UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
CURSO IPC
INGENIERÍA EN SISTEMAS
GRUPO #1

Catedrático: Bryan Prado Marroquín

INVESTIGACION PARTE 2

Nombre: PABLO ANDRES RODRIGUEZ LIMA

Carnet:

2	**0**	**2**	**2**	**–**	**0**	**1**	**9**	**4**	**7**

Investigue las siguientes estructuras en la sintaxis de java

- Operaciones Relacionales
  Los operadores relacionales evalúan dos operandos, de tipo primitivo (byte, int, char, float, etc.), relacionándolos entre si (de ahí su nombre).
  Devuelven un valor de tipo boolean. Dicho valor dependerá de si la relación (especificada por el operador) entre ambos operandos es cierta (true) o falsa (false).

  Mayor que (>)

El operador mayor que evalúa los dos operandos implicados y devuelve true si el valor del primero es mayor exclusivo que el segundo. Devuelve false en caso contrario, es decir, cuando el valor del primero sea menor o igual que el segundo.

  Mayor o igual que (>=)

El operador mayor o igual que evalúa los dos operandos implicados y devuelve true si el valor del primero es mayor o igual que el segundo. Devuelve false en caso contrario, es decir, cuando el valor del primero sea menor exlusivo. La diferencia con el operador anterior es que, cuando los valores de los operandos sean iguales también devolverá true.

  Menor que (<)

El operador menor que evalúa los dos operandos implicados y devuelve true si el valor del primero es menor exclusivo que el segundo. Devuelve false en caso contrario, es decir, cuando el valor del primero sea mayor o igual que el del segundo.

  Menor o igual que (<=)

El operador menor o igual que evalúa los dos operándos implicados y devuelve true si el valor del primero es menor o igual que el segundo. Devuelve false en caso contrario, es decir, cuando el primero sea menor exlusivo. La diferencia con el operador anterior es que, cuando los valores de los operándos sean iguales también devolverá true.

  Igual que (==)

El operador igual que evalúa los dos operandos implicados y si sus valores son iguales, devolverá valor true. En cualquier otro caso devolverá false.

  Distinto de (!=)

El operador distinto de es el opuesto al anterior. Evalúa los dos operandos implicados y devuelve true cuando sus valores son diferentes. Por lo tanto, devolverá false cuando sean iguales.

- Operaciones aritméticas

Los operadores aritméticos en Java son los operadores que nos permiten realizar operaciones matemáticas: suma, resta, multiplicación, división y resto.

Los operadores aritméticos en Java son:

Operador	Descripción
+	Operador de Suma. Concatena cadenas para la suma de String
-	Operador de Resta
*	Operador de Multiplicación
/	Operador de División
%	Operador de Resto

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);

- Operaciones Lógicas

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.

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.

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.

- Ciclo For

¿Cómo funciona un Ciclo For?

Para comprender mejor el funcionamiento del ciclo for, solucionemos un pequeño ejercicio práctico, supongamos que queremos mostrar los números pares (múltiplos de dos :P) entre el 500 y el 1000. Con esta información inmediatamente podemos determinar que por medio de un ciclo debemos mostrar una serie de números como la siguiente: 500 502 504 506 ... 600 ... 852 ... 906 ... 980 .. 1000. Tenemos entonces todo lo necesario para nuestro ciclo. Tenemos un valor inicial que sería el 500, un valor final (1000) y tenemos un tamaño de paso que es 2 (los números pares). Estamos ahora en capacidad de determinar los componentes esenciales para un ciclo for. Vamos a ver ahora la sintaxis de un ciclo for en Java, así estaremos listos para usarlos en nuestros programas de ahora en adelante.

Sintaxis del Ciclo For en Java:

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

- Ciclo While

Los bucles do-while y while en Java te permiten, como su nombre indica (while significa mientras), repetir una acción en un bucle siempre y cuando se cumpla una condición booleana de control. Es posible que el código contenido en un bucle while no se ejecute, porque no se cumpla la condición. No obstante, el código contenido en un bucle do-while se ejecuta, por lo menos, una vez. Si quieres saber más sobre los bucles en Java, échale un ojo también al artículo que he publicado sobre los bucles for y for-each.

- Ciclo Do-While

La diferencia entre ambos es que, en el bucle while, la condición booleana se evalúa antes de que se ejecute el código. En cambio, en un bucle do-while primero se ejecuta una vez el código (do) y, a continuación, se evalúa la condición. Si esta no se cumple, el hilo de ejecución sale del bucle.

Condición booleana: ámbito de la variable

En un artículo anterior sobre las condiciones booleanas ya te expliqué las diversas estructuras que podía adoptar una condición booleana. Como sabes, las condiciones booleanas suelen aparecer antes del código que se ejecuta si esta se cumple. Una excepción ocurre, como acabas de ver, en el bucle do-while.

En el caso del bucle do-while, no puedes evaluar una referencia, o variables, declaradas dentro del mismo bucle.

Esto se debe a una razón obvia.

Aunque modificases el valor de la variable o del objeto al que apunta la referencia, con cada iteración esta referencia o variable se volvería a declarar, volviendo a adquirir su valor original.

- Tipos de Casteos

Casteos

Definición:

El casteo (casting) es un procedimiento para transformar una variable primitiva de un tipo a otro, o transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas.

Existen distintos tipos de casteo (casting) de acuerdo a si se utilizan tipos de datos o clases.

Casteo Implícito (Widening Casting)

El casteo implícito radica en que no se necesita escribir código para que se lleve a cabo. Ocurre cuando se realiza una conversión ancha – widening casting – es decir, cuando se coloca un valor pequeño en un contenedor grande.

Ejemplo:

```
//Define una variable de tipo int  con el valor 100

int numeroEntero = 100;

//Define una variable de tipo long a partir de un int

long numeroLargo = numero;
```

Casteo Explicito (Narrowing Casting)

El casteo explicito se produce cando se realiza una conversión estrecha – narrowing casting – es decir, cuando se coloca un valor grande en un contenedor pequeño. Son susceptibles de perdida de datos y deben realizarse a través de código fuente, de forma explicita.

Ejemplo:

```
//Define una variable del tipo int con el valor 250

int numeroEntero = 250;

//Define una variable del tipo short y castea la variable numero

short s = (short) numero;
```

Upcasting

El upcasting se produce a nivel objetos. Suponiendo que existe una clase Empleado y clase Ejecutivo, la cual es una subclase de esta. Un Ejecutivo entonces ES UN Empleado y se puede escribir de la siguiente forma:

```
//Instancia un ejecutivo en una variable de tipo Empleado

Empleado e1 = new Ejecutivo (“Maximo Dueño”, 2000);
```

A esta operación en donde un objeto de una clase derivada se asigna a una referencia cuyo tipo es alguna de las superclases se la denomina upcasting.

Cuando se realiza este tipo de operaciones, hay que tener cuidado porque para la referencia e1 no existen los atributos y métodos que se definieron en la clase Ejecutivo, aunque la referencia apunte a un objeto de este tipo.

Por el contrario, si ahora existe una variable de tipo Empleado y se desea acceder a los métodos de la clase derivada – suponiendo que contiene un método ejecutarPlanificacion() en la clase Ejecutivo – teniendo una referencia de una clase base, como en el ejemplo anterior, hay que convertir explícitamente la referencia de un tipo a otro. Esto se hace con el operador de cast de la siguiente forma:

```
//Instancia un ejecutivo en una variable de tipo Empleado

Empleado emp = new Ejecutivo (“Máximo Dueño”, 2000);

//Se convierte (o castea) la referencia de tipo

Ejecutivo ej = (Ejecutivo) emp;

//Usamos el método de la clase Ejecutivo

ej.ejecutarPlanificacion();
```

La expresión de la segunda línea convierte la referencia de tipo Empleado asignándola a una referencia de tipo Ejecutivo. Para el compilador es correcto porque Ejecutivo es una clase derivada de Empleado. En tiempo de ejecución la JVM convertirá la referencia si efectivamente emp apunta a un objeto de la clase Ejecutivo, caso contrario lanzara una excepción.

Por el contrario, si intenta realizar lo siguiente:

```
Empleado emp = new Empleado (“Mario Gómez”, 2000);

Ejecutivo ej = (Ejecutivo) emp;
```

No dará problemas de compilación, pero al ejecutar se pero al ejecutar se producirá un error ya que la referencia emp apunta a un objeto de la clase Empleado y no a un objeto de clase Ejecutivo.

CONCLUSION

- Las funciones lógicas en programación nos sirven para poder identificar cada acción a un grupo, facilitando así su interpretación en un algoritmo.

- La computadora y el lenguaje de programación es capaz de realizar operaciones básicas las cuales son determinadas por los signos que conocemos estas ayudan a las operaciones y en la aplicación de la vida diaria nos podría ayudar a la hora de hacer una factura

- Las aplicaciones lógicas pueden ayudar a la verificación de condiciones ya que si se exponen ante estas se puede generar y clasificar las ordenes según los datos estipulados esto nos ayuda y funciona como las personas a la hora de tomar una decisión ya que se le asigna un criterio para que pueda decidir.