Nombre: Alvaro Josué Morales Rodríguez

Nombre de usuario en GitHub: ELFBIBv

Número de teléfono: 40905407

Carnet: 202203856

Correo personal: jm54336720@gmail.com

Correo institucional: 3021420020101@ingenieria.usac.edu.gt

Carrera: Ingeniería en Ciencias y Sistemas de la Universidad de San Carlos de Guatemala

Fecha: 11/09/2022

Parte 2

Investigación de las estructuras de sintaxis de java

1. Números relacionales

¿que son?
Sirven principalmente para comparar valores, permiten comprobar si un valor es mayor que (operador >), menor que (operador <), mayor o igual que (>=) y menor o igual que (<=).

Por ejemplo:

Al final el operador lo valida entre dos valores o variables con la estructura:

```
vble1 > vble2
vble1 < vble2
vble1 >= vble2
vble1 <= vble2
```

De esta forma podemos tener un código fuente que nos ayude a realizar estas validaciones de relación:

```
int vble1 = 5;
int vble2 = 3;

if (vble1 > vble2)
  System.out.println("La variable 1 es mayor que la variable 2");

if (vble1 < vble2)
  System.out.println("La variable 1 es menor que la variable 2");

if (vble1 >= vble2)
  System.out.println("La variable 1 es mayor o igual que la variable 2");

if (vble1 <= vble2)
  System.out.println("La variable 1 es menor o igual que la variable 2");
```

2. Operaciones aritméticas

Los operadores aritméticos en Java son los operadores que nos permiten realizar operaciones matemáticas siendo las siguientes: ***suma, resta, multiplicación, división y resto***, la de "operación de resto" puede llegar a ser poco conocida pero es el residuo de la división de una fracción división.

Los operadores aritméticos en Java los utilizaremos entre dos literales o variables y el resultado, normalmente lo asignaremos a una variable o bien lo evaluamos.

```
variable = (valor1|variable1) operador (valor2|variable2);
```

Así podemos tener los siguientes usos en el caso de que queramos asignar su valor.

```
suma = 3 + 7;             // Retorna 10
resta = 5 - 2;            // Retorna 3
multiplicacion = 3 * 2;   // Retorna 6
division = 4 / 2;         // Retorna 2
resto = 5 % 3;            // Retorna 2
```

Ten en cuenta que pueden ser valores o variables:

```
suma = vble1 + 3;   // Sumamos 3 al valor de la variable vble1
resta = vble1 - 4;  // Restamos 4 al valor de la variable vble1
...
```

O podríamos utilizarlo en una condición

```
if (variable > suma + 3) { ... }
```

En este caso no asignamos el resultado de la suma a una variable, solo lo evaluamos.

3. Operaciones lógicas

Los operadores lógicos están relacionados con el álgebra de Boole. Evalúan una expresión en la que están implicados uno o dos operandos con valor de tipo boolean. Retornan como resultando un valor lógico, que será true si la evaluación de la expresión es cierta o false en caso contrario.

Existen tres operadores lógicos:

Operador	Descripción	Sintaxis
&&	And lógico (y)	a && b
||	Or lógico (o)	a || b
!	Negación lógica (no)	!a

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	7 == 38	false
!=	distinto que	'a' != 'k'	true
<	menor que	'G' < 'B'	false
>	mayor que	'b' > 'a'	true
<=	menor o igual que	7.5 <= 7.38	false
>=	mayor o igual que	38 >= 7	true

Ejemplos de los operadores logicos

Operador and (&&)

Evalúa dos operandos de tipo lógico (pueden ser expresiones, variables o literales). Si ambos tienen valor true el resultado de la evaluación será true. Con que uno de los dos (o los dos) sea false, el resultado también lo será.

	true	false
true	true	false
false	false	false

Operador or (||)

Evalúa dos operandos de tipo lógico (pueden ser expresiones, variables o literales). Si al menos uno de los dos tiene valor true el resultado de la evaluación será true. Para que el resultado sea false ambos tendrán que serlo.

	true	false
true	true	true
false	true	false

Not (!)

El operador de negación evalúa un solo operando con valor lógico y devuelve como resultado el valor de este invertido. Es decir, si el operando tiene valor true este operador devolverá false y viceversa.

Valor real	Su negación
Verdadero	Falso
Falso	Verdadero

4. Ciclo For

La sintaxis de un ciclo for es simple en Java, en realidad en la mayoría de los lenguajes de alto nivel es incluso muy similar, de hecho, con tan solo tener bien claros los 3 componentes del ciclo for (inicio, final y tamaño de paso) tenemos prácticamente todo hecho

```
for(int i = valor inicial; i <= valor final; i = i + paso)
{
        ....
        ....
    Bloque de Instrucciones....
        ....
        ....
}
```

Por ejemplo

```
for(int i=500;i<=1000;i+=2)
{//Notemos que escribir i+=2 es similar a escribir i = i + 2
    System.out.println(i);
}
```

Lo que hará esto, será crear un ciclo de ir sumando 2 a la variable, como valor inicial tomara el numero 500, luego de eso lo escribira por el System.out.println(i); luego lo repitira pero le sumara 2 unidades por lo que el 500 tomara el valor de 502, luego lo escribira y hará lo mismo hasta que se deje de cumplir la condición y continuara con lo restante del código.

5. Ciclo While

¿Qué es?

Utilice la sentencia while para ejecutar en bucle un conjunto de instrucciones hasta que se cumpla una condición determinada.

¿Qué hace?

La sintaxis de la sentencia while consta de la palabra clave while seguida de una expresión booleana encerrada entre paréntesis. Esta expresión está seguida por un bloque de sentencias delimitado por llaves de cierre.

```
while (condición) { sentencias }
```

Donde condición es una expresión booleana y sentencias es una o varias sentencias. Por ejemplo:

```
I = 10;
while(I > 0) {
    Log("El valor de I es: " + I);
    I = I - 1;
}
```

Lo que hará es repetir la sentencia hasta que la condición deje de cumplirse, al igual que el ciclo For, pero a while se le puede poner una condición, y a for solamente un intervalo.

6. Ciclo Do While

Sintaxis del Ciclo Do-While en Java:

La sintaxis de un ciclo do-while es un tanto más larga que la del ciclo while en Java, sin embargo no se hace más complicado, de hecho con tan solo tener bien clara una condición de finalización para el ciclo tendremos prácticamente todo terminado.

```
do
{
        ....
        ....
    Bloque de Instrucciones....
        ....
        ....
}
while(condición de finalización); //por ejemplo numero != 23
```

Ejemplo

```
import java.util.Scanner;

public class CicloDoWhile
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int numero;

        do
        {
            System.out.println("Ingrese un numero: ");
            numero = sc.nextInt();
        }
        while(numero <= 500);
    }
}
```

El numero ingresado debe de ser menor a 500 para que la condición se siga ejecutando, si el numero ingresado es menor a 500 se repetirá el ciclo hasta que el numero ingresado sea mayor a 500 y se deje de cumplir la condición.

7. Tipos de Casteos

Nuevo tipo	Tipos origen
short	byte
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Tabla de conversiones numéricas implícitas en Java

Conversión de tipos explícita

En el casting explicito es tarea del programador especificar el nuevo tipo al que se va a transformar el dato. Se escribe de forma explícita entre paréntesis delante del dato.

Ejemplos

```
1  byte a = 20;
2  int x = (int) a;
```

Al escribir int entre paréntesis se fuerza a cambiar el dato de tipo byte a int.

Hay que tener cuidado al realizar esta conversión ya que se puede aplicar a tipos no compatibles, lo que puede derivar en perdidas de información e incluso errores en ejecución.

```
1  float x = 5.7F;
2  int y = (int) x;
```

Se realizará la conversión, pero se perderá la parte decimal del número con punto flotante al guardarlo en *y*. Solo se guardará 5 en *y*. Aunque parezca ilógico, habrá ocasiones en las que esta perdida de precisión nos pueden resultar útil.