Material de Repaso para Examen

# Conceptos Teóricos Generales

## Algoritmo:

Serie de **instrucciones ordenadas** que permiten, al ejecutarse correctamente, realizar **una tarea específica**. Esto significa que cada algoritmo permite realizar una única tarea y **sólo esa tarea**.

Un algoritmo puede requerir **de datos extra** para que se lo pueda ejecutar. Estos datos son llamados “datos de entrada” o “parámetros”. También puede generar datos extra durante la ejecución, que deben entregarse de alguna manera al finalizar el algoritmo. Estos datos son llamados “datos de salida” o “datos de retorno” del algoritmo.

Un programa está formado por, al menos, un algoritmo.

## Sentencias:

Una sentencia es el nombre que se le da a cada instrucción específica que forma un algoritmo. Es una orden que describe una acción a realizar. Una sentencia ocupa, por lo general, una sola línea de código.

En lenguajes de programación como Java, las sentencias se distinguen en que terminan con un punto y coma.

## Comentarios:

Son líneas de código precedidas por una serie de símbolos (por lo general “//”) que le indican al compilador que no debe tener en cuenta dichas líneas al generar el programa.

Se usan para introducir información en lenguaje cotidiano dentro del código para explicar su funcionamiento o facilitar la lectura del mismo.

Ejemplo:

esta sentencia se ejecutará (no está comentada);

// esta sentencia NO se ejecutará (esta comentada);

## Bloques:

Un bloque es **un conjunto de sentencias declaradas entre dos llaves**. **Todas las estructuras de control generan bloques**.

Las sentencias entre esas llaves actúan solo en el contexto de esas llaves. Esto significa que, si una sentencia declara una estructura o un objeto (una **variable**, por ejemplo) dentro de un bloque, estos **dejarán de existir** (serán borrados de la memoria) cuando la ejecución salga de ese bloque. Sin embargo, las sentencias de un bloque **pueden actuar** sobre **objetos declarados fuera de su bloque,** y **los efectos** que las sentencias tengan sobre esos objetos **seguirán presentes** una vez que la ejecución salga de ese bloque.

Ejemplo:

sentencia fuera del bloque;

**estructuraQueGeneraElBloque**{

sentencia dentro del bloque;

}

sentencia fuera del bloque;

## Estructuras de datos:

Son espacios en la memoria identificados (tienen un nombre proporcionado por el programador) que permiten almacenar datos de forma temporal (son borradas de la memoria al terminar la ejecución). Estos datos se pueden modificar o utilizar más adelante en otras sentencias del algoritmo.

Estas estructuras además tienen un *tipo*, que indica cómo deben ser los datos que se pueden almacenar en ellas. Existen muchos tipos de datos, pero los más usados son:

|  |  |
| --- | --- |
| Tipo | Características de los datos almacenados |
| char | Texto, pero solo un carácter (letra, número, símbolo o espacio). |
| String | Texto. |
| int | Números enteros (sin decimales) positivos o negativos. |
| double | Números reales (pueden tener decimales) positivos o negativos. |
| boolean | Valores lógicos (Verdadero o Falso) |

Hay dos tipos de estructuras de datos básicas:

### Variables:

Las variables permiten almacenar un único dato a la vez.

//Declaración de una variable (en Java)

**TipoDeDatos nombreDeVariable = valorInicial;**

//Modificación de los datos contenidos en una variable

**nombreDeVariable = valorNuevo;**

//Acceder a los datos contenidos en una variable

**funcionDeMiPrograma**( nombreDeVariable );

### Listas (también llamadas *arrays* o *vectores*)

Las listas son parecidas a las variables, pero tienen *posiciones internas* que permiten almacenar varios datos en la misma estructura al mismo tiempo.

La cantidad de posiciones disponibles en una lista se denomina longitud. La longitud se establece al declarar la lista en el código del programa y no puede modificarse.

Las posiciones de la lista se identifican y manejan mediante un número: la primera posición tiene el número 0, la siguiente, el número 1, y así hasta la última posición.

//Declaración de una lista (en Java)

**TipoDeDatos[] nombreDeLista = new TipoDeDatos[5] ;**

//Para este ejemplo se usa 5 como longitud para la nueva lista, pero puede ser cualquier número entero.

//Almacenar un dato en una posición específica de la lista, por ejemplo, en la posición 3.

**nombreDeLista[3] = valorNuevo;**

//Acceder a los datos de una posición específica de la lista, por ejemplo, de la posición 3.

**funcionDeMiPrograma( nombreDeLista[3] );**

## Estructuras de Control:

Son estructuras que controlan la forma en que el programa se ejecuta, modificando su dirección o flujo. Dependen, para su ejecución, de condiciones, las cuales deben cumplirse (tener un valor de Verdadero o *true*). Si la condición no se cumple, las sentencias contenidas en el bloque de la estructura de control no se ejecutan.

En general, las condiciones son **comparaciones entre valores** (que pueden provenir de variables, por ejemplo), o sea, que se debe comparar si tal o cual valor es *igual, menor o mayor* que tal otro valor. Si se cumple la condición, se dice que **la condición es verdadera**, si no se cumple, entonces decimos que **la condición es falsa**.

Hay tres estructuras de control básicas:

### IF (Condicional)

Genera un bloque cuyas sentencias **solo se ejecutan si la condición resulta *verdadera***.

Puede incluir un bloque secundario (declarado mediante la palabra clave ***else***) que solo se ejecuta si la condición es falsa.

//Declaración sencilla

**if( condición ){**

**//este bloque solo se ejecuta si la condición resulta verdadera**

**}**

//Declaración completa

**if( condición ){**

**//este bloque solo se ejecuta si la condición resulta verdadera**

**}**

**else{**

**//este bloque solo se ejecuta si la condición resulta falsa**

**}**

### WHILE (Bucle controlado por condición)

Genera un bloque cuyas sentencias **se ejecutan repetidamente** (lo que se denomina *en bucle*) **mientras la condición resulta *verdadera*.**

Tras cada ejecución completa del bloque, la condición se vuelve a chequear. Si la condición sigue resultando verdadera, **se ejecuta el bloque nuevamente**. Si la condición resulta falsa, **el bloque no se ejecuta** y la ejecución sale de la estructura del *while.*

Un *while* mal declarado puede generar un ***bucle infinito:*** las sentencias del bloque se ejecutan de forma repetitiva, sin control y sin forma interna de detenerse, consumiendo recursos de forma descontrolada. La única forma de detenerlo es “matando” el proceso desde una terminal del Sistema Operativo.

//Declaración de un ***while***

**while( condición ){**

//sentencias que se ejecutarán repetidamente

**}**

### FOR (Bucle controlado por contador)

Genera un bloque cuyas sentencias se ejecutan repetidamente **mientras el valor de una variable numérica** (que llamaremos “contador”) **no alcance un valor “tope”,** que se establece en la declaración del*for***.**

Tras cada ejecución completa del bloque, se le suma al valor contenido en la variable “contador” una cantidad determinada (por lo general 1), que también se establece en la declaración del *for.* Al comenzar una nueva ejecución del bucle, se compara el valor del contador con el valor del tope. Si la comparación sigue resultando verdadera, **se ejecuta el bloque nuevamente**. Si la comparación resulta falsa, **el bloque no se ejecuta** y la ejecución sale de la estructura del *for.*

La variable “contador” **siempre debe ser de tipo *int***, al igual que el valor del tope y la cantidad a sumarle al contador tras cada bucle. Las comparaciones entre el contador y el tope deben ser **<**, **>**, **<=**, o **>= .**

El valor de la variable “contador” **puede utilizarse en las operaciones y sentencias del bloque** para lograr diferentes efectos durante el bucle (como ser recorridas entre los valores de una lista). Una vez que el bucle *for* termina, el valor de la variable “contador” se pierde, **ya que es interna al bloque**. Para conservar ese valor se debe almacenar en una variable declarada previamente.

//Declaración de un **for** con contador que avanza desde 0 hasta 9

**for( int contador = 0; contador < 10; contador++ ){**

//sentencias que se ejecutarán repetidamente

}

 //Declaración de un **for** con contador que retrocede desde 9 hasta 0

**for( int contador = 9; contador > 0; contador-- ){**

//sentencias que se ejecutarán repetidamente

**}**

//Declaración de un **for** con contador que avanza desde 0 hasta 9, avanzando de dos en dos

**for( int contador = 0; contador < 10; contador = contador+2 ){**

//sentencias que se ejecutarán repetidamente

**}**

## Algoritmos Básicos:

Todos estos algoritmos deben ubicarse **dentro del bloque del método *main*** de una aplicación desarrollada en Java. Además, para que el ejemplo funcione, se debe incluir la biblioteca ***java.util.Scanner***, que permite (entre otras cosas) obtener los datos ingresados con el teclado.

El código debe verse parecido a este ejemplo al comenzar el desarrollo:

**import java.util.Scanner;**

**public class Ejemplo {**

**public static void main (String[] args) {**

//El código de los algoritmos **DEBE UBICARSE A PARTIR DE ESTA LÍNEA**

**}**

**}**

## Bucle Principal

Es un bucle *while* controlado por el valor de una variable externa. Durante la ejecución del bucle, se le solicita al usuario que realice una acción que modifica el valor de esa variable.

Una condición en la declaración del *while* chequea el valor de la variable, comparando ese valor con un valor establecido en la declaración. Si ese valor coincide con el valor establecido, el bucle *while* se cierra.

Este algoritmo es útil para desarrollar aplicaciones, ya que permite que el usuario controle cuándo se debe terminar la ejecución de la aplicación.

**Scanner teclado = new Scanner(System.in);**

//Declaramos un objeto llamado 'teclado' para obtener los datos del teclado

**int opcion = 0;**

//Declaramos una variable llamada 'opcion' para almacenar la opción ingresada

//por el usuario

**while( opcion != 1 ){**

//Declaramos el bucle while.

//La condición compara que la variable 'opcion' NO VALGA 1 para mantener

//el bucle en ejecución

**System.out.println("Escriba un número.");**

**System.out.println("Escriba 1 para salir.");**

//Le mostramos mensajes al usuario para indicarle qué acción requiere el programa

**opcion = teclado.nextInt();**

//Almacenamos en la variable 'opcion' el siguiente dato de tipo int que

//el usuario ingrese en el teclado

**}**

## Recorredor de Listas

Es un bucle (puede ser *for* o puede ser *while*) que avanza por cada una de las posiciones de una lista, realizando operaciones sobre los datos almacenados en cada posición. Es importante que los bucles sean declarados correctamente para que nuestro algoritmo no intente leer en una posición de la lista que no existe (por ejemplo, leer los datos de la posición número 5, en una lista de 5 posiciones, siendo que la última posición de esa lista tiene el número 4).

Si tenemos varias listas de la misma longitud, podemos usar un solo bucle para recorrerlas todas al mismo tiempo.

### Recorredor de Listas con bucle FOR

Si utilizamos un bucle *for*, debemos establecer el tope como **menor que la longitud de la lista** a recorrer. La variable ‘contador’ del bucle será la encargada de indicar, con su valor, en **cuál posición** de la lista estamos trabajando.

El ejemplo muestra un recorredor que **muestra en pantalla los datos contenidos en cada posición de la lista**.

**String[] lista = new String[5];**

//Declaramos una lista

**lista[0]="abc";**

**lista[1]="bcd";**

**lista[2]="cde";**

**lista[3]="def";**

**lista[4]="efg";**

//Le agregamos datos a la lista.

**for (int contador = 0; contador < 5; contador++) {**

//Declaramos el bucle 'for', teniendo cuidado en que el tope del contador

//sea igual a la longitud de la lista

**System.out.println( lista[contador] );**

//Mostramos en pantalla el elemento de la lista en la posición marcada

//por el contador

**}**