Guia 3 [Curso Backend 1]

Estructuras de control

Material de

lectura

## OBJETIVOS DE LA GUÍA

En esta guía aprenderemos a:

* Definir y utilizar estructuras de control
* Definir y utilizar estructuras repetitivas
* Entender las sentencias de salto

## 

# 



# Estructuras de control

Ya conocemos las estructuras de control de nuestro paso por PseInt pero ahora vamos a verlas en detalle en Java y como es su funcionamiento.

# Instrucciones de bifurcación

Mediante estas instrucciones el desarrollo lineal de un programa se interrumpe. Las bifurcaciones o al flujo de un programa puede ser según el punto del programa en el que se ejecuta la instrucción hacia adelante o hacia atrás. De esto se encargan las estructuras de control.

Para esto también vamos a usar los operadores lógicos o condicionales, estos son los mismos que en PseInt pero se escriben de distintas formas:

| **Operadores Condicionales** | |
| --- | --- |
| **&& [Y]** | **AND** [se representa con dos ampersands (&&) y devuelve "true" si ambas expresiones son verdaderas. Si al menos una de las expresiones es falsa, el operador devuelve "false"]. |
| **|| [O]** | **OR** [se representa con dos barras verticales (||) y devuelve "true" si al menos una de las expresiones es verdadera. Solo devuelve "false" si ambas expresiones son falsas]. |
|  | En resumen, el operador "and" [Y] solo devuelve "true" si ambas expresiones son verdaderas, mientras que el operador "or" [O] devuelve "true" si al menos una de las expresiones es verdadera. |
| **!** | Operador Lógico de Negación. |

## Estructuras de Control

Las estructuras de control son construcciones hechas a partir de palabras reservadas del lenguaje que permiten modificar el flujo de ejecución de un programa. De este modo, pueden crearse construcciones de alternativas mediante sentencias condicionales y bucles de repetición de bloques de instrucciones. Hay que señalar que un bloque de instrucciones se encontrará encerrado mediante llaves {……..} si existe más de una instrucción.

## Estructuras condicionales

Los condicionales son estructuras de control que cambian el flujo de ejecución de un programa de acuerdo con si se cumple o no una condición. Cuando el flujo de control del programa llega al condicional, el programa evalúa la condición y determina el camino a seguir. Existen dos tipos de estructuras condicionales, las estructuras ***if / else*** y la estructura ***switch*.**

**If/Else**

La estructura *if* es la más básica de las estructuras de control de flujo. Esta estructura le indica al programa que ejecute cierta parte del código sólo si la condición evaluada es verdadera («true»). La forma más simple de esta estructura es la siguiente:

if(**<condición>**){

<sentencias>

}

En donde, <*condición*> es una expresión condicional cuyo resultado luego de la evaluación es un dato booleano(lógico) verdadero o falso. El bloque de instrucciones <sentencias> se ejecuta si, y sólo si, la expresión (que debe ser lógica) se evalúa a true, es decir, se cumple la condición.

Luego, en caso de que la condición no se cumpla y se quiera ejecutar otro bloque de código, otra forma de expresar esta estructura es la siguiente:

if(**<condición>**){

<sentencias A>

} else {

<sentencias B>

}

El flujo de control del programa funciona de la misma manera, cuando se ejecuta la estructura if, se evalúa la expresión condicional, si el resultado de la condición es verdadero se ejecutan las sentencias que se encuentran contenidas dentro del bloque de código if (<sentencias A>). Contrariamente, se ejecutan las sentencias contenidas dentro del bloque else (<sentencias B>).

En muchas ocasiones, se anidan estructuras alternativas **if-else**, de forma que se pregunte por una condición si anteriormente no se ha cumplido otra y así sucesivamente.

if (**<condicion1>**) {

<sentencias A>

} else if(**<condicion2>**){

<sentencias B>

} else {

<sentencias C>

}

EJEMPLO:

| public static void main(String[] args) {  int num1 = 5;  int num2 = 7;  if (num1 < num2) {  System.out.println("La variable num1 aloja un número menor a la variable num2");  }else {  System.out.println("La variable num1 aloja un número mayor a la variable num2");  } |
| --- |

## Ejercicio 6

Implementar un programa que le pida dos números enteros al usuario y determine si ambos o alguno de ellos es mayor a 25.

package esmayora25;

import java.util.Scanner;

public class EsMayorA25 {

    public static void main(String[] args) {

        Scanner leer = new Scanner(System.in);

        int num1, num2;

        System.out.println("Ingrese el primer número entero:");

        num1 = leer.nextInt();

        System.out.println("Ingrese el segundo número entero:");

        num2 = leer.nextInt();

        if (num1 > 25 && num2 > 25) {

            System.out.println("Ambos números son mayores a 25.");

        } else if (num1 > 25 || num2 > 25) {

            System.out.println("Al menos uno de los números es mayor a 25.");

        } else {

            System.out.println("Ninguno de los números es mayor a 25.");

        }

    }

}

### \*Este programa solicita al usuario que ingrese dos números enteros y luego los compare con 25.

### \*Si ambos números son mayores a 25, se imprime el mensaje "Ambos números son mayores a 25".

### \*Si alguno de los dos números es mayor a 25, se imprime el mensaje "Al menos uno de los números es mayor a 25".

### \*Si ninguno de los dos números es mayor a 25, se imprime el mensaje "Ninguno de los números es mayor a 25".

## Switch [INTERRUPTOR]

El bloque *switch* evalúa qué valor tiene la variable, y de acuerdo con el valor que posee ejecuta las sentencias del bloque case correspondiente, es decir, del bloque case que cumpla con el valor de la variable que se está evaluando dentro del switch.

switch(**<variable>**) {

case **<valor1>**:

<sentencias1>

break;

case **<valor2>**:

<sentencias2>

break;

default:

<sentencias3>

}

El uso de la sentencia break que va detrás de cada case termina la sentencia switch que la envuelve, es decir que el control de flujo del programa continúa con la primera sentencia que se encuentra a continuación del cierre del bloque switch. Si el programa comprueba que se cumple el primer valor (valor1) se ejecutará el bloque de instrucciones <sentencias1>, pero si no se encuentra inmediatamente la sentencia break también se ejecutaría las instrucciones <sentencias2>, y así sucesivamente hasta encontrarse con la palabra reservada break o llegar al final de la estructura.

Las instrucciones dentro del bloque default se ejecutan cuando la variable que se está evaluando no coincide con ninguno de los valores case. Esta sentencia equivale a else de la estructura if-else.

EJEMPLO:

| public static void main(String[] args) {   Scanner leer = new Scanner(System.in);   int opcion;   System.out.println("Ingrese una opción");   opcion = leer.nextInt();   switch (opcion) {  case 1:  System.out.println("Esta línea de código se ejecuta si opcion = 1");  break;   case 2:  System.out.println("Esta línea de código se ejecuta si opcion = 2");  break;   default:  System.out.println("El valor ingresado en la variable opcion es diferente" + "a todos los casos analizados por el switch");  } } |
| --- |

## Ejercicio 7

Considera que estás desarrollando una web para una empresa que fabrica motores (suponemos que se trata del tipo de motor de una bomba para mover fluidos). Definir una variable tipoMotor y permitir que el usuario ingrese un valor entre 1 y 4. El programa debe mostrar lo siguiente:

o Si el tipo de motor es 1, mostrar un mensaje indicando “La bomba es una bomba de agua”.

o Si el tipo de motor es 2, mostrar un mensaje indicando “La bomba es una bomba de gasolina”.

o Si el tipo de motor es 3, mostrar un mensaje indicando “La bomba es una bomba de hormigón”.

o Si el tipo de motor es 4, mostrar un mensaje indicando “La bomba es una bomba de pasta alimenticia”.

o Si no se cumple ninguno de los valores anteriores mostrar el mensaje “No existe un valor válido para tipo de bomba”

package fabricademotores;

// @author Horacio

 import java.util.Scanner;

public class FabricaDeMotores {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int tipoMotor;

        System.out.println("Ingrese el tipo de motor (1-4): ");

        tipoMotor = scanner.nextInt();

        switch (tipoMotor) {

            case 1:

                System.out.println("La bomba es una bomba de agua.");

                break; // El break corta la secuencia repetitiva del bucle.

            case 2:

                System.out.println("La bomba es una bomba de gasolina.");

                break;

            case 3:

                System.out.println("La bomba es una bomba de hormigón.");

                break;

            case 4:

                System.out.println("La bomba es una bomba de pasta alimenticia.");

                break;

            default:

                System.out.println("No existe un valor válido para tipo de bomba.");

        }

    }

}

\*Este programa utiliza la estructura de control switchpara comparar el valor ingresado por el usuario con los posibles

\*valores para el tipo de motor y mostrar el mensaje correspondiente. Si el valor ingresado no coincide con ninguno de los posibles valores,

\*se muestra un mensaje de error.

Ver más en el vídeo del enlace: https://youtu.be/ZC1vQ6U-U3g

**Estructuras repetitivas**

Durante el proceso de creación de programas, es muy común encontrarse con que una operación o conjunto de operaciones deben repetirse muchas veces. Para ello es importante conocer las estructuras de algoritmos que permiten repetir una o varias acciones, un número determinado de veces.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan *bucles*, y se denomina *iteración* al hecho de repetir la ejecución de una secuencia de acciones.

Todo bucle tiene que llevar asociada una condición, que es la que va a determinar cuándo se repite el bucle y cuando deja de repetirse.

## Sentencias de Salto

En Java existen dos formas de realizar un salto incondicional en el flujo “normal” de un programa. A saber, las instrucciones, break y continúe.

### Break [interrumpe o aborta el bucle en ejecución]

La instrucción break sirve para abandonar una estructura de control, tanto de las condicionales (if-else y switch) como de las repetitivas (for, do-while y while). En el momento que se ejecuta la instrucción break, el control del programa sale de la estructura en la que se encuentra contenida y continua con el programa.

### Continue [Nos permite dentro del bucle avanzar a la siguiente iteración]

La sentencia *continue* corta la iteración en donde se encuentra el continue, pero en lugar de salir del bucle, continúa con la siguiente iteración. La instrucción continue transfiere el control del programa desde la instrucción continue directamente a la cabecera del bucle (for, do-while o while) donde se encuentra.

Ejemplo:

public class SentenciaSalto {

    private int[] numeros;

    private int objetivo;

    public SentenciaSalto(int[] numeros, int objetivo) {

        this.numeros = numeros;

        this.objetivo = objetivo;

    }

    public int obtenerIndiceDeObjetivo() {

        int salto = (int) Math.floor(Math.sqrt(numeros.length));

        int anterior = 0;

        while (numeros[Math.min(salto, numeros.length) - 1] < objetivo) {

            anterior = salto;

            salto += (int) Math.floor(Math.sqrt(numeros.length));

            if (anterior >= numeros.length) {

                return -1;

            }

        }

        while (numeros[anterior] < objetivo) {

            anterior++;

            if (anterior == Math.min(salto, numeros.length)) {

                return -1;

            }

        }

        if (numeros[anterior] == objetivo) {

            return anterior;

        }

        return -1;

    }

}

La clase **SentenciaSalto** recibe dos argumentos en su constructor: un arreglo de enteros **numeros**y un entero **objetivo**. La clase tiene un método llamado **obtenerIndiceDeObjetivo()**que devuelve el índice del elemento en el arreglo **numeros**que coincide con el valor de **objetivo.** Si no se encuentra el valor, el método devuelve -1.

Esta clase utiliza el algoritmo de salto para buscar el valor de **objetivo**en el arreglo **numeros.** El algoritmo de salto es una optimización de la búsqueda secuencial que utiliza saltos en lugar de recorrer todo el arreglo elemento por elemento.

## While [bucle Mientras]

La estructura *while* ejecuta un bloque de instrucciones mientras se cumple una condición. La condición se comprueba antes de empezar a ejecutar por primera vez el bucle, por lo tanto, si la condición se evalúa a «false» en la primera iteración, entonces el bloque de instrucciones no se ejecutará ninguna vez.

while (**<condición>**) {

<sentencias>

}

Ejemplo:

| public static void main(String[] args) {    Scanner leer = new Scanner(System.in);    String respuesta = "S";    while (respuesta.equalsIgnoreCase("S")) {    System.out.println("Desea continuar?");  respuesta = leer.nextLine();  }  } |
| --- |

Ver más Bucles y sentencias de salto en el enlace: https://youtu.be/vG7DsUP22mY

## Ejercicio 8

Escriba un programa que valide si una nota está entre 0 y 10, sino está entre 0 y 10 la nota se pedirá de nuevo hasta que la nota sea correcta.

import java.util.Scanner;

public class ValidarNota {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        float nota = -1;

        while(nota < 0 || nota > 10) {

            System.out.print("Introduce una nota entre 0 y 10: ");

            nota = sc.nextFloat();

            if(nota < 0 || nota > 10) {

                System.out.println("La nota introducida no es correcta.");

            }

        }

        System.out.println("La nota introducida es: " + nota);

    }

}

## Este programa en Java pide al usuario que introduzca una nota entre 0 y 10. Si la nota reemplazada es menor que 0 o mayor que 10, el programa muestra un mensaje de error y vuelve a pedir al usuario que introduzca una nota. Si la nota necesaria es, el programa muestra un mensaje válido indicando la nota necesaria.

## Do / While [bucle Hacer Mientras]

En este tipo de bucle, el bloque de instrucciones se ejecuta siempre al menos una vez. El bloque de instrucciones se ejecutará mientras la condición se evalúe a «true». Por lo tanto, entre las instrucciones que se repiten deberá existir alguna que, en algún momento, haga que la condición se evalúe a «false», de lo contrario el bucle será infinito.

do {

<sentencias>

} while (**<condición>**);

Ejemplo:

| import java.util.Scanner;  public class ValidarNota {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          float nota = -1;          while(nota < 0 || nota > 10) {              System.out.print("Introduce una nota entre 0 y 10: ");              nota = sc.nextFloat();              if(nota < 0 || nota > 10) {                  System.out.println("La nota introducida no es correcta.");              }          }          System.out.println("La nota introducida es: " + nota);      }  } |
| --- |

## Ejercicio 9

Escriba un programa que lea 20 números. Si el número leído es igual a cero se debe salir del bucle y mostrar el mensaje "Se capturó el número cero". El programa deberá calcular y mostrar el resultado de la suma de los números leídos, pero si el número es negativo no debe sumarse. **Nota: recordar el uso de la sentencia break.**

package leernumero;

// @author Horacio

import java.util.Scanner;

public class LeerNumero {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        int suma = 0; //La i++ es un operador de incremento.

        for (int i = 1; i <= 20; i++) {

            System.out.println("Ingrese un número:");

            int num = sc.nextInt();

            if (num == 0) {

                System.out.println("Se capturó el número cero");

                break;

            } else if (num > 0) {

                suma += num;

            }

        }

        System.out.println("La suma de los números ingresados es: " + suma);

    }

}

Este programa en Java utiliza un bucle forpara leer 20 números ingresados ​​por el usuario usando la clase Scanner. Si el número ingresado es cero, el bucle se detiene y se muestra el mensaje "Se capturó el número cero". Si el número es positivo, se suma a la variable suma. Si el número es negativo, simplemente se omite.

Finalmente, el programa muestra el resultado de la suma de los números ingresados ​​(sin contar los negativos).

[La diferencia entre do-while y while es que, do-while evalúa su condición al final del bloque en lugar de hacerlo al inicio. Por lo tanto, el bloque de sentencia después del “do” siempre se ejecutan al menos una vez]

Lo aprendido hasta aquí:

* Utilizar una estructura DO/WHILE
* Distinguir cómo se ejecutarán los comandos y cuándo se verifica la condición.
* Comprender cuándo debo usar una estructura DO/WHILE.
* Diferenciar una estructura WHILE de una DO/WHILE.

## For [bucle PARA]

La estructura *for* proporciona una forma compacta de recorrer un rango de valores cuando la cantidad de veces que se debe iterar un bloque de código es conocida. La forma general de la estructura for se puede expresar del siguiente modo:

for (<inicialización>; <terminación>; <incremento>) {

<sentencias>

}

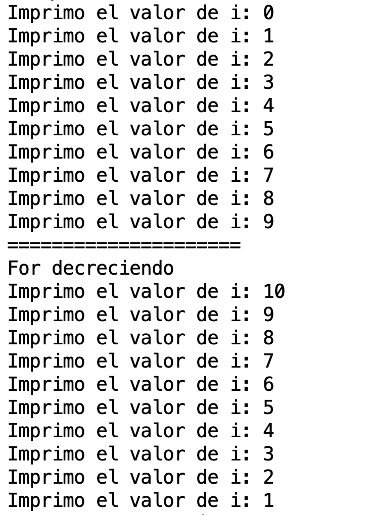
La expresión <inicialización> inicializa el bucle y se ejecuta una sola vez al iniciar el bucle. El bucle termina cuando al evaluar la expresión <terminación> el resultado que se obtiene es false. La expresión <incremento> se invoca después de cada iteración que realiza el bucle; esta expresión incrementa o decrementa un valor hasta que se cumpla la condición de <terminación> del bucle.

Como regla general puede decirse que se utilizará el bucle for cuando se conozca de antemano el número exacto de veces que ha de repetirse un determinado bloque de instrucciones. Se utilizará el bucle do-while cuando no se conoce exactamente el número de veces que se ejecutará el bucle, pero se sabe que por lo menos se ha de ejecutar una. Se utilizará el bucle while cuando es posible que no deba ejecutarse ninguna vez.

EJEMPLO:

| package leernumero;  // @author Horacio  import java.util.Scanner;  public class LeerNumero {      public static void main(String[] args) {          Scanner sc = new Scanner(System.in);          int suma = 0;          for (int i = 1; i <= 20; i++) {              System.out.println("Ingrese un número:");              int num = sc.nextInt();              if (num == 0) {                  System.out.println("Se capturó el número cero");                  break;              } else if (num > 0) {                  suma += num;              }          }          System.out.println("La suma de los números ingresados es: " + suma);      }  } |
| --- |

**Resultado:**

****

Más de Estructuras REPETITIVAS:

Definir una interfaz común para las estructuras repetitivas que permitan encapsular su funcionalidad y facilitar su uso en diferentes contextos. Por ejemplo, se puede crear una interfaz llamada "Repetible" que contenga métodos para inicializar, ejecutar y detener la estructura repetitiva.

public interface Repetible {

void inicializar();

void ejecutar();

void detener();

}

Implementar la interfaz en una clase abstracta que contenga los métodos comunes a todas las estructuras repetitivas, como una variable de tiempo de actualización y un bucle principal que se encargue de ejecutar la estructura repetitiva hasta que se detenga.

public abstract class EstructuraRepetitiva implements Repetible {

    protected long tiempoActualizacion;

    protected boolean detener;

    public void setTiempoActualizacion(long tiempoActualizacion) {

        this.tiempoActualizacion = tiempoActualizacion;

    }

    public void detener() {

        this.detener = true;

    }

    public void ejecutar() {

        inicializar();

        while (!detener) {

            actualizar();

            try {

                Thread.sleep(tiempoActualizacion);

            } catch (InterruptedException e) {

                e.printStackTrace();

            }

        }

    }

    protected abstract void actualizar();

## }

Crear clases concretas que extiendan la clase abstracta y que definan la funcionalidad específica de cada estructura repetitiva. Por ejemplo, se puede crear una clase llamada "BucleFor" que implemente un bucle para su método "actualizar".

public class BucleFor extends EstructuraRepetitiva {

    private int contador;

    public void inicializar() {

        contador = 0;

    }

    protected void actualizar() {

        for (int i = 0; i < contador; i++) {

            // Hacer algo

        }

        contador++;

    }

}

## De esta manera, se pueden crear fácilmente nuevas clases concretas que implementen diferentes tipos de estructuras repetitivas, y que puedan ser reutilizadas en diferentes contextos de manera eficiente.

## Ejercicio 10

Realizar un programa que lea 4 números (comprendidos entre 1 y 20) e imprima el número ingresado seguido de tantos asteriscos como indique su valor. Por ejemplo:

5 \*\*\*\*\*

3 \*\*\*

11 \*\*\*\*\*\*\*\*\*\*\*

2 \*\*

package leercuatronum;

// @author Horacio

 import java.util.Scanner;

public class LeerCuatroNum {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        int[] numbers = new int[4];

        // Leer los 4 números

        for (int i = 0; i < 4; i++) {

            System.out.print("Ingrese un número entre 1 y 20: ");

            int num = scanner.nextInt();

            // Verificar que el número esté entre 1 y 20

            while (num < 1 || num > 20) {

                System.out.print("El número debe estar entre 1 y 20. Ingrese otro número: ");

                num = scanner.nextInt();

            }

            numbers[i] = num;

        }

        // Imprimir los números con asteriscos

        for (int i = 0; i < 4; i++) {

            System.out.print(numbers[i] + " ");

            for (int j = 0; j < numbers[i]; j++) {

                System.out.print("\*");

            }

            System.out.println();

        }

    }

}

Este programa en Java le pide al usuario que ingrese 4 números entre 1 y 20, verifica que cada número esté dentro de ese rango, y luego imprime cada número seguido de tantos asteriscos como indique su valor. Utiliza un ciclo for para leer los números y otro ciclo for para imprimir los números con asteriscos.

Revisemos lo aprendido hasta aquí:

* Utilizar una estructura [for]
* Distinguir el inicio, la terminación y el incremento.
* Comprender cómo itera el valor [i]
* Implementar [for] anidados.