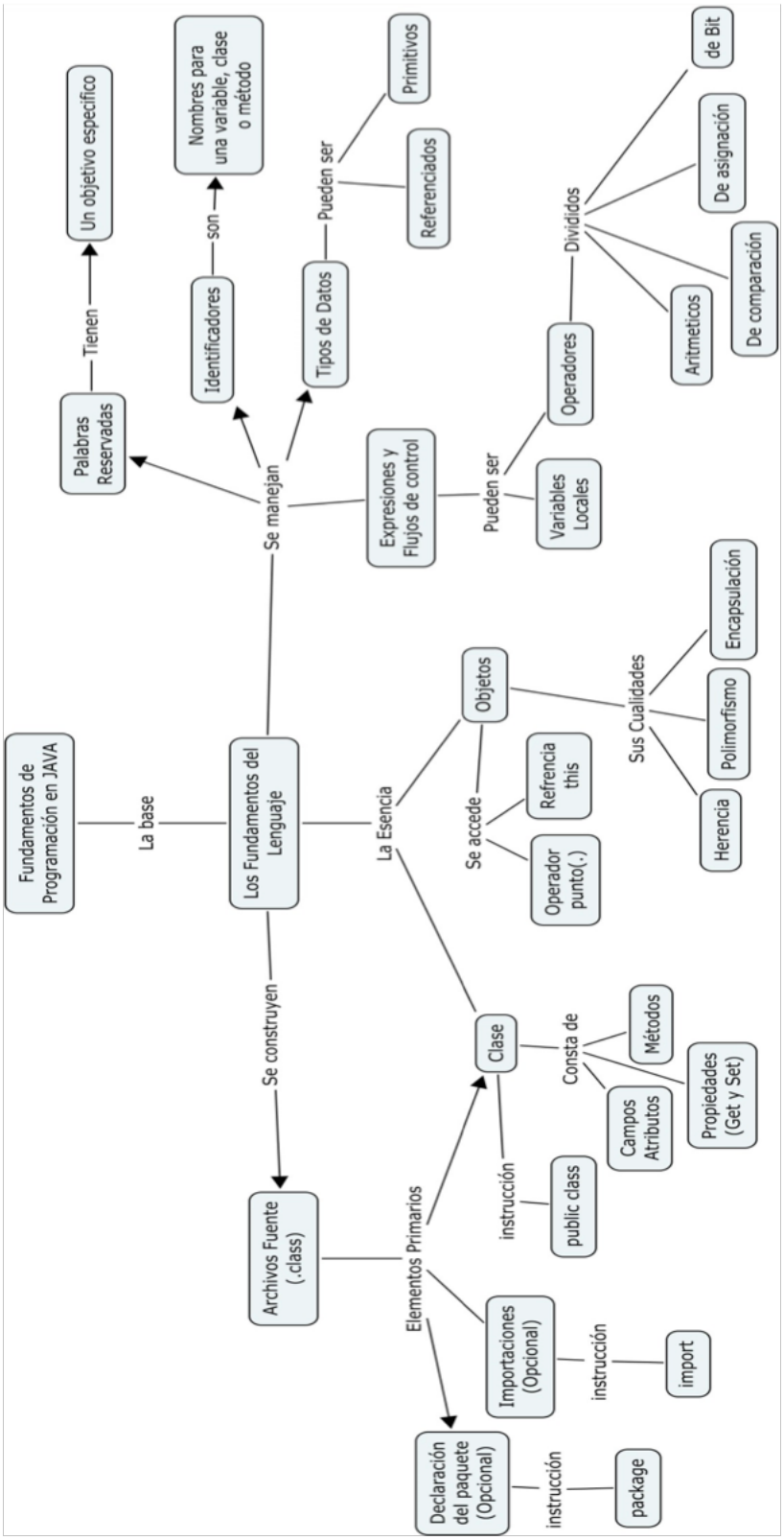
FUNDAMENTOS DEL LENGUAJE DE PROGRAMACIÓN JAVA

Mapa conceptual
Fundamentos del Lenguaje de programación JAVA



FUNDAMENTOS DEL LENGUAJE DE PROGRAMACIÓN JAVA

INTRODUCCIÓN

Este material busca que usted adquiera una buena fundamentación en cuanto al lenguaje JAVA se refiere, para ello se van a tomar como objetivos básicos los siguientes tópicos: fundamentos del lenguaje, declaración y control de acceso, Operadores y asignaciones, control de Flujo, Objetos, Clases y Manejo de Excepciones.

La apropiación de estos conocimientos, además de facilitar su proceso de aprendizaje, le permitirá pensar en un futuro presentarse a un proceso de certificación internacional en JAVA.

1. FUNDAMENTOS DE JAVA

Recordemos que en el material de introducción a JAVA se trataron términos como, Código fuente, compilador, Bytecode, API, JRE(Entorno de Ejecución) y Máquina Virtual(JVM), así como el proceso básico de codificación en JAVA.

Al momento de diseñar el archivo fuente (java class), se debe tener en cuenta los siguientes tres elementos de "primer nivel":

- **Una declaración de paquete (opcional):**

Un paquete contiene los enlaces con la toda la información de las clases. Se reconoce su uso mediante la palabra reservada package.

- **Cualquier declaración de importaciones(opcional):** Esta importación se realiza con la palabra import. Algunas de las clases importadas comúnmente son:

java.lang
java.awt
java.applet
java.net
java.io
java.util

Estas clases se encuentran en una serie de bibliotecas estándar, reconocida comúnmente como el API(Application Programming Interface) de JAVA. Estas bibliotecas van actualizándose en la medida que se lanza una nueva versión del JDK y son una referencia para los programadores en JAVA.

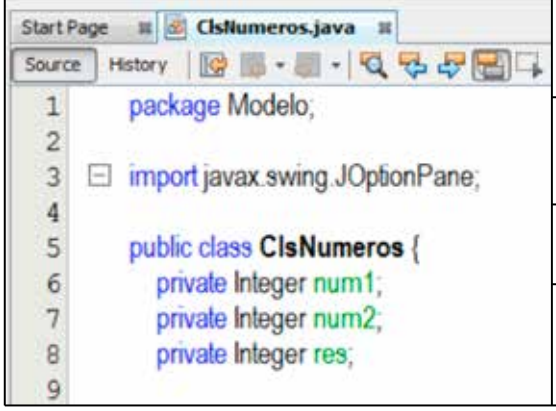
PAQUETE	NOMBRE	BREVE DESCRIPCIÓN
De utilidades	java.lang	Fundamental para el Lenguaje.
	java.io	Se utiliza para la entrada y salida de datos, así como para ficheros del sistema.
	java.util	Gracias a las colecciones de datos y clases, permite generación aleatoria de números y facilidades horarias.
	java.math	Esta clase permite manejar con precisión operaciones aritméticas básicas y avanzadas.
Para desarrollo gráfico	java.swing	Conjunto de componentes gráficos, los cuales funcionan igual en todas las plataformas JAVA.
	java.awt	Se utiliza para dibujar imágenes y gráficos
	java.awt	Para crear applets y sus respectivas clases de comunicación.

Para profundizar sobre el manejo de estas clases, puede consultar la siguiente dirección

http://docs.oracle.com/javase/6/docs/api/.

- Declaración de la clase o de la interface, normalmente se utiliza la declaración de la clase.

Revisemos la siguiente fracción de código:

IMAGEN	LÍNEA	COMENTARIO CÓDIGO
	1	La clase pública ClsNumeros se encuentra ubicada en el paquete denominado Modelo.
	3	Esta esta línea se está importando la herramienta JOptionPane, ubicada en la clase javax
	5	Se está inicializando la clase pública ClsNumeros
	6 al 8	Se están declarando tres variables de tipo Integer y con nivel de acceso privado. (private).

2. ELEMENTOS BÁSICOS DEL LENGUAJE

2.1. COMENTARIOS:

Recordemos que en la fase de codificación, es fundamental una adecuada documentación del código construido, y recordemos también los tres estilos permitidos por JAVA para los comentarios:

- Comentarios de Una sola línea:

  // comment on one line

- Comentarios de múltiples líneas:

  /* comment on one
  or more lines */

- Comentarios para documentación Automática:

  /** documenting comment */

2.2. PALABRAS RESERVADAS:

Estas palabras también son conocidas como palabras Clave, son aquellos identificadores reservados por Java para un objetivo específico. La tabla que se presenta a continuación relaciona algunas de las palabras reservadas; es importante aclarar que dependiendo de la versión de JAVA se tendrá el listado de palabras reservadas:

Abstract	boolean	break	byte	byvalue
Case	cast	catch	char	class
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new
null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	syncroniced	this
throw	throws	transient	true	try
var	Void	volatile	while	

Las palabras subrayadas son palabras reservadas que no se utilizan.

2.3. IDENTIFICADORES:

Recordemos que el nombre dado a una variable, clase o método se conoce como "Identificador". En el material de introducción a JAVA se trabajó este tópico y algunos ejemplos sobre la declaración de ellos son:

identifier
username
user_name
_sys_var1
$change

En esta ocasión, vamos a complementar nuestra declaración de variables aplicando los modificadores final y static. Estos modificadores se utilizan, para crear variables que no modifican su valor durante la ejecución del programa.

Las constantes, también pueden ser de tipo numérico, boolean, carácter o String.

Un ejemplo de declaración de constante declarada por nosotros puede ser:

```
final static int ALTURA_MAXIMA = 200;
```

Para que la variable sea total y absolutamente invariable	Variable de tipo numérica	Nombre de la Variable, Por ser una constante va TODO EN MAYÚSCULA

2.4 TIPOS DE DATOS

En JAVA se manejan dos tipos de datos, los primitivos y los referenciados.

2.4.1 Tipos de datos Primitivos:

Los tipos de datos primitivos se presentan en la siguiente tabla.

2.4.1. TIPOS DE DATOS PRIMITIVOS

Tipo	Tipo de Variable	Descripción	Bytes ocupados en memoria	Rango	Ejemplo
Lógicos	boolean	Para variables que tendrán la opción true o false	1 byte		boolean esColombiano=true;
Texto	char	Para variables que almacenan un solo carácter (letra, signo,?)	2 bytes		char sexo= ”m”
Enteros	byte	Para variables con valores enteros menores o iguales a 127	1 byte	-128 y 127	byte edad= 50
	short	Para variables con valores enteros menores o iguales a 32767	2 bytes	-32768 y 32767	short kilometrosdia= 1200
	int	Para variables con valores enteros menores o iguales a 2.147.483.647 Una tardeja de identidad no estaría en este rango	4 bytes	-2.147.483.648 y 2.147.483.647	int valorProducto= 500000
	long	Para variables con valores enteros meno-res o iguales a 9.223.372.036.854.775.807	8 bytes	-9.223.372.036.854.775.808 y 9.223.372.036.854.775.807	long gananciaAnual= 1147483648
Decimales	double	Números con unas 15 cifras decimales	8 bytes	De - 1,79769313486232E308 a - 4,9405645841247E324 y de 4,9405645841247E324E-324 a 1,79769313486232E308	double definitiva= 4.5134;

2.4.2 Tipos de datos Referenciados:

Los tipos de datos referenciados son cualquier variable declarada de un tipo no primitivo. Ejemplo de estos tipos de datos son: los arreglos, las clases y las interfaces.

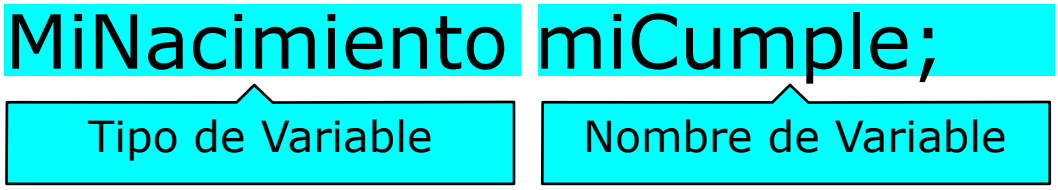
Creación y declaración de una variable de tipo referenciado

Un ejemplo de para la creación de este tipo de datos sería:

Dentro de la clase MiNacimiento

```
public class MiNacimiento {
int dia;
int mes;
int año;
}
```

Ahora, para declarar un tipo de dato Referenciado para la variable MiNacimiento, inicialmente se utilizará el siguiente código:



Si se desea crear otra variable de este mismo tipo, la instrucción a utilizar sería MiNacimiento otroCumple;

Acceso a una variable de tipo referenciado:

Como al crear una variable de tipo referenciado, podemos acceder a las variables que tiene la clase original (MiNacimiento), vamos a acceder a las variables dia, mes, año y a cada una de ellas les vamos a asignar valores. La codificación para asignar dichos valores seria:

```
micumple.dia=12;
micumple.mes=4;
micumple.año=1980;
```

Para poder utilizar en todo momento los valores asignados a estas variables, se hace necesario que el sistema las conozca, y para ello estas variables deben ser ubicadas en un espacio de memoria.

Analicemos el siguiente código:

```
1  package Modelo;
2  //Clase para visualizar la fecha de cumpleaños de una persona
3  public class MiNacimiento {
4   //Declaración de Variables
5    int dia;
6    int mes;
7    int año;
8
9    public static void main(String[] args)
10     MiNacimiento micumple = new MiNacimiento();//Creación del tipo de dato referenciado micumple                micumple.dia = 12; //Asignación de valor a la
11 variable dia
12     micumple.mes = 4;//Asignación de valor a la variable mes
13     micumple.año = 1980;//Asignación de valor a la variable año
14     System.out.print("Mi cumpleaños es el "+micumple.dia);//Visualización del valor de la variable dia
15     System.out.println(" del mes "+micumple.mes);//Visualización del valor de la variable mes
16    }
17   }
```

Este código al intentar ejecutarlo genera el siguiente mensaje.

```
run:
Exception in thread "main" java.lang.NullPointerException
        at Modelo.MiNacimiento.main(MiNacimiento.java:11)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Vamos a actuar un momento como computadoras para el código que hemos trabajado hasta el momento:

Código Construido	Comportamiento del Computador
public class MiNacimiento {	MiNacimiento
int dia;	dia
int mes;	mes
int año;	año
}	
MiNacimiento miCumple	MiCumple
micumple.dia=12;	NullPointerException
micumple.dia=4;	NullPointerException
micumple.año=1980;	NullPointerException
}	

Esto se debe a que el sistema no tiene espacio en memoria destinado a recibir los valores para estas variables

Instanciación de una variable de tipo referenciado

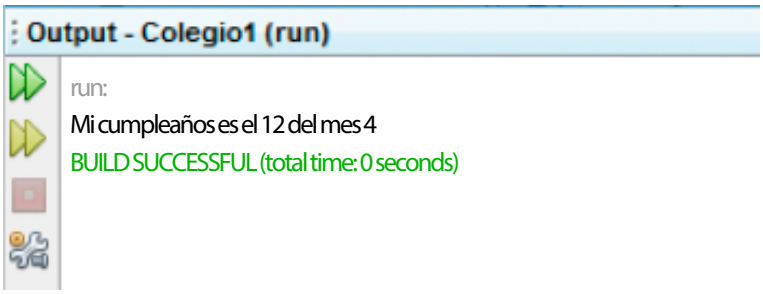
Crear los espacios en memoria para las variables a utilizar en los datos referenciados, es función de la palabra reservada new.

NEW, lo que hace es crear una copia exactamente igual de la clase origen, en nuestro caso MiNacimiento. Esto se conoce como "instanciación", a ese conjunto de variables que se crean se le denomina variables de instancia. La potencia de estas variables es que se obtiene un conjunto distinto de ellas cada vez que se crea un objeto nuevo.

Ahora veamos el mismo código con la implementación de la palabra reservada new.

```
1  package Modelo;
2  //Clase para visualizar la fecha de cumpleaños de una persona
3  public class MiNacimiento {
4   //Declaración de Variables
5    int dia;
6    int mes;
7    int año;
8
9    public static void main(String[] args)
10     MiNacimiento micumple = new MiNacimiento();//Creación del tipo de dato referenciado micumple          micumple.dia = 12; //Asignación de valor a la
11 variable dia
12     micumple.mes = 4;//Asignación de valor a la variable mes
13     micumple.año = 1980;//Asignación de valor a la variable año
14     System.out.print("Mi cumpleaños es el "+micumple.dia);//Visualización del valor de la variable dia
15     System.out.println(" del mes "+micumple.mes);//Visualización del valor de la variable mes
16    }
17   }
```

Al correr la clase el resultado será el siguiente:



Vamos a actuar nuevamente como computadoras, para el código que acabamos de trabajar:

Código Construido

```
public class MiNacimiento {
int dia;
int mes;
int año;
}
```

Comportamiento del Computador

MiNacimiento
dia
mes
año

Código Construido

```
MiNacimiento micumple=new MiNacimiento();
micumple.dia=12;
micumple.mes=4;
micumple.año=1980;
```

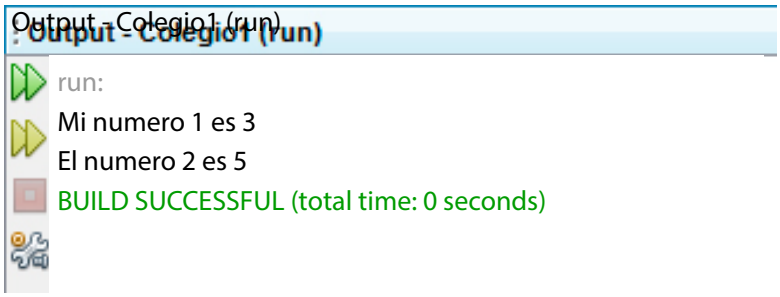
Comportamiento del Computador. New crea la estructura exactamente igual a la de MiNacimiento.

MiCumple
dia=12
mes=4
año=1980

Ejemplo de Acceso y utilización de una variable de tipo referenciado

```
1   package Modelo;
2
3   public class ClsNumeros {
4    //Declaración de variables
5
6    int num1;
7    int num2;
8
9   public static void main(String[] args) {
10    //Creación del tipo de dato referenciado misnumeros
11    ClsNumeros misnumeros = new ClsNumeros();
12
13    misnumeros.num1 = 3;//Asignación de valor a la variable num1
14    misnumeros.num2 = 5;//Asignación de valor a la variable num2
15    // Visualización del valor de la variables num1 y num2
16    System.out.println("El numero 1 es " + misnumeros.num1);
17    System.out.println("El numero 2 es " + misnumeros.num2);
18   }
19  }
20
```

La salida por sistema sería:



TENGA EN CUENTA:

Cuando necesite crear copia de una clase, utilice la palabra reservada NEW.

2.5 VARIABLES

Son aquellas que se definen dentro de un método y son llamadas variables locales, automáticas, temporales o variables de stack.

Se crean cuando el método es ejecutado y se destruyen cuando se finaliza el método.

Estas variables deben ser inicializadas antes de usarlas, porque de lo contrario ocurre un error en tiempo de compilación.

Variable	Inicialización
Byte	0
Short	0
Int	0
Long	0L
Float	0.0f
Doublé	0.0d
Char	´\u000´
Boolean	False
Referenciadas	null

2.6 OPERADORES:

Los operadores se utilizan para realizar operaciones entre objetos, datos, identificadores y/o constantes, y devuelven un valor.

Los operadores en JAVA se pueden dividir en las siguientes categorías:

- **OPERADORES ARITMETICOS**

Operador	Uso	Descripción	Ejemplo
+	op1+op2	Suma op1 y op2	5 + 7 = 12
-	op1-op2	Resta op1 y op2	7 - 5 = 2
*	op1*op2	Multiplica op1 y op2	2 * 2 = 4
/	op1/op2	Divide op1 y op2	6 / 3 = 2
%	op1%op2	Obtiene el resto de dividir op1 por op2	6 % 3 = 0

- **OPERADORES RELACIONALES**

Un operador relacional permite comparar dos valores y determina la relación existente entre ellos. El resultado de su utilización es siempre un valor lógico true o false.

Operador	Uso	Devuelve true
>	op1 > op2	Si op1 es mayor que op2
> =	op1 > = op2	Si op1 es mayor o igual que op2
<	op1 < op2	Si op1 es menor que op2
< =	op < =p2	Si op1 es menor o igual que op2
= =	op1 = = p2	Si p op1y p2 son iguales
! =	op1 ! = op2	Si op1 son op2 diferentes

• OPERADORES LÓGICOS

El uso de los operadores lógicos radica en la manera como queremos que se evalúe una condición compuesta. El comportamiento de los operadores lógicos básicos se muestra en la siguiente tabla:

Operador	Nombre	Uso	El resultado
&&	AND	op1 && op2	True si op1 y op2 son true. Si op1 es false YA no se evalúa op2
||	OR	op1 || op2	True si op1 u op2 son true. Si op1 es true YA no se evalúa op2.
!	NEGACIÓN	! op	True si op false y false si op es true
&	AND	op1 % op2	True si op1 y op2 son true. Siempre se evalúa op2.
|	OR	op1 | p2	True si op1 u op2 son true. Siempre se evalúa op2.

• OPERADORES DE BIT

El operador de asignación básico es el igual (=) y se utiliza para asignar un valor a otro. Un ejemplo de este operador sería:

int num1=0; Inicializar la variable num1 en 0, el tipo de esta variable es entero.

El objetivo de los operadores de asignación es minimizar la escritura de código, y para esto JAVA proporciona varios operadores. Mediante la utilización de estos operadores se pueden realizar operaciones aritméticas, lógicas, de bit y de asignación con un único operador.

Supongamos que necesitamos sumar un número a una variable y almacenar el resultado en la misma variable, como a continuación, lo que en algoritmos conocimos como acumulador:

acu = acu + 2;

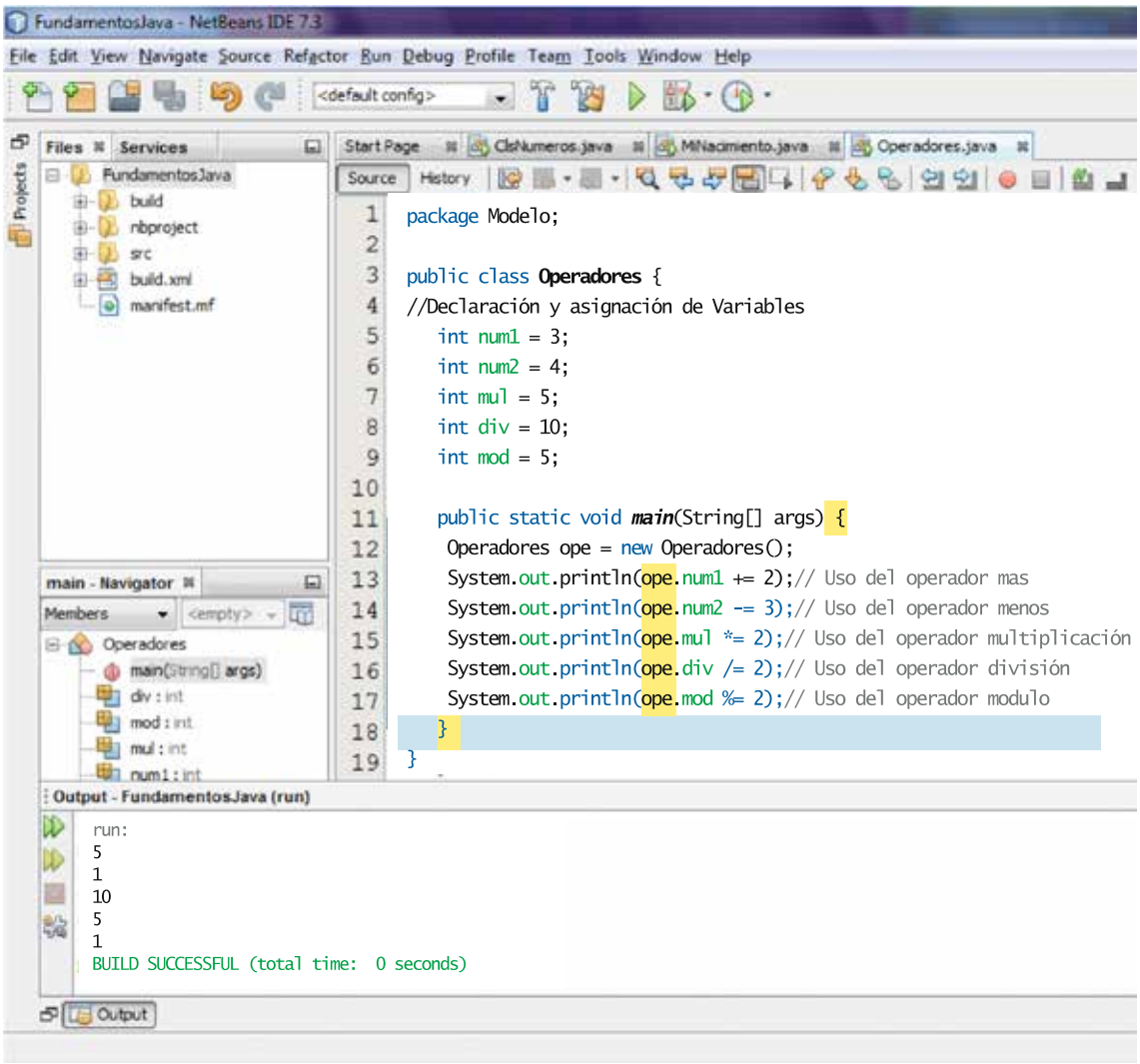
En JAVA se puede abreviar esta sentencia con el operador de atajo +=, de la siguiente manera:

acu += 2;

La siguiente tabla muestra los operadores de asignación y su expresión equivalente:

Operador	Uso	Expresión Equivalente	Ejemplo
+ =	op1 + = op2	op1 = op1 + op2	Para num1 = 3 num1 + = 2
- =	op1 - = op2	op1 = op1 - op2	Para num2 = 4 num2 - = 3
* =	op1 * = op2	op1 = op1 * op2	Para mul = 5 mul* = 2
/ =	op1 / = op2	op1 = op1 / op2	Para div = 10 div/ = 2
% =	op1 % = op2	op1 = op1 % op2	Para mod = 5 mod% = 2

La siguiente pantalla muestra ejemplos para la codificación de estos operadores y el resultado de las operaciones



3 OBJETOS Y CLASES

3.1 Definiciones

Para este tema necesitamos tomar como referencia algunos términos fundamentales como:

Clase: En general, una clase es una representación abstracta de algo. En inglés, se definiría como "is a template or model". Un ejemplo de una clase sería "Carro". Para crear una clase solo se necesita un archivo que contenga la palabra clave reservada class, seguía de un nombre permitido y un bloque delimitado por dos llaves para el cuerpo de la clase.

Un ejemplo sería: public class ClsDefinitiva { }

Una clase consta de: **campos, propiedades, métodos y eventos.**

Los **campos** son similares a las variables debido a que se pueden leer o definir directamente, un ejemplo de campos sería: color, marca, número de puertas, cilindraje, etc.

En términos de nuestro día a día, los campos serían los datos que tienen en común la mayoría de los carros.

Las **propiedades** se definen igual que los campos, y se implementan mediante los procedimientos de las propiedades Get y Set.

Los **métodos** representan las acciones que puede realizar un objeto. En inglés, un método se podría definir como "the operations for a class and Methods must belong to a class."

Por ejemplo, La clase "Carro" puede tener definidos los métodos "Arranque," "Velocidad" y "Parada".

Objeto: Es un ejemplo utilizable de ese algo representado por la clase. En ingles se podría definir como "An object is created based on that model"

Un ejemplo de un objeto sería: "MiCarro"

3.2 ACCESO A LOS OBJETOS

3.2.1 El operador punto (.)

Se utiliza para acceder a las variables de instancia y a los métodos contenidos en un objeto, mediante su referencia a objeto.

Sintaxis:

referencia_a_objeto.nombre_de_variable_de_instancia

Un ejemplo de la codificación con el operador punto sería:

```
MiNacimiento micumple = new MiNacimiento
micumple.dia = 12;
```

3.2.2 La Referencia this

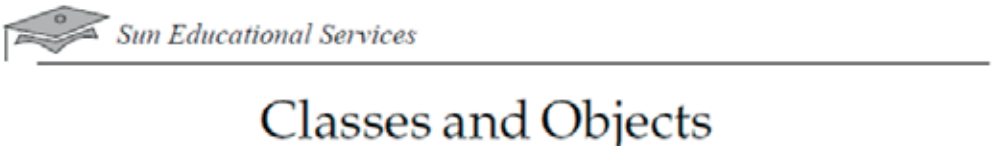
Dentro de los valores por referencia, Java incluye uno especial llamado "this". El valor this se utiliza dentro de cualquier método, para referirse al objeto actual y se refiere al objeto sobre el que ha sido llamado el método actual.

Se puede utilizar "this" siempre que se requiera una referencia a un objeto del tipo de una clase actual.

Un ejemplo de este valor sería:

```
public void setNot1(double not1) {
    this.not1 = not1;
```

Con esta instrucción modificamos la variable de instancia not1

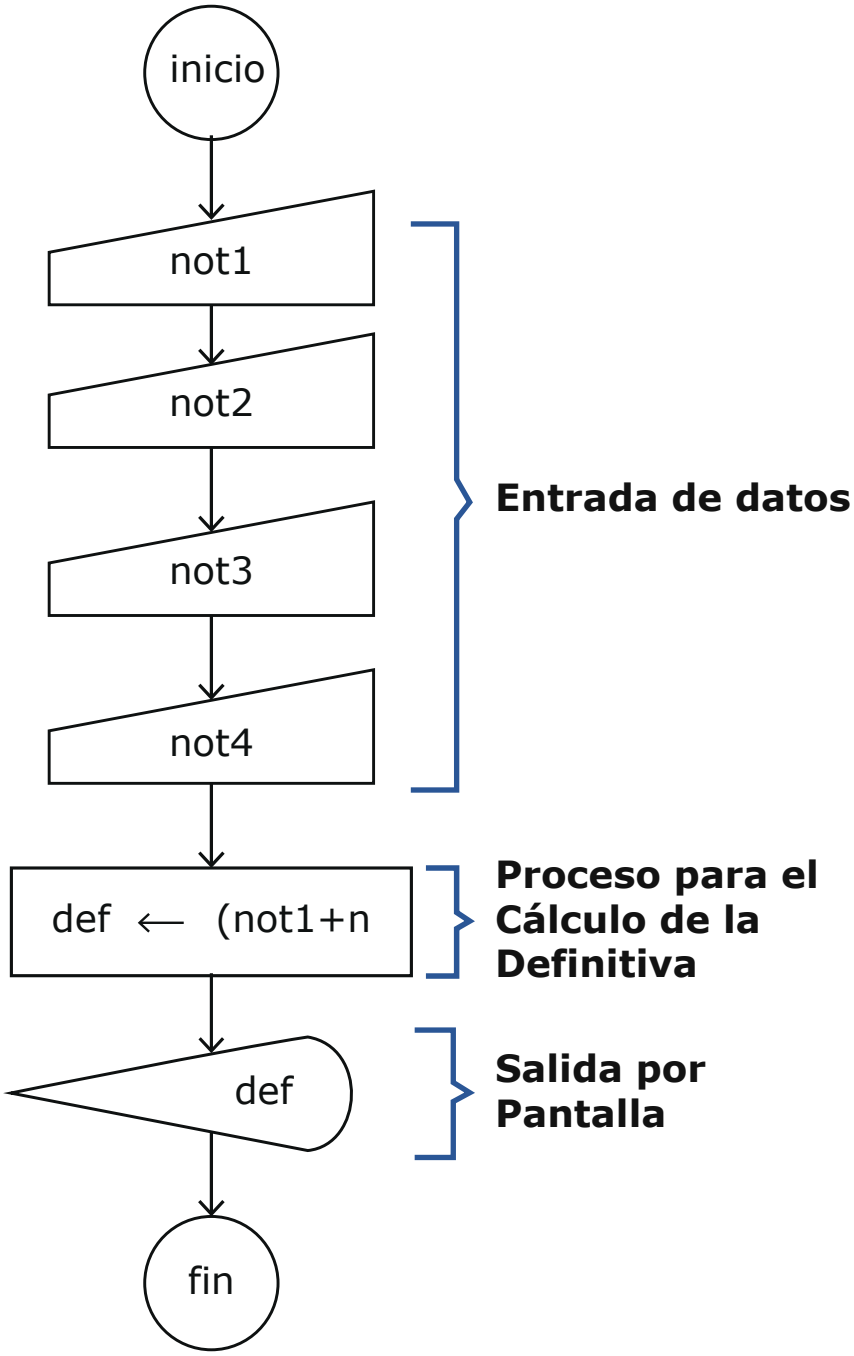


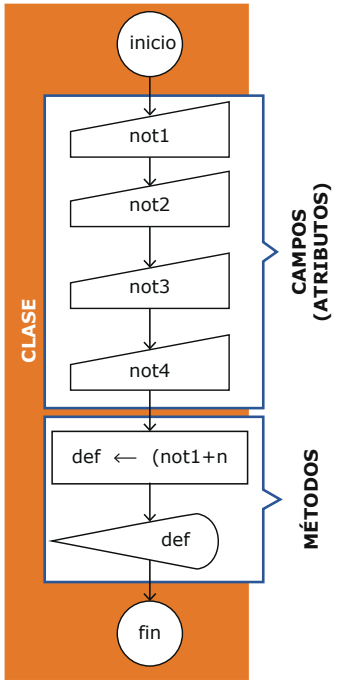

```
1   class EmpInfo {                                  Clase
2     String name;
3     String designation;                            Campos o Atributos
4     String departament;
5   }
6
7   // Create instance                               Creación del Objeto
8   EmpInfo employee = new EmpInfo();
9
10  // Intializes the three members
11  employee.name = "Robert Javaman"; employee.designation = "Manager";
12  employee.department = "Coffee Shop";
13
14  System-out-println(employee.name + "is" +
15                  employee.designation + "at" +
16                  employee.department);
```

Para apropiar el tema de clases, objetos y para iniciar el manejo de métodos tomemos como referente el mundo académico, donde normalmente se calcula la definitiva de una materia a la cual se le aplican cuatro notas, recordemos un poco el tema de Algoritmos. Una propuesta de este algoritmo sería:

Datos de entrada: nota1, nota2,nota3,nota4
Proceso: cálculo para la definitiva
Dato de Salida: definitiva

El Diagrama de flujo sería:


inicio
not1
not2
not3
not4
Entrada de datos
def ← (not1+n
Proceso para el Cálculo de la Definitiva
def
Salida por Pantalla
fin




```
1  package Modelo;
2
3  public class ClsDefinitiva {
4  //Declaración de Campos
5   double not1;
6   double not2;
7   double not3;
8   double not4;
9   double def; //Esta variable se declara porque es la que se va a dedicar
10        para obtener el resultado
11 //Declaración de Métodos
12 public void definitiva(){
13  def=(not1+not2+not3+not4)/4;
14  System.out.println("La definitiva de la materia es"+def);
15  }
16
17 public static void main (String[]args) {
18 ClsDefinitiva def=new ClsDefinitiva();
19 def.not1=4.2;
20 def.not2=3.3;
21 def.not3=4.1;
22 def.not4=3.3;
23 def.definitiva();
24  }
25 }
```

Explicación Líneas de Código en JAVA	
Línea	**Explicación**
3	Explicación del Código
5 al 9	Declaración de Variables. Recordemos que dependiendo del tipo de dato a utilizar se declara la variable. Para este caso fue doublé porque los datos a recibir son decimales.
12	Declaración del método
13	Proceso a realizar en el método, para este caso el cálculo de la definitiva.
14	Presentación por pantalla del resultado de la operación.
15	Finalización del mètodo
17	Declaración del método main. Recordemos que el método main es aquel que permite ejecutar la clase.
18	Creación del objeto. Para este caso el nombre del objeto es def
19 al 22	Asignación de valores a cada uno de los campos, es decir a los datos de entrada del algoritmo.
23	Llamar al método, para que el sistema ejecute los procesos incluidos en él. Para este caso líneas de código 13 y 14.
24	Finalización del método main.
25	Finalización de la clase.

DEFINIENDO METODOS

La declaración de los métodos tiene la siguiente estructura:

[ ]: Argumentos opcionales
< >: Argumentos obligatorios

```
<modifiers> <return_type> <name> ([<argument_list>])
[throws <exception>] {
< block >
}
```

Algunos ejemplos para la definición de los métodos podría ser:

EJEMPLO 1

```
        public  void definitiva(){
 def=(not1+not2+not3+not4)/4;
    System.out.println("La definitiva de la materia es "+ def);
  }
```

Este método no contiene argumentos porque los paréntesis que están después del nombre del método están vacíos.

De igual manera este método no retorna ningún dato, porque es de tipo void. Solo visualiza información.

EJEMPLO 2

```
        public double getNot1() {
    return not1;
  }
```

Este método retorna un dato de tipo double. Esto se sabe en el momento de la declaración public "double".

Este método No tiene argumentos.

TENGA EN CUENTA:

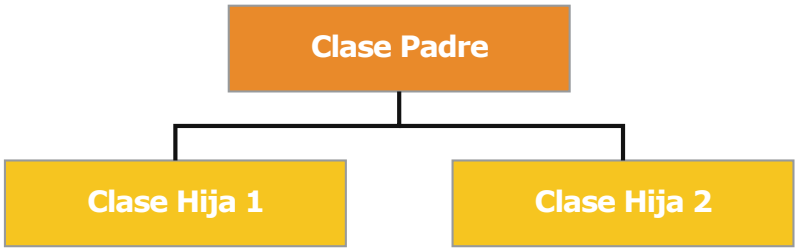
Todo método que maneje un <return_type> diferente de void, debe retornar algún dato, es decir debe incluir la instrucción return

3.3 CUALIDADES DE LOS OBJETOS

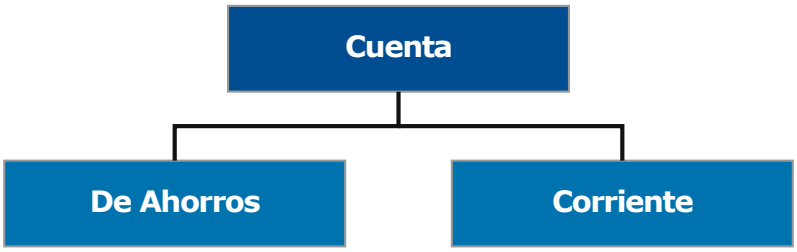
La verdadera programación orientada a objetos requiere que los objetos admitan tres cualidades:

3.3.1 Herencia:

- Describe la capacidad de crear clases nuevas a partir de una clase existente.
- Cuando la clase hereda de UNA sola clase se llama "Herencia Simple".
- La nueva clase hereda todos los atributos y métodos de la clase base, siempre y cuando éstos se hayan declarados como heredables.
- La nueva clase también se puede personalizar con atributos y métodos adicionales.
- Para indicar que una clase deriva de otra, se usa el término extends.
- En nuestro día a día, la herencia se puede relacionar con las jerarquías.



En nuestra vida diaria, las finanzas las manejamos mediante diferentes tipos de cuentas, que pueden ser Corriente o de Ahorro, en este ejemplo básico representemos la herencia.



Pensemos en los movimientos que se hacen con una cuenta: Consignar, Retirar, Consultar Saldo.

El código para implementar esta clase sería:

```
1   package Modelo;
2
3   public class ClsCuenta {
4    double saldo;
5
6    public void consignar(double monto) {
7     saldo += monto;
8    }
9
10   public double conSaldo() {
11    return saldo;
12   }
13
14   public void retirar(double monto){
15    saldo=-monto;
16   }
17  }
```

Explicación Líneas de Código	
Línea	**Explicación**
1	Clase ubicada en el paquete denominado Modelo
3	Creación de la Clase
4	Declaración de la variable de tipo double saldo
6 y 14	Definición de los métodos consignar y retirar, los cuales reciben como argumento la variable monto. Recordemos que los métodos pueden ser vacíos o con argumentos, en este ejemplo se recibe un argumento y son de tipo void, que no retornan ningún valor.
10	Definición del método conSaldo (consultar el saldo) que retorna la variable saldo.

Que es lo que hace diferente a una cuenta de ahorros y una cuenta corriente?? Que a la cuenta corriente le pueden asignar un sobregiro y también una chequera.

En eso orden de ideas, la clase cuentaCorriente, tendría acceso a todos los métodos de la Clase cuenta (consignar, conSaldo, retirar) y la variable saldo que está en la misma clase. Lo que se debería hacer crear un método sobregiro y uno chequera.

El método sobregiro necesitaría un argumento para el valor del Sobregiro y el método chequera dos argumentos, uno para el numero inicial del cheque y otro para el número final del cheque. Y ese es precisamente el objetivo de la herencia.

El código para implementar la clase cuentaCorriente, como clase hija de cuenta sería:

```
1  package Modelo;
2
3  public class ClsCuentaCorriente extends ClsCuenta{
4
5
6   public double sobregiro(Double sobregiro){
7    return sobregiro;
8   }
9
10  public void chequera (String pricheque, String ultCheque){
11   System.out.print ("Su chequera inicia en el numero "+
12   pricheque +"y termina en el número"+ ultCheque);
13  }
14 }
```

Explicación Líneas de Código	
Línea	**Explicación**
1	Clase ubicada en el paquete denominado Modelo
3	Creación de la Clase, con la palabra extends, se está informando que esta clase va a heredar de la clase padre, en este caso ClsCuenta
4	Declaración de métodos sobregiro el cual recibe argumento sobregiro, para informar el valor de sobregiro asignado a la cuenta.
10	Definición del método chequera, donde recibe el número del primer cheque y el número del último cheque.

3.3.2 Polimorfismo:

- Habilidad de tener diferentes formas.
- Posibilidad de tener varias clases que se pueden usar de forma intercambiable.
- Permite usar elementos con los mismos nombres, sin importar qué tipo de objeto esté en uso en ese momento.
- Un objeto tiene solo una forma.
- Una variable de referencia tiene muchas formas.
- Una variable de referencia puede referirse a objetos con diferentes formas.

Sobrescritura de un método

- Desde la clase hijo, modificar comportamientos heredados de la clase Padre
- Modificar el cuerpo de un método, para realizar una funcionalidad de diferente manera.
- En la clase hija se crea el mismo método con diferentes funcionalidades de la clase padre. Este método debe tener el mismo nombre, el mismo tipo de retorno y la misma cantidad de argumentos y tipo de los mismos.

Para este aspecto vamos a tomar como referencia la clase denominada Gerente que hereda de la clase padre llamada empleado, esta clase empleado tiene los atributos nombre y salario, así como un método llamado visualizaDatos, cuya función es visualizar los datos de la clase padre.

La clase "padre" empleado tiene el siguiente código

```
1  package Modelo;
2
3  public class ClsEmpleado {
4   String nombre;
5   int salario;
6
7
8   public String visualizaDatos (){
9   return "Impresión de los datos de la clase Empleado +"\n"
10  +Nombre" + nombre + "Salario" + salario;
11  }
   }
```

La clase "padre" empleado tiene el siguiente código

```
1  package Modelo;
2
3  public class ClsGerente extends ClsEmpleado{
4
5   String departamento;
6
7   public String visualizaDatos(){
8    return "Impresión de los datos de la clase Gerente +"\n"+ Nombre"          +nombre+"Salario"+salario+
9  "Departamento"+departamento;
10   }
11  }
```

Explicación Líneas de Código	
Línea	**Explicación**
1	Clase ubicada en el paquete denominado Modelo
3	Creación de la clase, con la palabra extends, se está informando que esta clase va a heredar de la clase padre, en este caso ClsEmpleado
7	Creación del método visualizaDatos() Observe que este método se llama EXACTAMENTE igual que el de la clase padre (ClsEmpleado), lo que cambia es que se agregó un campo más para visualizar que esta definido especialmente para la clase hijo (ClsGerente)

Ahora para visualizar la información de cada una de las clases (padre e hijo) y para reforzar la creación de objetos, vamos a crear una clase con método main, recordando que las clases que contienen método main, son aquellas que se pueden ejecutar,en donde vamos a crear dos objetos o variables de tipo referenciado, una que hace referencia a la clase Empleado(padre) y otra que hace referencia a la clase Gerente(hijo).

El código de esta clase es el siguiente:

La clase "padre" empleado tiene el siguiente código

```
1   package Modelo;
2
3   public class MainEmpleado {
4
5   public static void main (String [] args)
6   //Objeto creado para manipular los datos en la clase Padre
7      ClsEmpleado emp = new ClsEmpleado();
8      emp.nombre = "Nombre Empleado";
9      emp.salario = 500000;
10     System.out.println(emp.visualizaDatos());
11  //Objeto creado para manipular los datos en la clase Hijo
12     ClsGerente ger = new ClsGerente();
13     ger.nombre = "Nombre Gerente";
14     ger.salario = 500000000;
15     ger.departamento = "Contabilidad";
16     System.out.println(ger.visualizaDatos());
17     }
18  }
```

Variables creadas en la clase Empleado

Aunque estas variables fueron creadas en la clase Empleado, se pueden utilizar gracias a la Herencia.

Método sobreescrito en el objeto gerente. Se agregó el atributo departamento.

En JAVA existe la palabra reservada super, cuya función es invocar el método de la clase padre que deseamos sobrecargar. Esta forma de codificación se realiza en la clase hija y la codificación para el ejemplo que venimos trabajando quedaría de la siguiente manera:

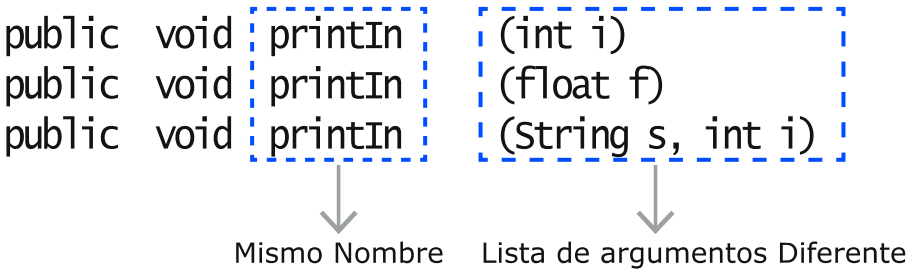
```
1   package Modelo;
2
3   public class ClsEmpleado {
4      String nombre;
5      int salario;
6
7
8      public String visualizaDatos (){
9         return "Impresión de los datos de la clase Empleado + "\n" + Nombre" + nombre + "Salario" + salario;
10     }
11  }
```

```
1   package Modelo;
2
3   public class ClsGerente extends ClsEmpleado{
4
5
6      String departamento;
7
8        public String visualizaDatos () {
9     //Return "Impresión de los datos de la clase Hijo Gerente
10    + "\n" + Nombre" + nombre +
11    //       "Salario" + salario + "Departamento" + departamento;
12
13       return super.visualizaDatos() + "Departamento" + departamento;
14    }
15 }
16 En la CLASE HIJA con el simple hecho de utilizar la palabra super
17 seguida del nombre del método el sistema llama el método con el
18 mismo nombre, que esta ubicado en la clase padre
```

Sobrecarga de un método

Un método sobrecargado se identifica porque se llaman igual, pero la lista de argumentos es diferente, bien sea por su cantidad o por el tipo de datos que recibe.

Un ejemplo de sobrecarga sería:



3.3.3 Encapsulación:

- Oculta los campos implementados en la clase, esto implica el tratamiento de los campos como una única unidad.
- Para que el usuario acceda a los datos, obligatoriamente debe usar una interface
- El mantenimiento del código se hace más fácil.

4. EXCEPCIONES

Las excepciones son otra forma más avanzada de controlar el flujo de un programa, se utilizan para asegurar el correcto funcionamiento del programa y en el caso de un posible error se llamaría la excepción.

Estas son representadas por la clase Exception.

Los problemas graves, que normalmente se tratan como "fatal error", son errores que se han dejado de tratar con excepciones.

El ejemplo mas común de errores generados por falta de excepciones es el "RuntimeException".

La API de JAVA presenta excepciones ya predefinidas que pueden ser usadas en los diferentes métodos.

Estas excepciones se pueden asimilar con un condicional, solo que las palabras utilizadas para este proceso son try, catch

Una excepción puede ocurrir cuando:

- El archivo que intenta abrir no existe
- La conexión de red se ha interrumpido
- Los operadores que se están utilizando están fuera del rango.
- No se encuentra la clase que es está intentando cargar

Analicemos el siguiente código:

```
1   package Modelo;
2
3   public class ClsException {
4
5   public static void main(String[] args) {
6    int i = 0;
7   String mensaje[] = {
8    "Hello world!",
9    "No, I mean it!",
10   "HELLO WORLD!!"
11   };
12
13   while (i < 4) {
14   System.out.println(mensaje[i]);
15   i++;
16   }
17   }
18   }
19
```

Al ejecutar el código aparecerá lo siguiente:

```
run:
Hello world!
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
No, I mean it!
HELLO WORLD!!
        at Modelo.ClsException.main(ClsException.java:14)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Este error se debe a que no se implemento una excepción. El código ideal seria el siguiente con manejo de excepciones (try, catch y finally)

```
1  package Modelo;
2
3  public class ClsException {
4
5      public static void main(String[] args) {
6          int i = 0;
7          String mensaje[] = {
8              "Hello world!",
9              "No, I mean it!",
10             "HELLO WORLD!!"
11         };
12         while (i < 4) {
13             try {
14                 System.out.println(mensaje[i]);
15             } catch (ArrayIndexOutOfBoundsException e) {
16                 System.out.println("Re-stting Index Value");
17                 i = -1;
18             } finally {
19                 System.out.println("This is always printed");
20             }
21             i++;
22         }
23     }
24 }
```

Manejo de la Excepción

GLOSARIO

API:	Application Programming Interface
Clase:	Is a template or model.
Encapsulación:	Oculta los campos implementados en una clase.
Excepciones:	forma más avanzada de controlar el flujo de un programa.
Final Static:	Modificadores que se utilizan para crear variables que no modifican su valor durante la ejecución del programa.
Herencia:	Describe la capacidad de crear clases nuevas a partir de una clase existente.
Métodos:	Representan las acciones que puede realizar un objeto.
New:	Palabra reservada cuya objetivo es crear la copia de una clase.
Operadores:	Se utilizan para realizar operaciones entre objetos, datos, identificadores y/o constantes, y devuelven un valor.
Package:	Palabra reservada que contiene los enlaces con la toda la información de las clases.
Palabras Reservadas:	Identificadores reservados por Java para un objetivo específico.
Poliformismo:	Habilidad de tener diferentes formas.
Super:	Palabra reservada cuya función es invocar el método de la clase padre que se desea sobrecargar.

GLOSARIO

This:	Este valor se utiliza dentro de cualquier método, para referirse al objeto actual.
Variables de Stack:	Son aquellas que se definen dentro de un método y son llamadas variables locales.

OBJETO DE APRENDIZAJE	Introducción al Lenguaje de Programación Java
Desarrollador de contenido Experto temático	**Magda Milena García Gamboa**
Asesor Pedagógico	**Claudia Marcela Hernández** **Rafael Neftalí Lizcano Reyes**
Productor Multimedia	**Adriana Marcela Suárez Eljure** **Victor Hugo Tabares Carreño**
Programadores	**Daniel Eduardo Martínez Díaz**
Líder expertos temáticos	**Ana Yaqueline Chavarro Parra**
Líder línea de producción	**Santiago Lozada Garcés**

RECURSOS BIBLIOGRÁFICOS

Microsoft. (2013). Programación Orientada a Objetos. Recuperado el 9 de Julio de 2013, de http://msdn.microsoft.com/es-ES/vstudio/ms789107.aspx

Sun Educational Services. (1999). Java Programming Language SL-275. Recuperado el 9 de Julio de 2013, de https://docs.google.com/folder/d/0B8UHI_K6Lw_cX3RwQmdGRWZkelU/edit?usp=drive_web&pli=1

Universidad de Salamanca. (Octubre de 1999). Guía de Iniciación al Lenguaje JAVA. Recuperado el 9 de Julio de 2013, de http://zarza.usal.es/~fgarcia/doc/tuto2/Index.htm