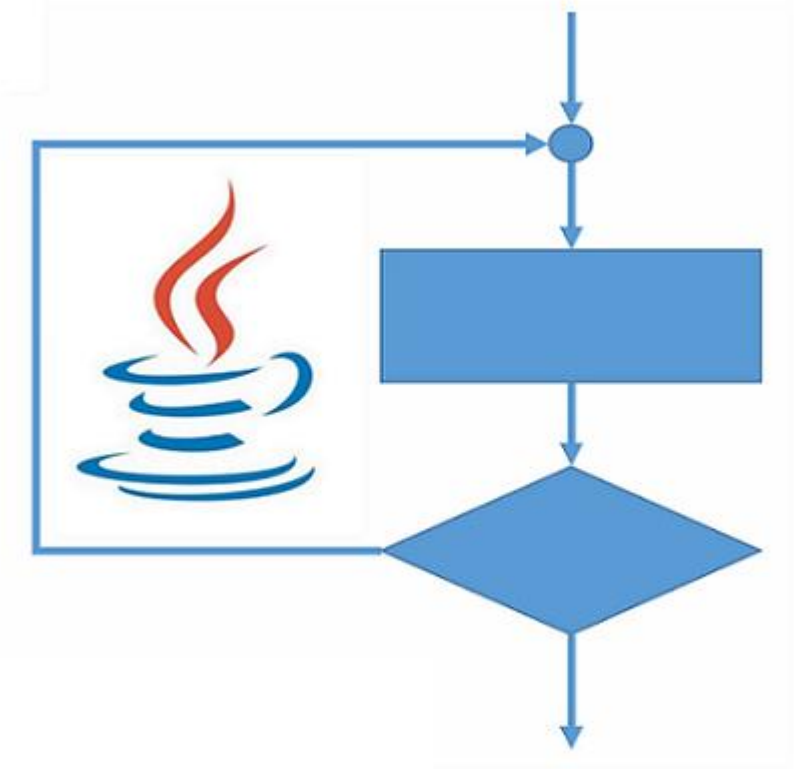
GUSTAVO CORONEL
DESARROLLA SOFTWARE

FUNDAMENTOS DE PROGRAMACIÓN CON JAVA



UNIDAD 04
FUNDAMENTOS DEL LENGUAJE

Eric Gustavo Coronel Castillo
youtube.com/DesarrollaSoftware
gcoronel@uni.edu.pe

INDICE

IDENTIFICADORES

Un identificador es un nombre que se le da a una variable, clase, o método. Para crear identificadores debes tener en cuenta las siguientes reglas:

1. Deben iniciar con una letra, el carácter subrayado (_), ó el signo dólar ($).
2. Los siguientes caracteres pueden también contener dígitos.
3. Los identificares son case-sensitive, esto quiere decir que hay diferencia entre mayúsculas y minúsculas.
4. No se pueden utilizar palabras claves.

Los siguientes con ejemplos de identificadores validos:

- nombre_usuario
- $tipo
- nombreUsuario
- articulo
- _sist_fecha

Los siguientes son ejemplos de identificadores no validos:

- 56_tema
- nombre@articulo
- tema-siguiente

PALABRAS CLAVES E IDENTIFICADORES RESERVADOS

Palabras claves e Identificadores reservados que son predefinidos en Java no pueden ser utilizados para definir nombres de variables, clases, o métodos. Todas estas palabras claves están en minúsculas y su uso incorrecto generan errores de compilación.

Palabras Claves

abstract	default	implements	protected	throw
assert	do	import	public	throws
bolean	double	instanceof	return	transient
break	else	int	short	try
byte	extends	interface	static	void
case	final	long	strictfp	volatile
match	finally	native	super	while
char	flota	new	switch	
class	for	package	synchronized	
continue	if	private	this	

Identificadores Reservados

null	true	false

Ejemplo 1

Identifica el error en el siguiente ejemplo.

```
public class Ejemplo01
{
   public static void main(String[] args)
   {
      int new;
      new = "Viva el Peru";
      System.out.println("Mensaje: " + new);
   }
}
```

COMENTARIOS

Los comentarios le ayudarán a documentar de manera muy precisa sus programas.

Los tres estilos permitidos para insertan comentarios son los siguientes:

Comentarios en línea

```
// Este es un comentario
```

Comentarios en una o más líneas

```
/*
Este es un comentario
en varias líneas
*/
```

Comentarios de documentación

```
/*
 *  Esta clase implementa un mantenimiento de tabla
 *  @autor Eric Gustavo Coronel Castillo
 *  @version 1.0
 */
```

PUNTO Y COMA, BLOQUES, Y ESPACIOS EN BLANCO

En Java, una sentencia es una o más líneas de código finalizadas con punto y coma (;).

Por ejemplo:

```
importe = a + b + c + d + e + f + g;
```

Es lo mismo que:

```
importe = a + b + c +
   d + e + f + g;
```

Un bloque, a veces llamado declaración compuesta, es un grupo de sentencias delimitadas por llaves ({ }).

El siguiente es un ejemplo de un bloque simple:

```
{  // Inicio del bloque

   c = a + b;
   a = b;
   b = c

}  // Finalización del bloque
```

La definición de una clase está contenida dentro de un bloque, tal como se ilustra a continuación:

```
public class MiClase
{  Inicio del bloque

   . . .
   . . .

}  Finalización del bloque
```

Un bloque de sentencias puede estar contenida dentro de otro bloque de sentencias, tal como se ilustra a continuación:

```
while ( j < limite ) { // Inicio de bloque externo

   total = total + j;

   if ( total == maximo ) { // Inicio del bloque anidado
      acumulado = acumulado + total;
      total = 0;
   } // Fin del bloque anidado

   j = j + 1;

} // Fin del bloque externo
```

También puede usar espacios en blanco entre los elementos del código fuente, tal como se ilustra a continuación:

```
{ int        x;x=15*20;}
```

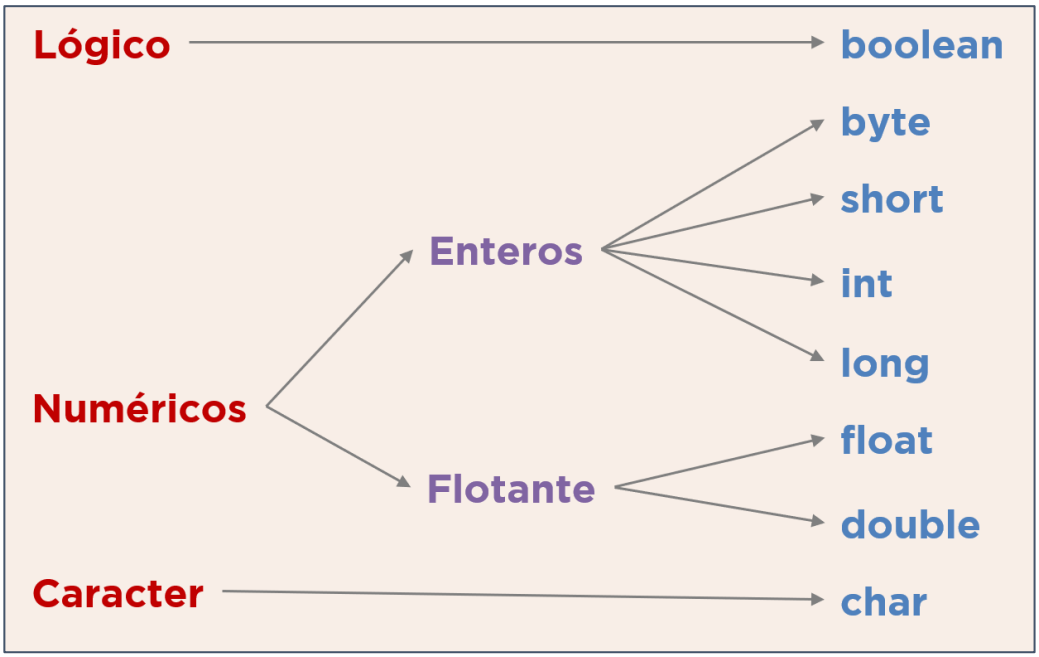
Es lo mismo que;

```
{
   int x;
   x = 15 * 20;
}
```

Es importante tener un código bien formateado para que sea de fácil lectura.

TIPOS DE DATOS PRIMITIVOS

Java define 8 tipos de datos primitivos tal como se ilustra en la siguiente imagen.



El tamaño efectivo de estos tipos de datos se puede apreciar en la siguiente imagen.

Tipo	Bits
boolean	1
byte	8
int	32
float	32

Tipo	Bits
char	16
short	16
long	64
double	64

En la siguiente tabla tenemos un resumen de los tipos de datos primitivos, donde podemos apreciar sus clases envolventes.

Tipo de Dato	Ancho en Bits	Valor Mínimo, Valor Máximo	Clase Envolvente
boolean	No Aplicable	true, false (valores discretos)	Boolean
byte	8	-2^7, 2^7-1	Byte
short	16	-2^{15}, $2^{15}-1$	Short
char	16	0x0, 0xffff	Character
int	32	-2^{31}, $2^{31}-1$	Integer
long	64	-2^{63}, $2^{63}-1$	Long
float	32	±1.40129846432481707e-45f, ±3.40282347663852886e+38f	Float
double	64	\'b14.94065645841246544e-324, \'b11.79769313486231570e+308	Double

Las clases envolventes de tipos primitivos describen el tipo de dato correspondiente, se trata de una clase estática, por lo tanto, no necesitamos instanciarla para poder utilizar sus métodos.

Tipo: boolean

Tipo lógico que representa dos valores: true y false. Una variable de este tipo solo puede tomar valor true o false.

Ejemplo 2

En el siguiente ejemplo se puede apreciar el uso del tipo boolean.

```
public class Ejemplo02
{
   public static void main(String[] args)
   {
      boolean pase = true;
      System.out.println("pase = " + Boolean.toString(pase));
   }
}
```

Tipo: char

Representa un carácter simple. Para representar un carácter debe encerrarlos entre apostrofes, tal como se aprecia en los siguientes ejemplos:

EJEMPLO	DESCRIPCIÓN
'a'	La letra **a**.
'\t'	Un tab.
'\u????'	Un carácter Unicode, ???? es reemplazado con exactamente cuatro dígitos hexadecimales, por ejemplo '\u03A6' es la letra Griega phi (ϕ).

El tipo **String** no es un tipo primitivo, es una clase que se utiliza para representar una secuencia de caracteres.

Ejemplo 3

En el siguiente ejemplo se aprecia el uso del tipo char.

```
public class Ejemplo03
{
   public static void main(String[] args)
   {
      char c1 = 'A', c2 = '\t', c3 = '\u03A6';
      String msg;
      msg = Character.toString(c1) +
         Character.toString(c2) +
         Character.toString(c3) ;
      System.out.println(msg);
   }
}
```

Tipos: byte, short, int, y long

Tenemos 4 tipos de datos enteros en Java. Podemos representar tipos enteros en formato decimal, octal, y hexadecimal, tal como se aprecia en el siguiente cuadro:

EJEMPLO	DESCRIPCIÓN
15	Es un número entero en formato decimal. Su valor es 15.
077	Es un número entero en formato octal. Su valor decimal es 63.
0xBAAC	Es un número entero en formato hexadecimal. Su valor decimal es 47788.

Todos los tipos enteros en Java representan números con signo.

Todos los literales enteros son de tipo **int** a menos que explícitamente se le ponga el sufijo **L**, que indica que se trata de un valor **long**, en el siguiente cuadro se pueden apreciar algunos ejemplos:

EJEMPLO	DESCRIPCIÓN
15L	Es un número entero en formato decimal tipo long. Su valor es 15.
077L	Es un número entero en formato octal tipo long. Su valor decimal es 63.
0xBAACL	Es un número entero en formato hexadecimal tipo long. Su valor decimal es 47788.

En la siguiente table puedes observar las características fundamentales de los tipos de datos enteros: tamaño y rango.

Tipo	Tamaño	Rango
byte	8 bits	-2^7 hasta 2^7-1
short	16 bits	-2^{15} hasta $2^{15}-1$
int	32 bits	-2^{31} hasta $2^{31}-1$
long	64 bits	-2^{63} hasta $2^{63}-1$

Ejemplo 4

En el siguiente ejemplo se puede apreciar el uso de tipos de datos enteros.

```
public class Ejemplo04
{
   public static void main(String[] args)
   {
      byte  n1 = 15;
      short n2 = 10000;
      int   n3 = 100000;
      System.out.println("n1 = " + Byte.toString(n1));
      System.out.println("n2 = " + Short.toString(n2));
      System.out.println("n3 = " + Integer.toString(n3));
   }
}
```

Tipos: flot y double

Tenemos 2 tipos de datos de punto flotante en Java. Un literal numérico es de tipo punto flotante si incluye un punto decimal, una parte exponencial (letra e o E), o esta seguida de las letras F o f (float), o D o d (double). Los valores de coma flotante son por defecto de tipo **double**. El siguiente cuadro muestra algunos ejemplos de números de coma flotante.

3.15	Un número de coma flotante simple, tipo double.
6.12E23	Un número de coma flotante grande.
2.718F	Un número de coma flotante simple.
123.4E+306D	Un número de coma flotante grande con **D** redundante.

En la siguiente tabla tenemos el tamaño en bits de cada tipo de dato de coma flotante.

Tipo	Tamaño
float	32 bits
double	64 bits

Ejemplo 5

En el siguiente ejemplo podemos apreciar el uso de tipos de datos de coma flotante.

```
public class Ejemplo05
{
   public static void main(String[] args)
   {
      float  n1 = 45.678f;
      double n2 = 1689.3483467;
      System.out.printf("n1 = %1$10.2f\n", n1 );
      System.out.printf("n2 = %1$.6e", n2 );
```

```
    }
}
```

VARIABLES Y OPERADORES

Variables

Para declarar variables debemos utilizar la siguiente sintaxis:

```
tipo nombre_variable [ = valor_inicial ], . . .  ;
```

Puedes observar que es posible asignar en la misma declaración un valor inicial, tal como se aprecia en el siguiente ejemplo.

Ejemplo 6

```
public class Ejemplo06
{
   public static void main(String[] args)
   {
      int a = 5, b = 10;
      int c;
      c = a * b;
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("c = " + c);
   }
}
```

Operadores Aritméticos

Operadores Unarios

Operador	Descripción	Ejemplo
+	Identificación de signo.	int n = -5; n = +n; // n toma valor -5
-	Negación de signo.	int n = -5; n = -n; // n toma valor +5

Operadores Binarios

Operador	Descripción	Ejemplo
+	Suma	int n1 = 5, n2 = 10, n3; n3 = n1 + n2; // n3 toma valor 15
-	Resta	int n1 = 5, n2 = 10, n3; n3 = n1 + n2; // n3 toma valor -5
*	Multiplicación	int n1 = 5, n2 = 10, n3; n3 = n1 * n2; // n3 toma valor 50

/	División	int n1 = 14, n2 = 4, n3;
		float n4;
		n3 = n1 / n2; // n3 toma valor 3
		n4 = (float)n1 / n2; // n4 toma valor 3.5

%	Resto	int n1 = 14, n2 = 4, n3; n3 = n1 % n2; // n3 toma valor 2

El comportamiento del operador de división depende de los operandos, si ambos son enteros la división será entera, si uno de los operandos es de punto flotante, entonces se tratará de una división real.

Ejemplo 7

En el siguiente ejemplo se ilustra el uso de operadores aritméticos.

```
public class Ejemplo07
{
   public static void main(String[] args)
   {
      int a, b, r1, r2, r3;
      float r4;
      a = Integer.parseInt( args[0] );
      b = Integer.parseInt( args[1] );
      r1 = a * b; /* Multiplicación */
      r2 = a / b; /* División Entera */
      r3 = a % b; /* Resto */
      r4 = (float)a / b; /* División Real */
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      System.out.println("Multiplicacion = " + r1);
      System.out.println("Division Entera = " + r2);
      System.out.println("Resto = " + r3);
      System.out.println("Division Real = " + r4);
   }
}
```

Operadores de Asignación

En el siguiente cuadro tenemos los diferentes operadores de asignación con los que contamos en Java.

OPERADOR	DESCRIPCIÓN	EJEMPLO

=	Asignación Simple	int a = 5;
		int b = a + 6;
		int c;
		c = a * b;

+=	Suma y Asignación	int a = 5, b = 10; b += a; // Equivalente a: b = b + a
-=	Resta y Asignación	int a = 5, b = 10; b -= a; // Equivalente a: b = b - a
***=**	Multiplicación y Asignación	int a = 5, b = 10; b *= a; // Equivalente a: b = b * a
/=	División y Asignación	int a = 5, b = 10; b /= a; // Equivalente a: b = b / a
%=	Resto y Asignación	int a = 5, b = 10; b %= a; // Equivalente a: b = b % a

Ejemplo 8

En el siguiente ejemplo se ilustra el uso del operador **+=**, puedes notar que el valor de **b** se incrementa en el valor de **a**.

```
public class prog0512
{
   public static void main(String[] args)
   {
      int a, b;
      a = Integer.parseInt( args[0] );
      b = Integer.parseInt( args[1] );
      System.out.println("a = " + a);
      System.out.println("b = " + b);
      b += a;
      System.out.println("Nuevo valor de b =  " + b);
   }
}
```

Expresión Lógica

Una expresión lógica puede tomar solo valores de tipo **boolean**, esto es valores **true** o **false**.

Las expresiones lógicas, cuando son utilizadas como condicionales en las instrucciones de control, permiten controlar el flujo del programa durante su ejecución.

Las expresiones lógicas se construyen con operadores de comparación, y operadores lógicos.

Operadores de Comparación

Los operadores de comparación permiten construir expresiones lógicas simples, ya que comparar dos valores dando un resultado lógico.

La siguiente tabla muestra los diferentes operadores de comparación con los que contamos en Java.:

OPERADOR	DESCRIPCIÓN	EJEMPLO
==	Igualdad	boolean a; a = 6 == 7; // Toma valor false
!=	Diferente	boolean a; a = '6' != 6; // Toma valor true
>	Mayor	boolean a; a = 6 > 7; // Toma valor false
>=	Mayor Que	boolean a; a = 'A' >= 'a'; // Toma valor true
<	Menor	boolean a; a = 20 < 50; // Toma valor true
<=	Menor Que	boolean a;

		a = 100 <= 100; // Toma valor true

Ejemplo 9

En el siguiente ejemplo se ilustra el uso del operador **==**, el programa recibe como dato de entrada un número entero, y nos informa si es par o impar.

```
public class prog0513
{
   public static void main(String[] args)
   {
      int n;
      boolean esPar;
      n = Integer.parseInt( args[0] );
      esPar = n % 2 == 0;
      System.out.println("n = " + n);
      System.out.println("Es Par = " + Boolean.toString(esPar));
   }
}
```

Operadores Lógicos

Los operadores lógicos permiten crear expresiones lógicas complejas, estos incluyen un operador unario **!** (complemento lógico), y tres operadores binarios: **&** (Y lógico), **|** (O lógico), **^** (O exclusivo).

Los operadores lógicos se aplican a operandos lógicos y su resultado es otro valor lógico.

Los operadores lógicos se definen en la siguiente tabla.

Operador	Descripción	Ejemplo
!	Complemento Lógico	boolean a; a = ! (6 == 7); // Toma valor true
&	Y Lógico	boolean a; a = (6 == 6) & (8<15); // Toma valor true
|	O Lógico	boolean a; a = (6 == 6) | (8<15); // Toma valor true
^	O Exclusivo	boolean a; a = (6 == 6) ^ (8<15); // Toma valor false

Asumiendo **x** e **y** representan dos expresiones lógicas, la siguiente tabla muestra un resumen del funcionamiento de los operadores lógicas:

x	y	!x	x & y	x | y	x ^ y
true	true	false	true	true	false
true	false	false	false	true	true

x	y	!x	x & y	x | y	x ^ y
false	true	true	false	true	true
false	false	true	false	false	false

Ejemplo 10

El siguiente ejemplo evalúa si un número entero es par y a la vez es múltiplo de 5. El número a evaluar se ingresa en la línea de comandos.

```
public class Ejemplo10
{
   public static void main(String[] args)
   {
      int n;
      boolean ok;
      n = Integer.parseInt( args[0] );
      ok = (n % 2 == 0) & (n % 5 == 0);
      System.out.println("n = " + n);
      System.out.println("Condicion = " + Boolean.toString(ok));
   }
}
```

Operadores de Asignación con Operadores Lógicos

También contamos con operadores de asignación compuestos por operadores lógicos, tal como se ilustra a continuación:

Operador	Descripción	Ejemplo
&=	Lógica "Y" y asignación.	boolean a = true, b = false; a &= b; // Equivale a: a = a & b
|=	Lógica "O" y asignación.	boolean a = true, b= false; a |= b; // Equivalente a: a = a | b
^=	Lógica "O" exclusiva y asignación.	boolean a = true, b = false; a ^= b;; // Equivalente a: a = a ^ b

Ejemplo 11

En el siguiente ejemplo verificamos si un número entero es múltiplo de 3 y 5 a la vez.

```
public class Ejemplo11
{
   public static void main(String[] args)
   {
      int n;
      boolean ok;
      n = Integer.parseInt(args[0]);
      ok = (n % 3 == 0);
      ok &= (n % 5 == 0);
      System.out.println("n = " + n);
      System.out.println("Condicion = " + Boolean.toString(ok));
   }
}
```

Operadores Condicionales

Los operadores **&&** y **||** son similares a sus equivales operadores lógicos **&** y **|**, tal como se ilustra en la siguiente tabla.

Operador	Descripción	Ejemplo
&&	Lógica “Y”	boolean ok; ok = (5 > 8) && (10 > 5); // Toma valor false
||	Lógica “O”	boolean ok; ok = (5 > 8) || (10 > 5); // Toma valor true

Las expresiones que utilizan estos operadores tienen la ventaja que finalizan en el momento en que se pueda determinar el valor de la expresión. Consideremos el siguiente ejemplo:

```
String cadena;
. . .
if( (cadena != null) && (cadena.length() > 0) ) {
   // Instrucciones
}
```

La expresión lógica de la estructura if() es correcta y totalmente segura, porque cuando la primera sub-expresión es falsa, la segunda sub-expresión no es evaluada por que toda la expresión es falsa. De forma similar ocurre con el operador ||.

Operador Condicional

Este operador permite definir una condición y en función a su resultado obtener un valor.

Sintaxis

```
condición ? expresión1 : expresión2
```

Si la **condición** es **true**, obtenemos el valor de la **expresión1**, en caso contrario obtenemos el valor de **expresión2**.

También es posible anidar este operador.

Ejemplo 12

En el presente ejemplo se hace uso del operador condicional para obtener el mayor de 2 números.

```
public class Ejemplo12
{
   public static void main(String[] args)
   {
      int n1, n2, mayor;
      n1 = Integer.parseInt(args[0]);
      n2 = Integer.parseInt(args[1]);
      mayor = (n1>n2)?n1:n2;
      System.out.println("n1 = " + n1);
      System.out.println("n2 = " + n2);
      System.out.println("mayor = " + mayor);
   }
}
```

Concatenación de Cadenas con +

El operador + permite concatenar objetos String, produciendo un nuevo String, tal como se ilustra en el siguiente ejemplo:

```
String equipo = "Alianza";
String condicion = "Campeon";
String frase = equipo + " " + condicion;
```

El resultado de la última línea es:

```
Alianza Campeon
```

Si uno de los operandos del operador + es un String, entonces el otro operando es convertido automáticamente a String. Todos los objetos pueden ser convertidos a un objeto String automáticamente, aunque el resultado puede ser algo críptico.

Transformación de Tipos: Casting

En muchas ocasiones hay que transformar una variable de un tipo a otro, por ejemplo de int a double, o de float a long.

La conversión entre tipos primitivos es más sencilla. En Java se realizan de modo automático conversiones implícitas de un tipo a otro de más precisión, por ejemplo, de int a long, de float a double, etc. Estas conversiones se hacen al mezclar variables de distintos tipos en expresiones matemáticas o al ejecutar sentencias de asignación, donde la variable a la izquierda del operador de asignación es de un tipo distinto (más amplio) que el resultado de evaluar la expresión a la derecha del operador de asignación.

Las conversiones de un tipo de mayor a otro de menor precisión requieren una instrucción explícita, pues son conversiones inseguras que pueden dar lugar a errores, por ejemplo, para pasar a short un número almacenado como int, hay que estar seguro de que puede ser representado con el número de cifras binarias de short. A estas conversiones explícitas de tipo se les llama cast. El cast se hace poniendo el tipo al que se desea transformar entre paréntesis, como, por ejemplo:

```
long r;
r = (long) (a/(b+c));
```

La expresión (long) hace que el resultado de la expresión a la derecha sea convertido a un valor de tipo long.

VIDEOS RECOMENDADOS

A continuación, tienes los enlaces de videos que para reforzar esta unidad.

DESCRIPCIÓN	ENLACE
Variables y tipos primitivos	https://youtu.be/edOvmNW_V9w
Operadores aritméticos	https://youtu.be/_TfpkRrFvX8
Operadores de comparación	https://youtu.be/dtcNuob2BQA
Clase Math	https://youtu.be/C8FIXMRtFKo

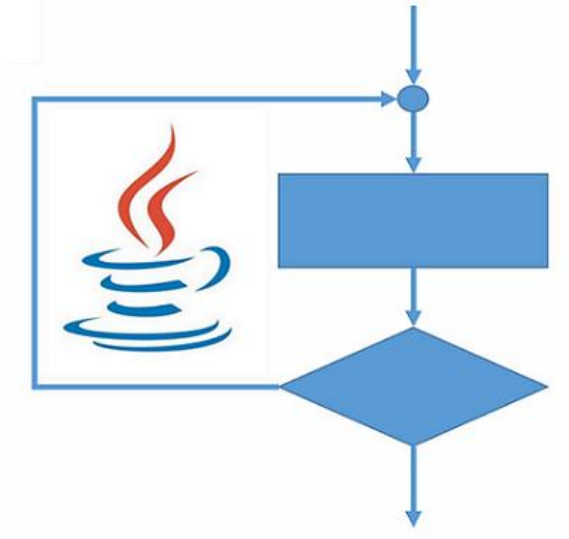
CURSOS VIRTUALES

CUPONES

En esta URL se publican cupones de descuento:

http://gcoronelc.github.io

FUNDAMENTOS DE PROGRAMACIÓN CON JAVA



Tener bases sólidas de programación muchas veces no es fácil, creo que es principalmente por que en algún momento de tu aprendizaje mezclas la entrada de datos con el proceso de los mismos, o mezclas el proceso con la salida o reporte, esto te lleva a utilizar malas prácticas de programación que luego te serán muy difíciles de superar.

En este curso aprenderás las mejores prácticas de programación para que te inicies con éxito en este competitivo mundo del desarrollo de software.

URL del Curso: **https://www.udemy.com/course/fund-java**

Avance del curso: **https://n9.cl/gcoronelc-fp-avance**

Cupones de descuento: **http://gcoronelc.github.io**

JAVA ORIENTADO A OBJETOS

CURSO PROFESIONAL DE JAVA ORIENTADO A OBJETOS

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
INSTRUCTOR

En este curso aprenderás a crear software aplicando la Orientación a Objetos, la programación en capas, el uso de patrones de software y Swing.

Cada tema está desarrollado con ejemplos que demuestran los conceptos teóricos y finalizan con un proyecto aplicativo.

URL del Curso: **https://bit.ly/2B3ixUW**

Avance del curso: **https://bit.ly/2RYGXIt**

Cupones de descuento: **http://gcoronelc.github.io**

PROGRAMACIÓN CON JAVA JDBC

PROGRAMACIÓN DE BASE DE DATOS ORACLE CON JAVA JDBC

Eric Gustavo Coronel Castillo
www.desarrollasoftware.com
INSTRUCTOR

En este curso aprenderás a programas bases de datos Oracle con JDBC utilizando los objetos Statement, PreparedStatement, CallableStatement y a programar transacciones correctamente teniendo en cuenta su rendimiento y concurrencia.

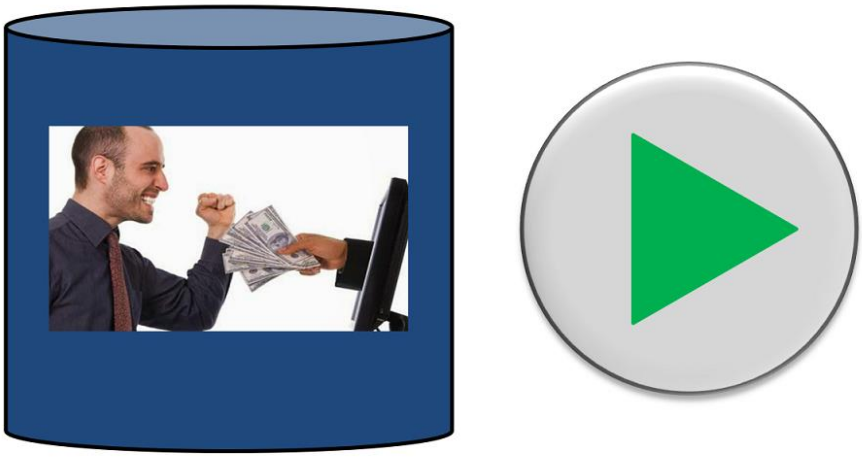
Al final del curso se integra todo lo desarrollado en una aplicación de escritorio.

URL del Curso: **https://bit.ly/31apy0O**

Avance del curso: **https://bit.ly/2vatZOT**

Cupones de descuento: **http://gcoronelc.github.io**

PROGRAMACIÓN CON ORACLE PL/SQL



En este curso aprenderás a programas las bases de datos ORACLE con PL/SQL, de esta manera estarás aprovechando las ventas que brinda este motor de base de datos y mejoraras el rendimiento de tus consultas, transacciones y la concurrencia.

Los procedimientos almacenados que desarrolles con PL/SQL se pueden ejecutarlo de Java, C#, PHP y otros lenguajes de programación.

URL del Curso: **https://bit.ly/2YZjfxT**

Avance del curso: **https://bit.ly/3bciqYb**

Cupones de descuento: **http://gcoronelc.github.io**