y dejamos un beeper.

El código quedaría de la siguiente manera:

PASCAL

```
1  iniciar-programa
2      inicia-ejecucion
3          avanza;
4          si frente-bloqueado entonces inicio
5              si izquierda-libre o derecha-libre entonces inicio
6                  si izquierda-libre entonces inicio
7                      gira-izquierda;
8                      avanza;
9                      repetir 3 veces
10                         gira-izquierda;
11                     avanza;
12                     repetir 3 veces
13                         gira-izquierda;
14                     avanza;
15                 fin sino inicio
16                     repetir 3 veces
17                         gira-izquierda;
18                     avanza;
19                     gira-izquierda;
20                     avanza;
21                     gira-izquierda;
22                     avanza;
23                 fin;
24                 deja-zumbador;
25             fin;
26         fin sino inicio
27             avanza;
28             deja-zumbador;
29         fin;
30         apagate;
31     termina-ejecucion
32 finalizar-programa
```

JAVA

```
1  class program {
2      program() {
3          move();
4          if( frontIsBlocked ){
5              if( leftIsClear || rightIsClear ){
6                  if( leftIsClear ) {
7                      turnleft();
8                      move();
9                      iterate(3)
10                         turnleft();
11                     move();
12                     iterate(3)
13                         turnleft();
14                     move();
15                 } else {
16                     iterate(3)
17                         turnleft();
18                     move();
19                     turnleft();
20                     move();
21                     turnleft();
22                     move();
23                 }
24                 putbeeper();
25             }
26         } else {
27             move();
28             putbeeper();
29         }
30         turnoff();
31     }
32 }
```

La instrucción Si marcada en rojo nos ayuda a saber si el Karel puede caminar libremente o si hay una pared.

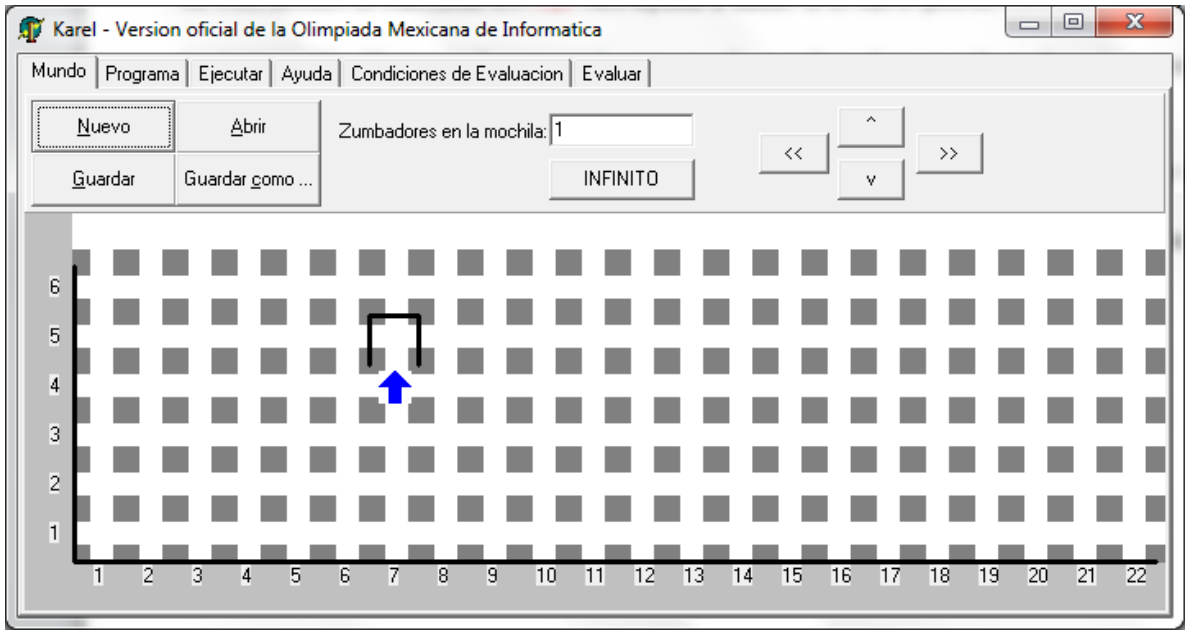
La instrucción Si marcada en verde nos ayuda a saber si se puede rodear la pared ya sea por la izquierda o por la derecha.

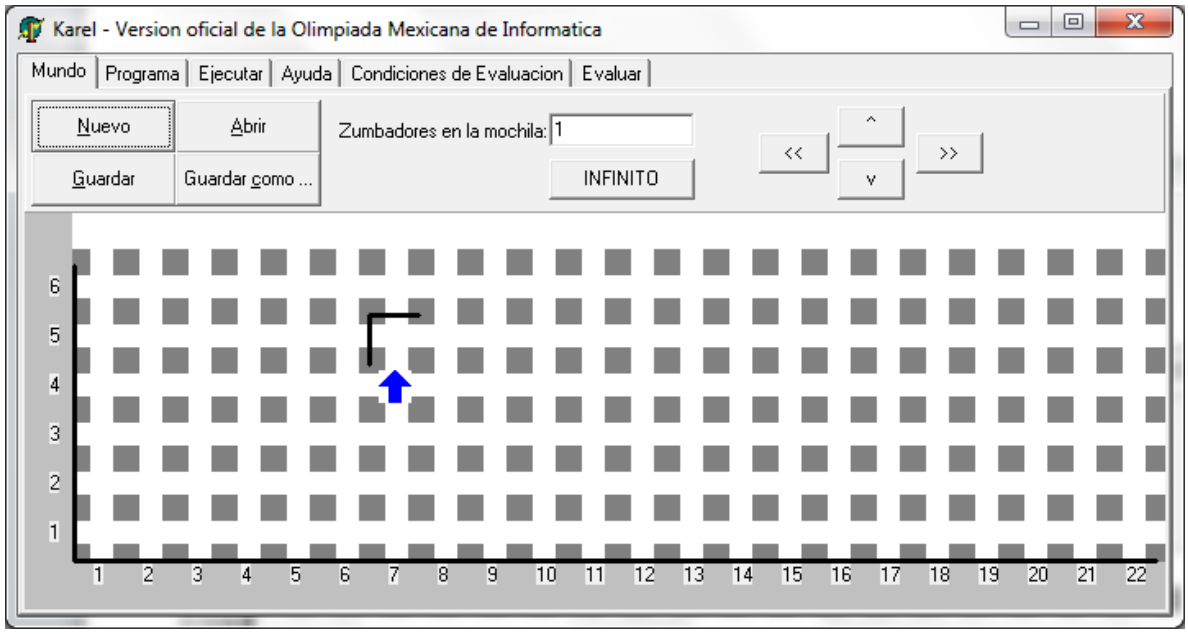
La instrucción Si marcada en azul nos ayuda a distinguir de qué lado Karel debe rodear la pared.

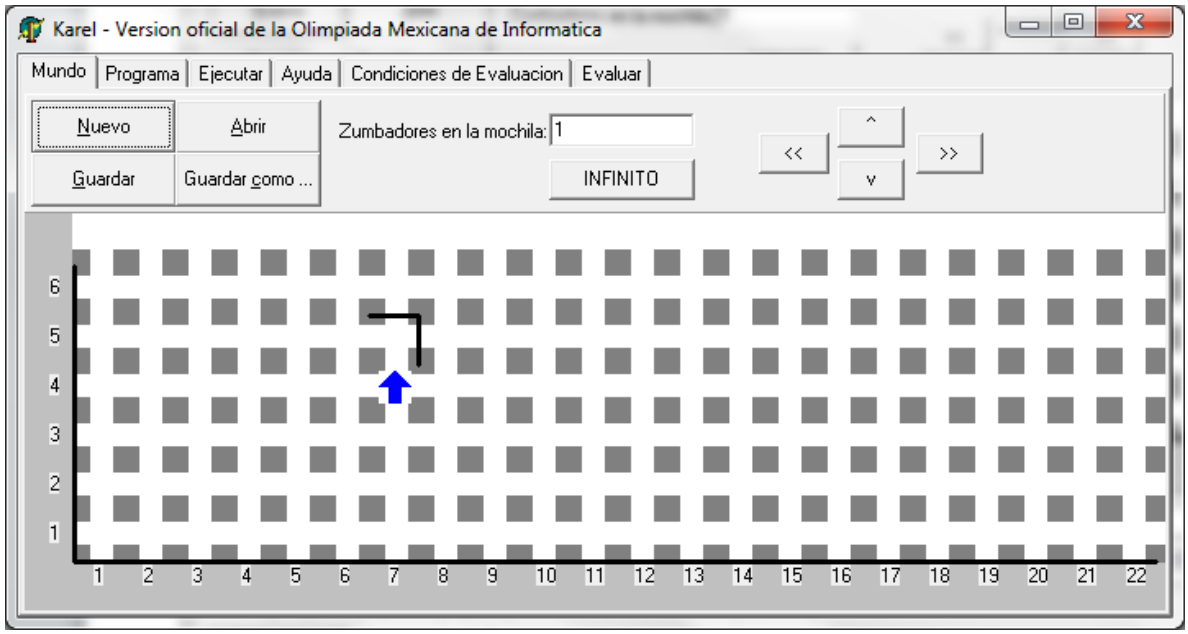
Antes de continuar, analiza el código unos minutos.

Escrito por Enrique Lira Vargas

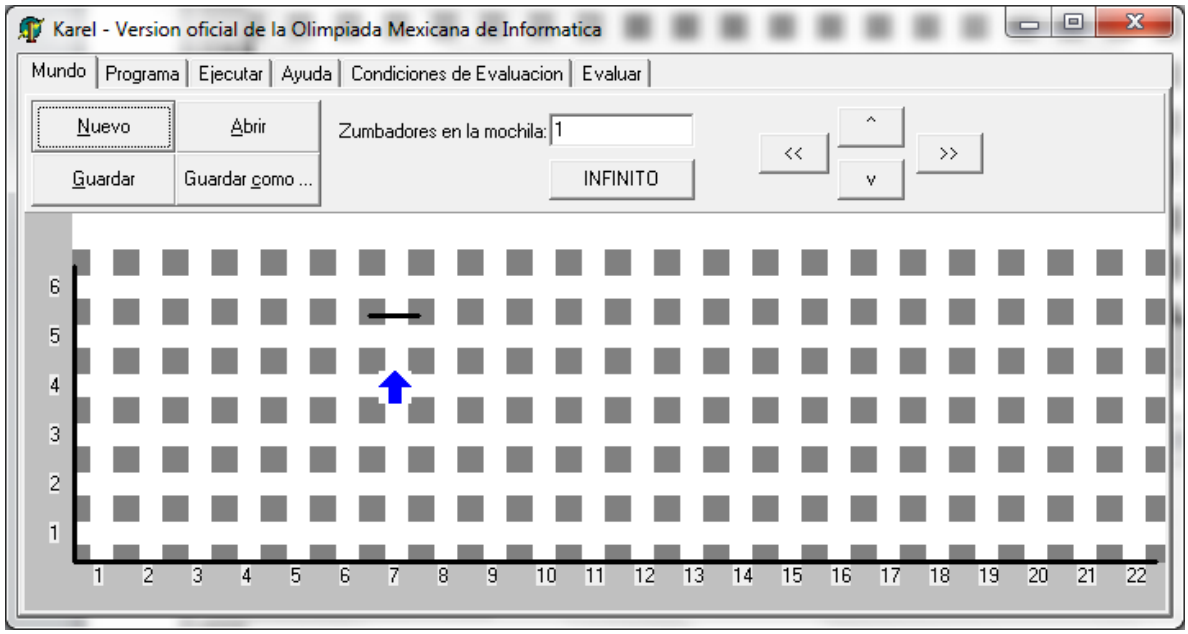
Ahora probemos el código con los siguientes casos:







Escrito por Enrique Lira Vargas



Ejecuta el programa para todos los casos paso por paso hasta que entiendas el funcionamiento del programa.

Escrito por Enrique Lira Vargas

Instrucción Mientras

Esta instrucción funciona de manera similar a la instrucción Repetir, sin embargo a diferencia de ella, la instrucción Mientras hace que el fragmento de código se ejecute una y otra vez mientras cierta función booleana sea VERDADERA. La manera de escribir esta instrucción es la siguiente:

PASCAL

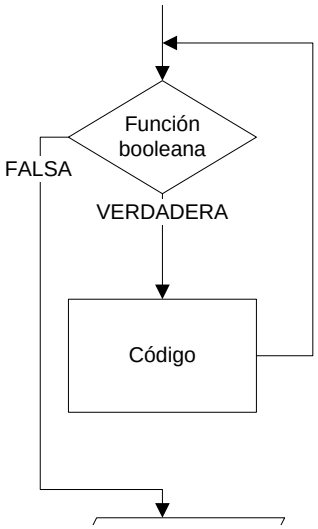
```
mientras <función booleana> hacer inicio
    <código>
fin;
```

JAVA

```
while ( <función booleana> ) {
    <código>
}
```

La forma en la que opera esta instrucción es la siguiente, primero evalúa la función booleana, si es FALSA, continua con la siguiente instrucción, pero si es VERDADERA, ejecuta el código y una vez que termina de ejecutarlo, vuelve a preguntar por la función booleana, si es FALSA, continua con la siguiente instrucción después de la instrucción Mientras, pero si es VERDADERA, ejecuta el código y una vez que termina de ejecutarlo, vuelve a preguntar por la función booleana, esto se repite hasta que la función booleana sea falsa (mientras la función booleana sea verdadera).

Visto como un diagrama de flujo:



Escrito por Enrique Lira Vargas

Ejemplo:

Hagamos que Karel camine hasta que encuentre una pared. La forma de indicarle a Karel que haga esto es: “camina mientras tengas el frente libre”. Ahora veámoslo en los lenguajes de Karel.

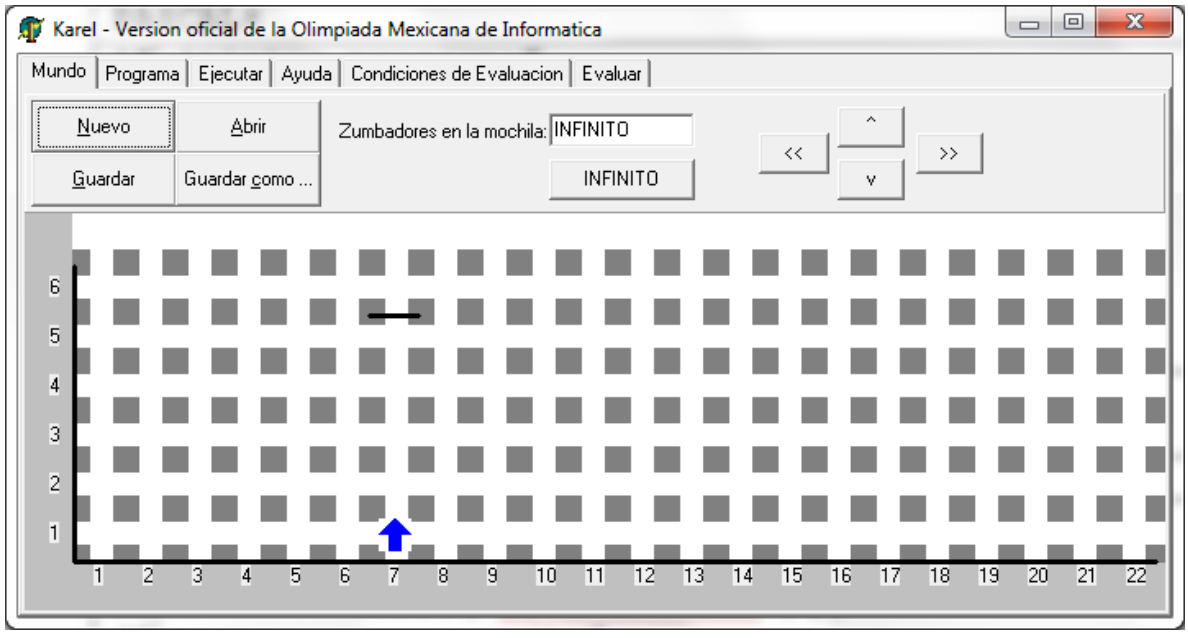
PASCAL

```
1  iniciar-programa
2      inicia-ejecucion
3          mientras frente-libre hacer inicio
4              avanza;
5          fin;
6          apagate;
7      termina-ejecucion
8  finalizar-programa
```

JAVA

```
1  class program {
2      program() {
3          while( frontIsClear ){
4              move();
5          }
6          turnoff();
7      }
8  }
```

Ahora probemos el programa con el siguiente caso:



Ejecuta el programa paso a paso y observa cómo funciona. Para finalizar mueve la pared de lugar y observa como sin importar donde la pongas, Karel siempre se detendrá al llegar a ella.

Como ejercicio, crea cuatro programas que orienten a Karel al Norte, Sur, Este y Oeste.

Escrito por Enrique Lira Vargas

Ejemplo 2:

Hagamos que Karel camine dejando un camino de zumbadores hasta que encuentre una pared frente a si o hasta que se le terminen los zumbadores.

La forma correcta de decírselo seria: “camina mientras tengas zumbadores y el frente esté libre, y en cada paso, deja un zumbador”. Ahora expresado en los lenguajes de Karel:

PASCAL

```
1  iniciar-programa
2      inicia-ejecucion
3          mientras frente-libre y algun-zumbador-en-la-mochila hacer inicio
4              deja-zumbador;
5              avanza;
6          fin;
7          apagate;
8      termina-ejecucion
9  finalizar-programa
```

JAVA

```
1  class program {
2      program() {
3          while( frontIsClear && anyBeepersInBeeperBag ){
4              putbeeper();
5              move();
6          }
7          turnoff();
8      }
9  }
```

Nuevas instrucciones y Parametros

Recordemos que Karel tiene de manera primitiva 5 instrucciones, sin embargo, nosotros podemos definirle nuevas instrucciones compuestas de otras. Esto ayuda en gran medida a poder dividir el programa en pedazos pequeños más fáciles de entender, y así abstraer más fácilmente el funcionamiento del programa. La forma de definir una nueva instrucción es la siguiente:

PASCAL

```
define-nueva-instruccion <nombre> como inicio
    <código>
fin;
```

JAVA

```
void <nombre>(){
    <código>
}
```

El nombre que le demos a la nueva instrucción será mediante el cual la mandaremos a llamar, al igual que se mandan a llamar las instrucciones primitivas de Karel. Este nombre debe cumplir ciertos requisitos, debe ser alfanumérico, no debe contener espacios y no puede empezar con números.

Ejemplos

Nombres Correctos	Nombres Incorrectos
Norte7	7Norte
giraDerecha	gira Derecha
numero	número

Una instrucción no puede ser definida dentro de otra y todas deben ser definidas en la siguiente región verde:

PASCAL

```
iniciar-programa
    inicia-ejecucion
        <código principal>
    termina-ejecucion
finalizar-programa
```

JAVA

```
class program {
    program(){
        <código principal>
    }
}
```

Escrito por Enrique Lira Vargas

Hay que recordar que Karel siempre empieza su ejecución en el <código principal> y si las instrucciones que definimos en la región verde nunca son llamadas, nunca se ejecutarán.

Ejemplo:

Modifiquemos el primer ejemplo de este documento para hacerlo más entendible.

El código original es:

PASCAL

```
1  iniciar-programa
2      inicia-ejecucion
3          repetir 4 veces inicio
4              repetir 4 veces inicio
5                  avanza;
6                  deja-zumbador;
7              fin;
8              repetir 3 veces
9                  gira-izquierda;
10         fin;
11         apagate;
12     termina-ejecucion
13 finalizar-programa
```

JAVA

```
1  class program {
2      program() {
3          iterate(4) {
4              iterate(4) {
5                  move();
6                  putbeeper();
7              }
8              iterate(3)
9                  turnleft();
10         }
11         turnoff();
12     }
13 }
```

Para hacerlo más entendible, definamos tres nuevas instrucciones:

- **giraDerecha** hará que Karel gire a la derecha.
- **construyeLado** hará que Karel construya uno de los lados del cuadrado.
- **construyeCuadrado** construirá el cuadrado.

El código final quedaría:

PASCAL

```
iniciar-programa
    define-nueva-instruccion giraDerecha como inicio
        repetir 3 veces
            gira-izquierda;
    fin;

    define-nueva-instruccion construyeLado como inicio
        repetir 4 veces inicio
            avanza;
            deja-zumbador;
        fin;
    fin;

    define-nueva-instruccion construyeCuadrado como inicio
        repetir 4 veces inicio
            construyeLado;
            giraDerecha;
        fin;
    fin;

    inicia-ejecucion
        construyeCuadrado;
        apagate;
    termina-ejecucion
finalizar-programa
```

JAVA

```
class program {
    void giraDerecha() {
        iterate(3)
            turnleft();
    }

    void construyeLado() {
        iterate(4) {
            move();
            putbeeper();
        }
    }

    void construyeCuadrado() {
        iterate(4) {
            construyeLado();
            giraDerecha();
        }
    }

    program() {
        construyeCuadrado();
        turnoff();
    }
}
```

Escrito por Enrique Lira Vargas

Escrito por Enrique Lira Vargas