FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

NOMBRE DEL PROFESOR: Ing. Héctor Manuel Quej Cosgaya
NOMBRE DE LA PRÁCTICA: Operadores y Expresiones
PRÁCTICA NÚM. [3]

LABORATORIO:	Centro de Ingeniería Computacional
MATERIA:	Lenguaje de Programación I
UNIDAD:	Subcompetencia II
TIEMPO:	2 horas

OBJETIVO:

Conocer los diferentes tipos de operadores que el lenguaje de programación Java pone a nuestra disposición, entender su funcionamiento e identificar las reglas los rigen.

MARCO TEÓRICO:

Los operadores son elementos de un lenguaje de programación que tienen una serie de funciones ya definidas, y que resultan de vital importancia para definir las instrucciones que conforman un programa. Los operadores resultan fundamentales a la hora de implementar los cálculos necesarios para la lógica de un programa, y por eso es importante aprender los diferentes tipos de operadores que el lenguaje pone a nuestra disposición y el funcionamiento de cada uno. En la práctica, se procede a mostrar un ejemplo del uso de los principales operadores de Java, así como explicar un poco acerca de lo que realizan.

LISTA DE MATERIALES:

- Java SDK
- Bloc de notas / Editor SciTE

EQUIPO DE LABORATORIO:

- Computadora Personal

DESARROLLO DE LA PRÁCTICA:

Primera parte**: Operadores Unarios**

1. Abre el **Bloc de Notas**.
2. Escribe el siguiente código fuente en el **Bloc de Notas**. ¡Recuerda copiarlo **exactamente** igual!

```
public class Practica3 {
   public static void main(String[] args) {
      // Primera parte : Operadores unarios
      // Operadores de incremento
      int x = 10;
      int y = 10;
      boolean b = false;
      System.out.println("Valor inicial de x : " + x);
      System.out.println("Valor inicial de y : " + y);
      System.out.println("Valor de X con pre-incremento : " + ++x);
      System.out.println("Valor de Y con pos-incremento : " + y++);
      System.out.println("Valor final de x : " + x);
      System.out.println("Valor final de y : " + y);
   }
}
```

3. **Guarda** el archivo como '**Practica3.java**'. Asegúrate que sea en una ubicación a la que puedas acceder fácilmente.
4. **Compila** y **ejecuta** el programa. ¿Qué notas?

Aunque al principio parece que el valor de Y no se altera, vemos que en realidad sí lo hace. Esta es la diferencia entre utilizar el **pre-incremento** y el **pos-incremento**. Si en lugar de **Incremento** utilizáramos el operador **Decremento**, el proceso seria el mismo.

5. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 2:

```
// Operadores positivo / negativo | complemento booleano
System.out.println("El negativo de x es : " + -x);
System.out.println("El complemento de b es : " + !b);
```

6. **Compila** y **ejecuta** el programa. ¿Qué observas?

Ambos operadores alteraron el valor de una variable, **Negativo** cambia el signo de un número, y **Complemento Booleano** el valor de un booleano.

7. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 5:

```
// Operador inversor a nivel de bits
x = 192;
System.out.println("El número 192 en binario es : " +
      Integer.toBinaryString(x));
```

```
System.out.println("El inverso de 192 en binario es : " +
      Integer.toBinaryString(~x));
System.out.println("El inverso de 192 es : " + ~x);
```

8. **Compila** y **ejecuta** el programa. ¿Qué sucede?

Aunque la primera cadena binaria parece un poco más corta, en realidad se omitieron todos los ceros que van antes del primer uno (los bits se leen de derecha a izquierda). Observa que cada 1 se cambia por un 0 y cada 0 por un 1.

9. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 7:

```
// Operador cast
x = (int) 5.9999999999; // (10 nueves)
System.out.println("Casteando un doble en entero : " + x);
```

10. **Compila** y **ejecuta** el programa. ¿Qué pasó?

Castear el valor **double** nos permite almacenarlo en una variable **int**, pero con un precio: toda la información de la parte fraccionaria **se pierde**. ¡Usa el casteo con cuidado!

Segunda parte**: Operadores binarios**

11. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 9:

```
// Segunda parte – Operadores binarios
// Operadores aritméticos
x = 5;
y = 5;
System.out.println("Suma : 5 + 5 = " + (x + y));
System.out.println("Resta : 5 – 5 = " + (x - y));
System.out.println("Multiplicación : 5 * 5 = " + (x * y));
System.out.println("División : 5 / 5 = " + (x / y));
System.out.println("Módulo : 5 % 5 = " + (x % y));
// Concatenación
String frase1 = "¡El operador Suma (+) ";
String frase2 = "también une cadenas de texto!";
System.out.println(frase1 + frase2);
// División entera
x = 7;
y = 4;
double z = x / y;    // El resultado real es 1.75
System.out.println("La división 7 / 4 es : " + z);
```

12. **Compila** y **ejecuta** el programa. ¿Los resultados son los que esperabas?

Los operadores aritméticos realizan las operaciones que ya conoces, el **módulo** es el único que puede ser nuevo para ti. Nota que el operador **Suma** puede utilizarse para unir cadenas de texto también: ¡lo hemos hecho desde el *principio*! ¿Lo habías notado? De igual manera, nota que cuando los dos operandos de un operador **División** son enteros, el resultado es una división entera: ¡la parte fraccionaria se pierde!

13. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 11:

```
// Operadores lógico-relacionales
b = (x > y);            // ¿7 mayor que 4? – true
boolean b2 = (x < y);   // ¿7 menor que 4? - false
boolean b3 = (x != y);  // ¿7 diferente de 4? – true
System.out.println(“ 7 > 4 AND 7 < 4? : “ + (b && b2));
System.out.println(“ 7 > 4 OR 7 < 4? : “ + (b || b2));
System.out.println(“ 7 > 4 AND 7 != 4? : “ + (b && b3));
System.out.println(“ 7 > 4 OR 7 != 4? : “ + (b || b3));
System.out.println(“¿frase1 es un String? ” +
      (frase1 instanceof String));
```

14. **Compila** y **ejecuta** el programa.

Primero utilizamos los operadores **Relacionales** para establecer algunas relaciones entre las variables X e Y, que almacenamos en variables booleanas. Luego utilizamos los operadores **Lógicos** para reducir dos valores booleanos a uno solo equivalente. También utilizamos el operador **instanceof** para verificar la clase de un objeto, en este caso, de frase1.

15. Añade el siguiente fragmento de código después de la última sentencia que escribiste en el paso 13:

```
// Operadores de Asignación
System.out.println(“¡Hemos estado utilizando el operador “ +
      “Asignación desde el principio! ¿Lo habías notado?”);
x = 10;
y = 10;
x += 20;
y = y + 20;
System.out.println(“Valor de x: “ + x);
System.out.println(“Valor de y: “ + y);
```

16. **Compila** y **ejecuta** el programa. ¿Notas alguna diferencia?

A pesar de que las operaciones son aparentemente diferentes, vemos que el resultado es el mismo. Los operadores de Asignación compuesta simplemente es una forma más compacta de escribir operaciones que modifican el valor de la misma variable.

17. Agrega el siguiente fragmento de código después de la última sentencia que escribiste en el paso 15:

```
// Operadores Bitwise
int byte1 = 51;
int byte2 = 112;
System.out.println("El byte1 en binario: "
      + Integer.toBinaryString(byte1));
System.out.println("El byte2 en binario: "
      + Integer.toBinaryString(byte2));
System.out.println("Operacion Bitwise AND: " +
      Integer.toBinaryString(byte1 & byte2));
System.out.println("Operacion Bitwise OR: " +
      Integer.toBinaryString(byte1 | byte2));
System.out.println("Operacion Bitwise XOR: " +
      Integer.toBinaryString(byte1 ^ byte2));
System.out.println("Desplazamiento 3 lugares a la izquierda: " +
      Integer.toBinaryString(byte1 << 3));
System.out.println("Desplazamiento 3 lugares a la derecha: " +
      Integer.toBinaryString(byte1 >> 3));
System.out.println("Desplazamiento sin signo 3 lugares a la " +
      "derecha: " + Integer.toBinaryString(byte1 >>> 3));
```

18. **Compila** y **ejecuta** el programa. ¿Entiendes lo que sucede?

Los operadores **Bitwise** funcionan con los bits que conforman a las variables enteras en Java.

Fin de la Práctica

RETROALIMENTACIÓN:

- Elabora un mapa conceptual acerca de los diferentes operadores disponibles en Java.
- Investiga las tablas de verdad de los operaciones lógicas negación, conjunción, disyunción, disyunción exclusiva, implicación y bicondicional (cualquier libro de Matemáticas Discretas incluye un capítulo acerca de Lógica). Elabora un programa que permita leer dos valores booleanos y aplique las operaciones mencionadas arriba. (Guarda ese programa, te servirá en tercero ;))

RECOMENDACIONES ADICIONALES:
• Leer el capítulo acerca de Lógica de cualquier libro de Matemáticas Discretas.

BIBLIOGRAFÍA:
• Dean, J. S., & Dean, R. H. (2009). Introducción a la programación con Java. México: Mc Graw Hill. • Roberts, Simon; Heller Philip y Ernest, Michael (1999). The Complete Java 2 Certification Study Guide. Alameda, California: SYBEX.