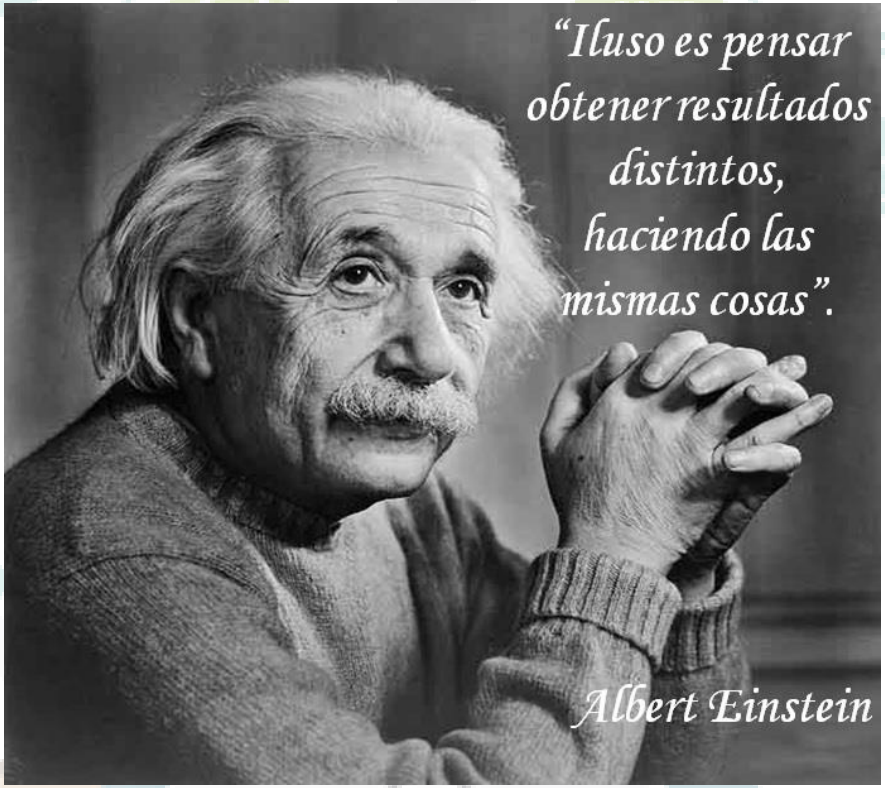
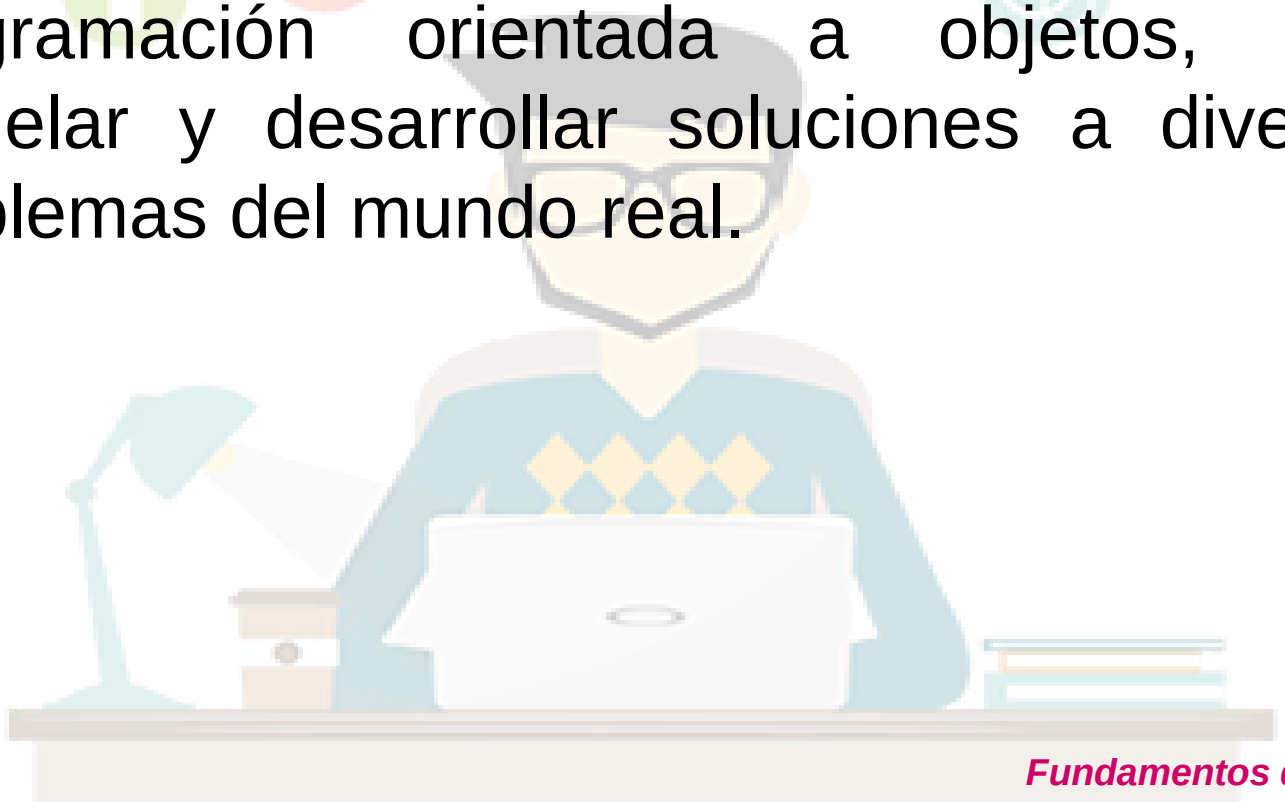
REAFIRMACIÓN DE COMPETENCIAS BÁSICAS DE FUNDAMENTOS DE PROGRAMACION ISIC-AED-1285



Competencia específica

Aplica las herramientas básicas de programación orientada a objetos, para modelar y desarrollar soluciones a diversos problemas del mundo real.



Lenguaje de Programación, Programa y Programación

Definición de programa

Es un conjunto de instrucciones internas, utilizadas para ejecutarse en una computadora y producir un resultado concreto.

Definición de programación

Es el proceso de escribir programas para computadora en algún lenguaje de alto nivel o lenguaje ensamblador (bajo nivel).

Lenguaje de Programación

Definición

Es un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Es utilizado para controlar el comportamiento físico y lógico de una máquina.

SOFTWARE

Concepto

- Es el conjunto de los programas de cómputo, procedimientos, reglas, documentación y datos asociados que forman parte de las operaciones de un sistema de computación.

Clasificación

- Software de sistema
- Software de aplicación

Software de sistema (software de base)

- Son programas que sirven a otros programas y ejecutan funciones comunes para todos los usuarios de la computadora.
- Implementa funciones de control que permiten al software de aplicación comunicarse con otros elementos del software.
- Son los **sistemas operativos**, **compiladores**, **ensambladores**, **cargadores**, **manejadores de bases de datos**, …

Software de aplicación

- **Es el software específico para aplicaciones particulares de los usuarios de un sistema de cómputo**.
- Para su codificación no se utiliza un lenguaje de bajo nivel, sino que se codifican en lenguajes de alto nivel.
- Éste software se apoya en el software de sistemas.

La POO (Programación Orientada a Objetos) y sus elementos primordiales.

Programación Orientada a Objetos (POO)

Es una forma o estilo de programar (paradigma de programación), más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación.

Objetos

- Los objetos, concretos y abstractos, están a nuestro alrededor, forman nuestro entorno. Podemos distinguir cada objeto en base a sus características y comportamientos.
- ¿Qué objetos observas en el aula?

  alumno, profesor, mesa, silla, butaca, pizarrón

Clase

- Descripción de un conjunto de objetos con características, atributos, y comportamientos similares.

- Automóvil

Características o Atributos:
Color, tamaño, modelo, marca,…

Comportamientos:
Encendido, apagado, en marcha, detenido,…

Relaciones entre clases y objetos

- Comportamiento de los objetos específico de su clase.
  - **alumno: estudia, aprende.**
  - **profesor: enseña, evalúa.**
  - **mesa: ordenada, desordenada.**
  - **silla: ocupada, desocupada.**
  - **butaca: ocupada, desocupada.**
  - **pizarrón: pintado, borrado.**
- Clase Alumno
- Clase Profesor

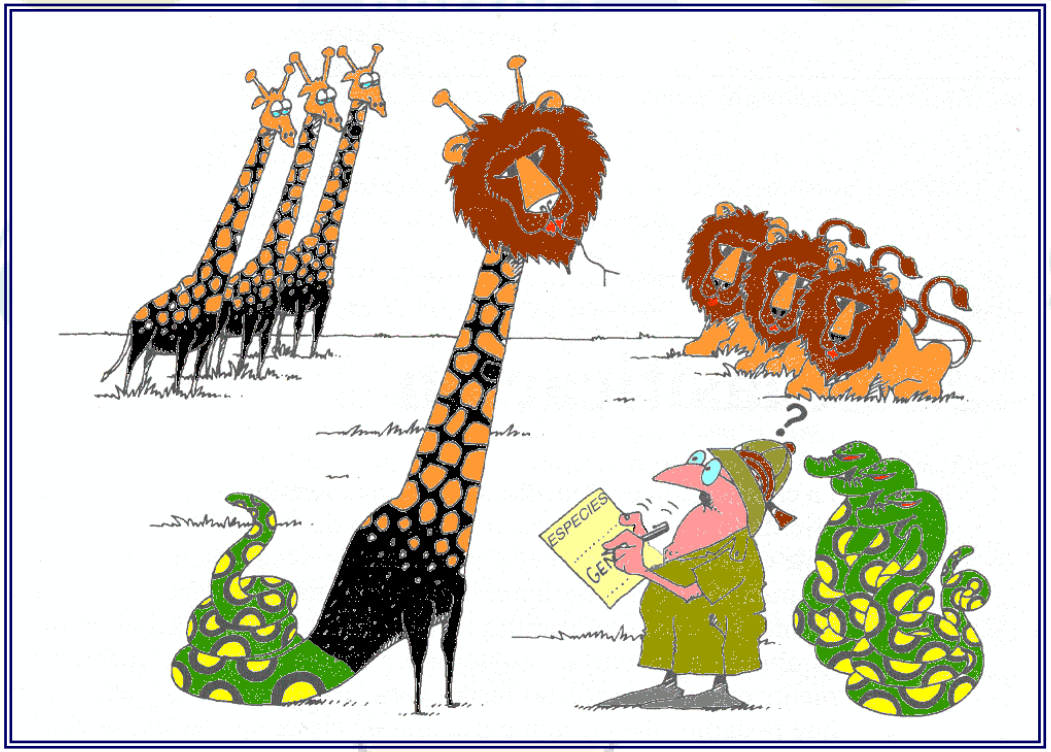
Instancia

- Objeto individual que esta descrito por un conjunto de características y comportamientos y es un miembro de una clase particular.

Instancia = Objeto

- Clase Alumno
  - Instancia: María López Gómez

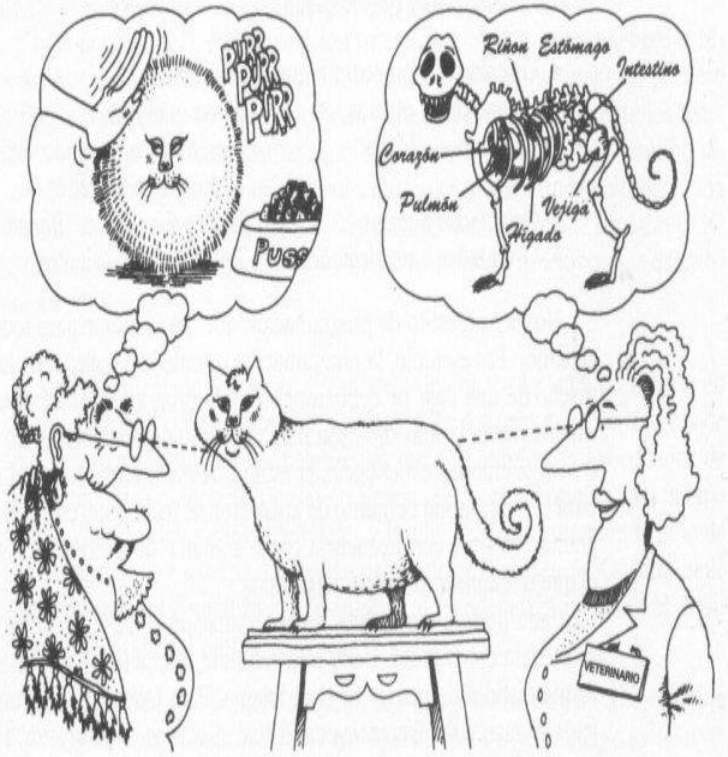
- Clase Profesor
  - Instancia: Jesús Coronado Hernández



¿Qué Objetos y Clases identificas?

Abstracción

- Es una descripción simplificada o especificación que enfatiza algunos de los detalles o propiedades de algo.
- Es un proceso mental, mediante el cual se extraen los razgos esenciales de algo para representarlos por medio de un lenguaje gráfico o escrito. Ya que es un proceso mental, la abstracción es subjetiva y creativa, es decir, depende del contexto psicológico de la persona que realiza.



La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

La abstracción como un proceso natural

En el modelo de objetos, las características esenciales de los objetos estarán determinadas por el tipo de problema que se vaya a resolver con ellos, es decir, se crean modelos diferentes del objeto para resolver cada problema.

Sistema de tránsito

Origen (nacional, extranjero)
Marca
Modelo
Color
Tipo de vehículo
Tipo de Combustible
Uso (particular, público)
Número de pasajeros
Nombre del propietario
Dirección del propietario

Vehículo

Sistema para Taller

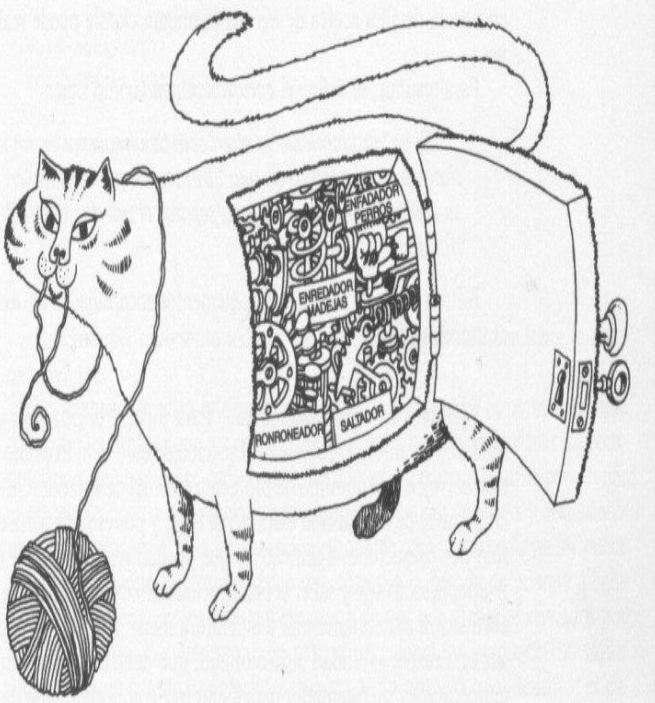
Marca
Modelo
Color
Falla detectada
Nombre del propietario
Dirección del propietario
Teléfono del propietario

Encapsulamiento

- Consiste en considerar no sólo las características o propiedades de un objeto sino también su comportamiento o funcionalidad.
- Es el proceso de almacenar en un mismo compartimento los elementos de una abstracción que constituyen su estructura y su comportamiento. Se consigue, a menudo, mediante la **ocultamiento** de información.

El encapsulamiento como proceso natural

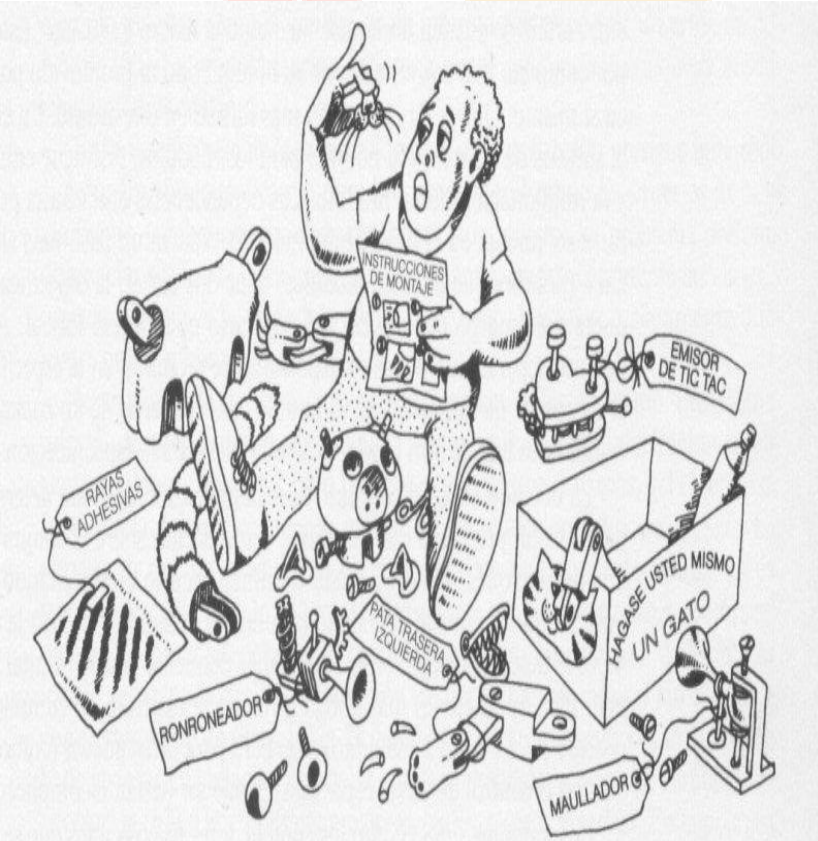
- Los nombres de grupos de objetos del mundo real son como un código creado por los humanos para representarlos a través de parejas nombre: definición, facilitando así la comunicación.



El encapsulamiento oculta los detalles de la implementación de un objeto.

Modularidad

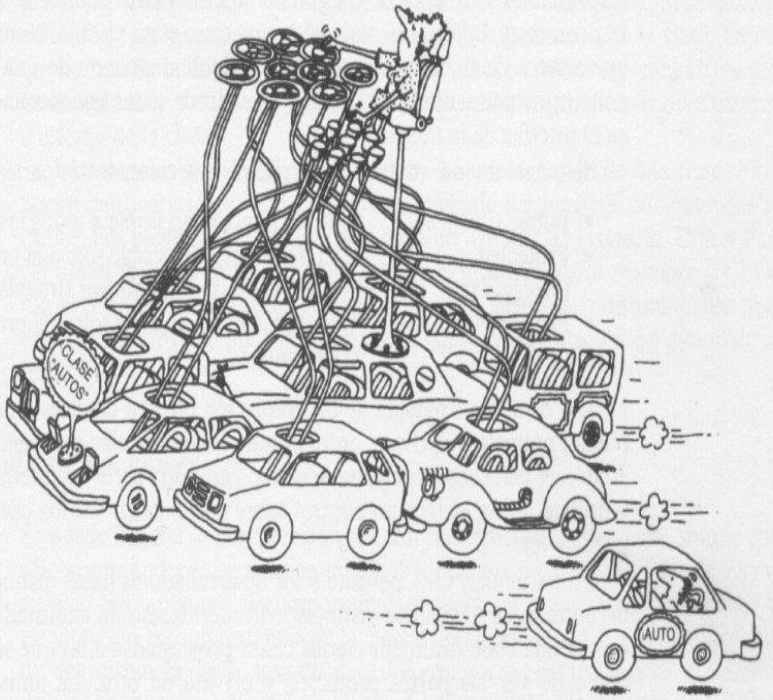
- Es la descomposición de un sistema en un conjunto de módulos cohesivos y débilmente acoplados.
- La estructura de cada módulo debe ser sencilla y entendible para permitir cambiar su implementación sin conocer detalles de los demás módulos y sin afectar su comportamiento.



La modularidad empaqueta las abstracciones en unidades discretas.

Jerarquía

- Es una **clasificación** u ordenación de **abstracciones.**
- La estructura de **clases** se representa por medio de una jerarquía que muestra la **relación de herencia es un.**



Una clase representa un conjunto de objetos que comparten una estructura común y un comportamiento común.

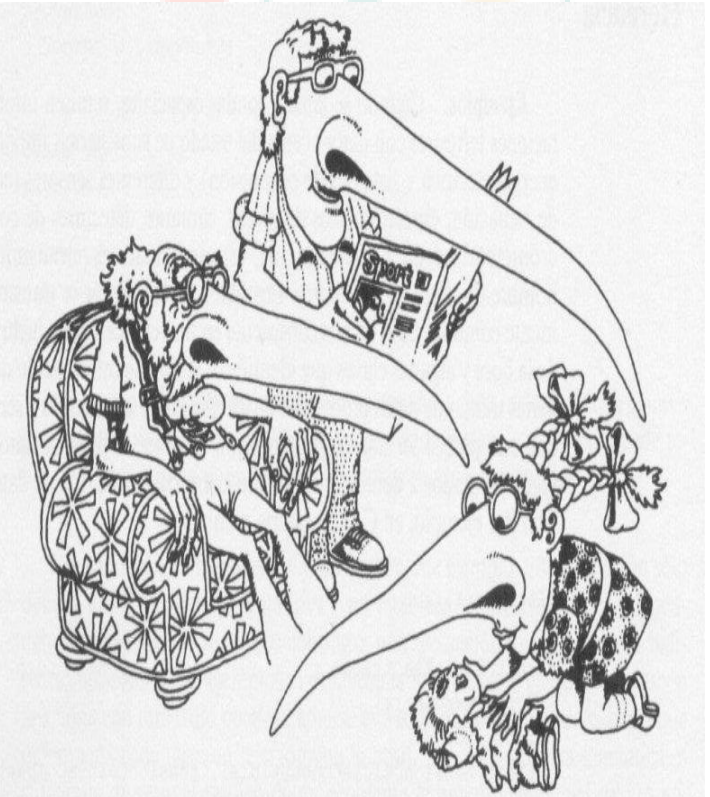
Herencia

• Define la relación entre clases **es un**, donde una **subclase** hereda de una o más **superclases.**

• **Generalización/especialización:** Jerarquía en la que una subclase especializa el comportamiento y/o la estructura, más general, de sus superclases.

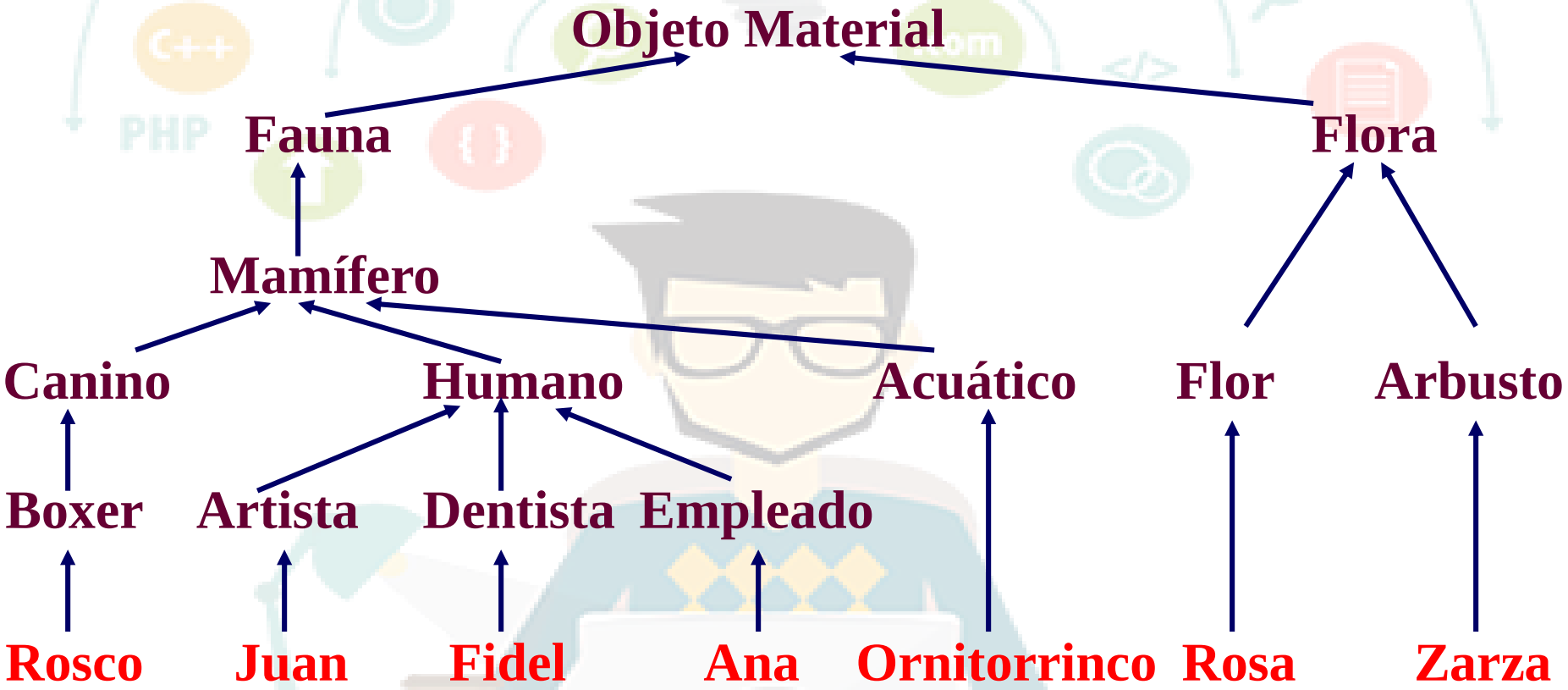
• **Herencia simple:** Se da cuando, en una jerarquía de clases, las **subclases** solamente pueden heredar de **una superclase.**

• **Herencia múltiple:** Las **subclases** pueden heredar de **más de una superclase**.



Una subclase puede heredar la estructura y comportamiento de su superclase.

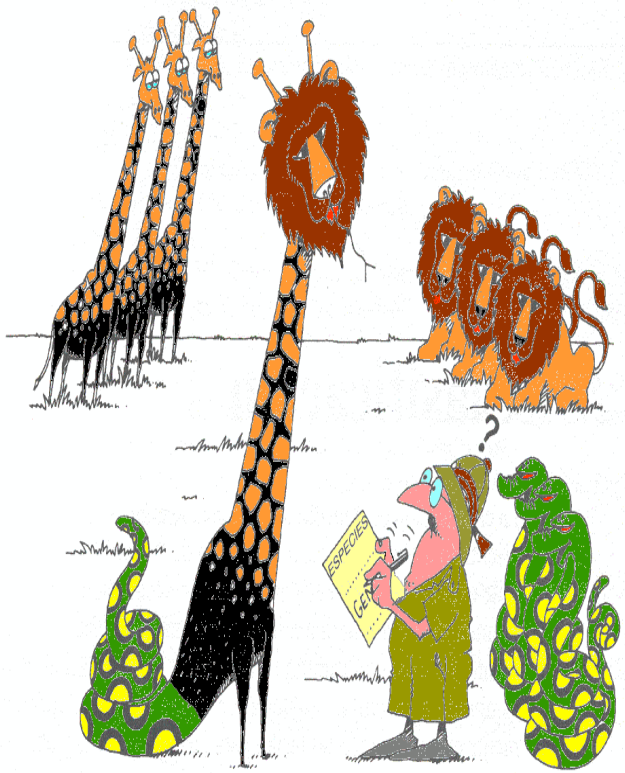
Jerarquía y Herencia…



Polimorfismo

•Del griego poli (**muchos**) y morfos (**formas**) y se utiliza para indicar que un **nombre** puede denotar **instancias** (objetos) de **clases diferentes** que están relacionadas por alguna **superclase común**.

• Se considera la característica más potente de los lenguajes O.O., después de su capacidad para soportar la **abstracción**.



Proceso de construcción de programas

De los problemas a los programas

Análisis de la problemática

Diseño de la solución

Prueba de escritorio

Implementación

Ejecución y validación

- PLANEACIÓN
- DOCUMENTACIÓN
- PARADIGMA DE DESARROLLO DE SW

Proceso de construcción de programas…

- **Análisis de la problemática:** Conocer con precisión el problema a resolver y su contexto, con la finalidad de proporcionar una solución efectiva.
- **Diseño de la solución:** Considerando el análisis del problema, se estructura la solución correspondiente.
- **Prueba de escritorio:** Se proporcionan valores de entrada a la solución diseñada para determinar si los valores de salida son los esperados.
- **Implementación:** Es la codificación de la solución diseñada y validada.
- **Ejecución y validación:** se ejecuta el código elaborado y se valida el resultado generado.

Herramientas

Algoritmo

Concepto

- Es un método sistemático para resolver un problema.
- Conjunto finito de instrucciones o pasos que sirven para ejecutar una tarea o resolver un problema.
- Secuencia finita de operaciones realizables, no ambiguas, cuya ejecución da una solución de un problema.

Origen: La palabra proviene de Mohammed al-KhoWârizmi (en latín ***algorismus)***, matemático del siglo IX que enunció las reglas paso a paso para sumar, restar, multiplicar y dividir números decimales.

Herramientas

Algoritmo

Características

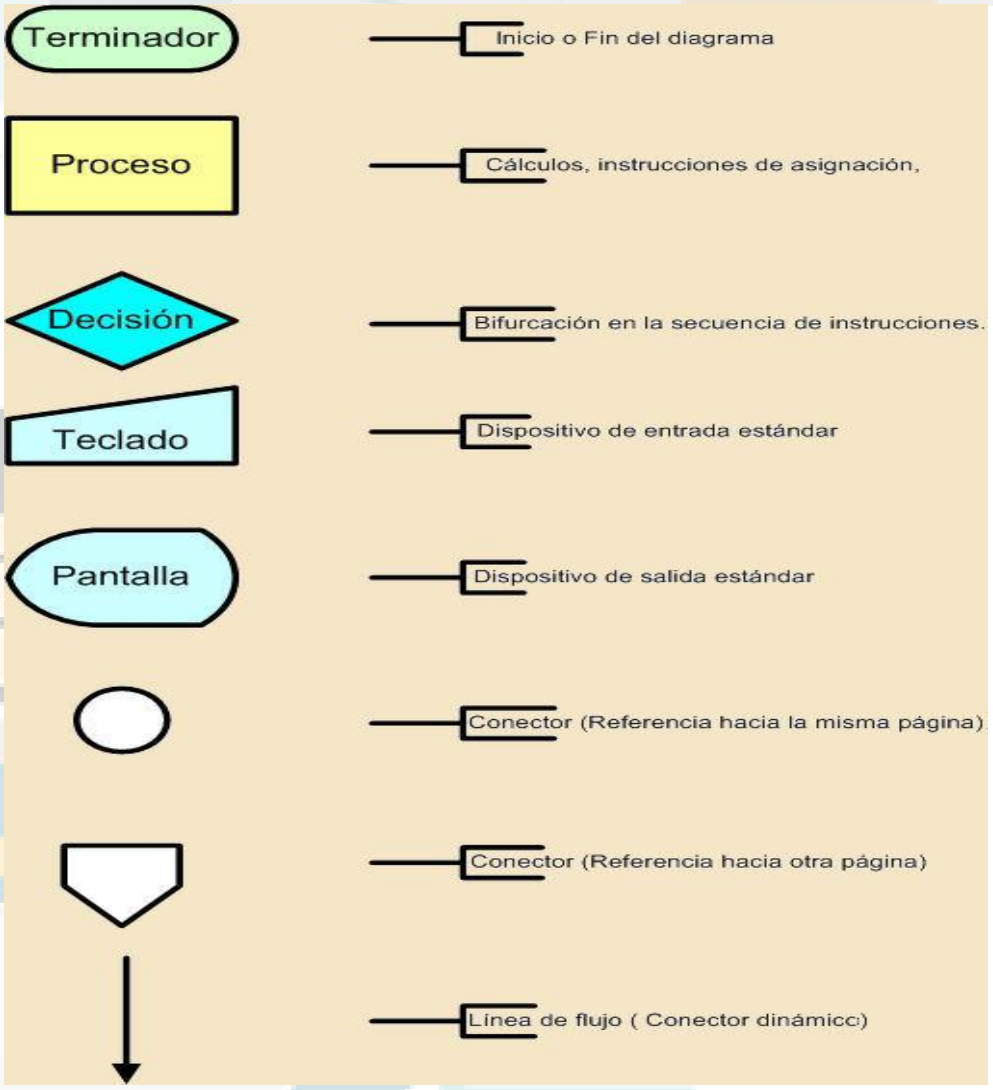
1. **Preciso** e indicar el orden de realización de cada paso.
2. **Definido**. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
3. **Finito**. Si se sigue un algoritmo, se debe terminar en algún momento; o sea, debe tener un número finito de pasos.

Diseño del algoritmo: Convierte los resultados del análisis del problema en un diseño modular con refinamientos sucesivos que permitan una posterior traducción a un lenguaje y es independiente del mismo.

Herramientas

Diagramas de flujo
(flowchart)

Es una representación gráfica de un algoritmo. Los símbolos utilizados para la elaboración de diagramas de flujo han sido normalizados por el ANSI.



Herramientas

Pseudocódigo

- Es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o al español, que facilita tanto la escritura como la lectura de programas.

- En esencia, es un lenguaje para la especificación de algoritmos.

Pseudocódigo

Ejemplo:

```
if  x < y entonces
        z <- 1000
  De lo contrario
        q <- 500
End_if
```

De algoritmos aplicados a problemas

Procedimiento

Considerando el modelo de vida clásico para el desarrollo de programas y partiendo de la descripción de la problemática a resolver:

<u>Análisis</u>:

¿Qué resuelvo?

Expresar son claridad y precisión lo que se ha comprendido de la problemática a resolver.

¿Qué necesito?

Describir claramente lo que se requiere para resolver la problemática expuesta.

<u>Diseño</u>:

¿Cómo lo resuelvo?

Se determinan las acciones necesarias para resolver el problema, expresando dicha solución mediante la aplicación de alguna herramienta de diseño.

Estructura básica de un programa

Importación de paquetes

Nombre de la clase

Área de declaraciones globales

Área de declaraciones locales al método

Declaración de métodos

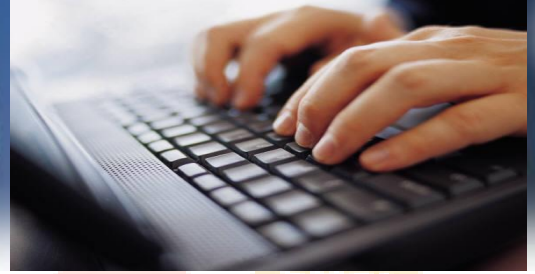
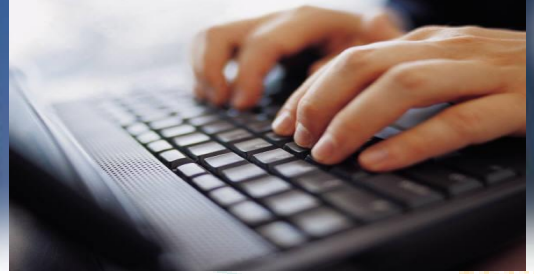
Cuerpo de la clase

Área de declaraciones locales al método

Área de declaraciones locales al método

Método principal de la clase

```
import java.io.*;
class arreglo{

    public int[] x = new int[5];

    public void leer(int n)
    {
      int i;

      BufferedReader dato = new BufferedReader(new InputStreamReader(System.in));

      System.out.println("\nTECLEE LOS DATOS DEL ARREGLO\n");

      for(i=0;i<n;++i){
        System.out.print("X["+ i+"]= ");
        try{
            this.x[i]= Integer.parseInt(dato.readLine());
           }
        catch(java.io.IOException ioex)
        {
        }

      }
    }

    public void escribe(int n){
        int i;
        System.out.println("\nLOS DATOS DEL ARREGLO SON:\n");
        for(i=0;i<n;++i)
          System.out.print("\nX["+ i+"]= "+ this.x[i]);
        System.out.print("\n\n***FIN DEL PROGRAMA***\n\n");
    }

public static void main(String[] args)throws IOException
{int n;
    BufferedReader dato2 = new BufferedReader(new InputStreamReader(System.in));
    System.out.println("\nCuantos datos son?\n");
    n= Integer.parseInt(dato2.readLine());
    arreglo miArreglo = new arreglo();
    miArreglo.leer(n);
    miArreglo.escribe(n);


}
}
```

ENTRADA Y SALIDA DE DATOS

- Los programas necesitan comunicarse con su entorno, tanto para recoger datos e información que deben procesar (*lectura o entrada*), como para devolver los resultados obtenidos (*escritura o salida*).

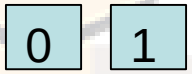
Elementos del lenguaje de programación.

Tipos de datos

- Un **tipo de datos** es una categoría que está determinada por un conjunto de datos con características comunes entre sí.
- Cada tipo de dato se tiene el espacio de almacenamiento que ocupa cada dato:

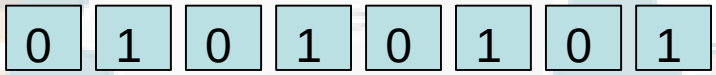
Bit

- Síntesis de Binary digit (dígito binario o con dos posibles valores). Es la unidad de información más sencilla posible en el sistema binario.



Byte

- Unidad de información que consta de 8 bits equivalente a un único caractér, como una letra, número o signo de puntuación.



Clasificación General:

- *Simples*: Éstos ocupan una casilla de memoria, por lo tanto una variable simple hace referencia a un único valor a la vez; en éste grupo de datos se encuentran: enteros, reales, caracteres, booleanos, etc.
- *Estructurados*: Se caracterizan por el hecho de que con el identificador de una variable estructurada se hace referencia a un grupo de casillas de memoria.

Tipos de datos en Java:

a) **Tipos de datos enteros:** Se usan para representar números enteros con signo.

TIPO	TAMAÑO
byte	1 Byte (8 bits)
short	2 Bytes (16 bits)
int	4 Bytes (32 bits)
long	8 Bytes (64 bits)

b) **Tipos de datos en coma flotante:** Son números con partes fraccionarias.

TIPO	TAMAÑO
float	4 Bytes (32 bits)
double	8 Bytes (64 bits)

c) Tipos de datos boolean: Almacena variables bi-estado, representados por los valores true y false.

d) Tipo de datos carácter (char): Almacena caracteres Unicode simples (valores de 16 bits).

‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘A’, ‘B’, ‘C’….’Y’, ‘Z’, ‘¡’, ‘-’, ‘+’, ‘*’,…

- **Cadena o String**: Secuencia finita de símbolos tomados de un conjunto de caracteres llamado alfabeto.

‘‘Hola Mundo’’

IDENTIFICADORES

- Es un nombre que se utiliza para etiquetar variables, constantes, métodos, clases, objetos, etc.

Reglas:

1. No debe contener caracteres especiales solo el carácter subrayado _
2. El primer carácter debe ser letra o _
3. Se hace distinción entre las letras mayúsculas y las minúsculas
4. Puede contener cualquier cantidad de caracteres pero solo los primeros 8 son reconocidos.
5. Debe ser claro, breve y referente a lo que representa.
6. Los nombres de las clases inician con mayúscula y los objetos con minúscula.

Identificadores…

Palabras reservadas (palabras clave)

- Son identificadores predefinidos por el lenguaje y reservados para un uso específico.

abstract	boolean	break	byte
case	continue	catch	char
else	extends	default	do
float	for	false	final
if	implements	import	native
int	interface	long	package
null	public	switch	return
protected	super	true	synchronized
static	trasient	while	try
throw	volatile	doble	instanceof
void	class	finally	new
private	short	this	

VARIABLES

- Es un **contenedor de datos** que tiene asociado un **tipo**, un **identificador** y un **valor** que **puede cambiar en tiempo de ejecución**.
- El **tipo** determina el **tamaño** de memoria que se asigna a la variable, el **identificador** es una **etiqueta** equivalente a la dirección de memoria del primer byte, y el **valor** es el **contenido** de la variable.

CONSTANTES

- Es un **valor** que **NO** puede cambiar en tiempo de ejecución. En java no se declaran constantes, se usan de manera directa en las expresiones e instrucciones.

OPERADORES

Son elementos del lenguaje que sirven para reducir expresiones y obtener resultados. Dependiendo de su tipo, se aplican sobre uno (unario), dos (binario) o tres (ternario) operandos.

Binarios

- Aritméticos:

Operador	Operación
+	Suma
-	Resta
*	Multiplicación
/	División
%	Residuo

Binarios…

- Relacionales:

Operador	Operación
= =	Igual que
!=	Diferente (no igual) que
<	Menor que
<=	menor o igual que
>	Mayor que
>=	Mayor o igual que

?

- Lógicos:

Operador	Operación
&&	Conjunción (Y)
||	Disyunción (O)
!	Negación (NO)

Ternarios

- Sintaxis

  expresión booleana **?** instrucción1 **:** instrucción2

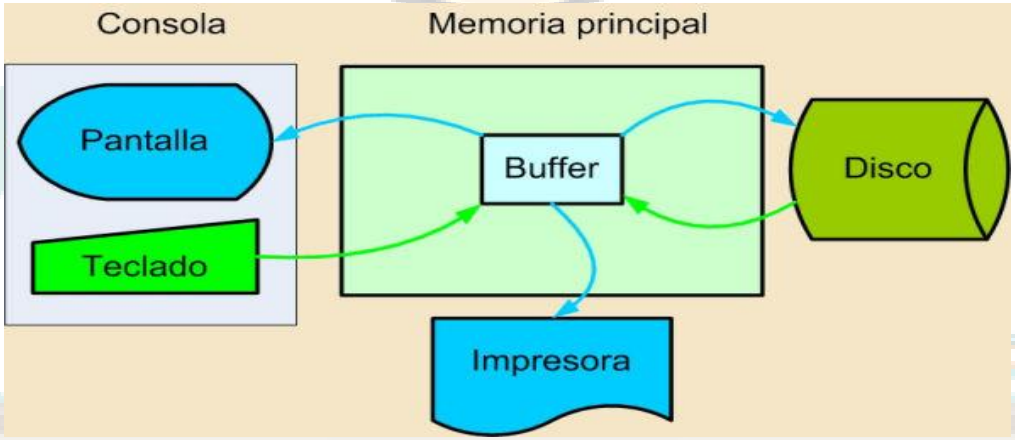
- Ejemplo:

  x > y **?** y = 10 **:** y = 0 ;

Unarios

Operador	Operación
++	Incremento
- -	Decremento

- **Entrada de datos:** consiste en colocar en la memoria principal datos provenientes desde algún dispositivo de entrada (teclado, disco, …) para que la computadora, de acuerdo a un programa, realice una tarea.
- **Salida de datos:** consiste en enviar datos (resultado de un procesamiento) desde la memoria principal hacia un dispositivo de salida (pantalla, impresora, disco, …).



- La representación de estas entradas y salidas en ***Java*** es mediante ***streams*** (***flujos de datos***).
- Un ***stream*** es una conexión entre el programa y la fuente o destino de los datos, por donde la información se traslada en serie (un caracter tras otro).
- La entrada/salida de datos puede realizarse de 2 maneras:
  - Con ***bytes*** *(1 sólo dato)*
  - Con ***caracteres*** *(un conjunto datos)*

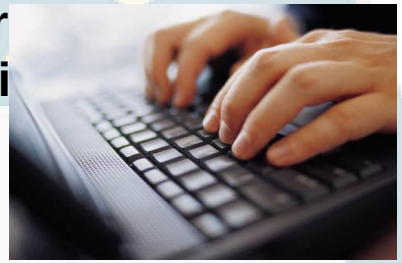
Entrada (lectura)

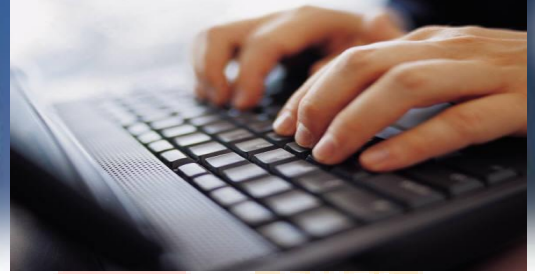
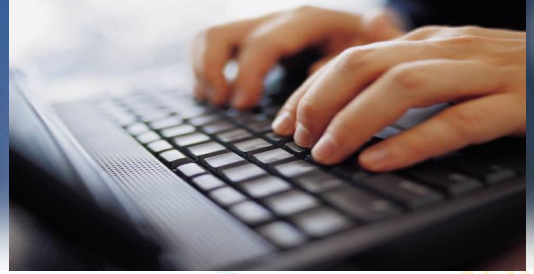
- **System.in** : Es un objeto de la clase **InputStream**, de tal forma que para leer tenemos que emplear sus métodos, el más básico es **read**, que permite leer un carácter:

char caracter = (char) System.in.read();

- Para leer una línea completa, es decir, más de un caractrer se utiliza el siguiente código:

```
BufferedReader br = new Buffer
    InputStreamReader(System.i
String linea = br.readLine();
```



Entrada (lectura)…

Donde:

InputStreamReader: Es una clase puente que permite convertir **streams** que utilizan bytes en otros que manejan caracteres.

System.in : Objeto de la clase ***InputStream*** preparado para recibir datos desde la entrada estándar del sistema (el teclado).

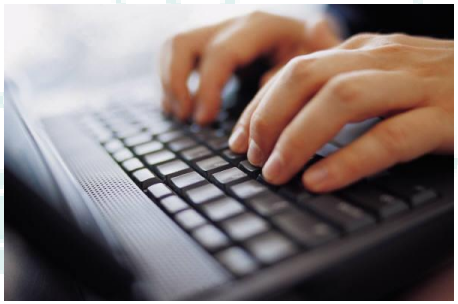
BufferedReader: Clase derivada que permite la creación de un stream para la entrada de datos.

br: Es el nobre del objeto creado de la clase BufferedReader.

readLine(): es un método de la clase BufferedReader que permite leer una linea y devolverla como un String o cadena de caracteres.

- Es necesario realizar un **import java.io.*** para poder emplear esta lectura de líneas.

String línea : Es la declaración de la variable línea en la que se almacenará la cadena de caracteres leída.



Entrada (lectura)…

Ejemplos:

```
int x;
BufferedReader entrada = new
    BufferedReader(new
    InputStreamReader(System.in));
x = Integer.parseInt(entrada.readLine());
```

Salida (escritura)

- En Java, la entrada desde teclado y la salida a pantalla están reguladas a través de la clase ***System.***
- Esta clase pertenece al package ***java.lang*** y agrupa diversos métodos y objetos que tienen relación con el sistema local: System.in (lectura) y System.out (escritura)
- **System.out:** Objeto de la clase ***PrintStream*** que imprimirá los datos en la salida estándar del sistema (normalmente asociado con la pantalla).

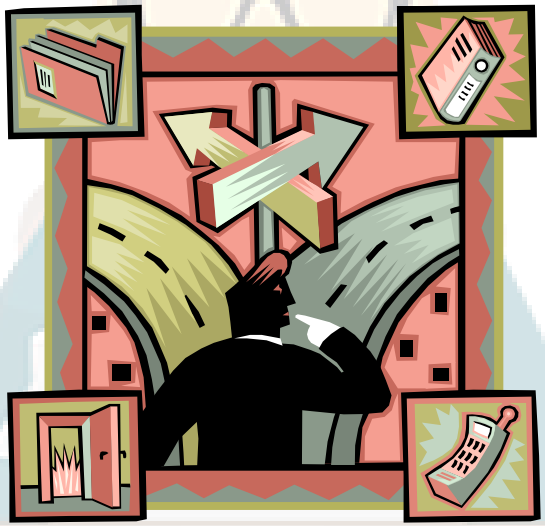
Salida (escritura)…

Ejemplos:

```
System.out.println("Teclea el segundo operando = ");
System.out.println(“El valor de x es“+x);
System.out.println(línea);
```

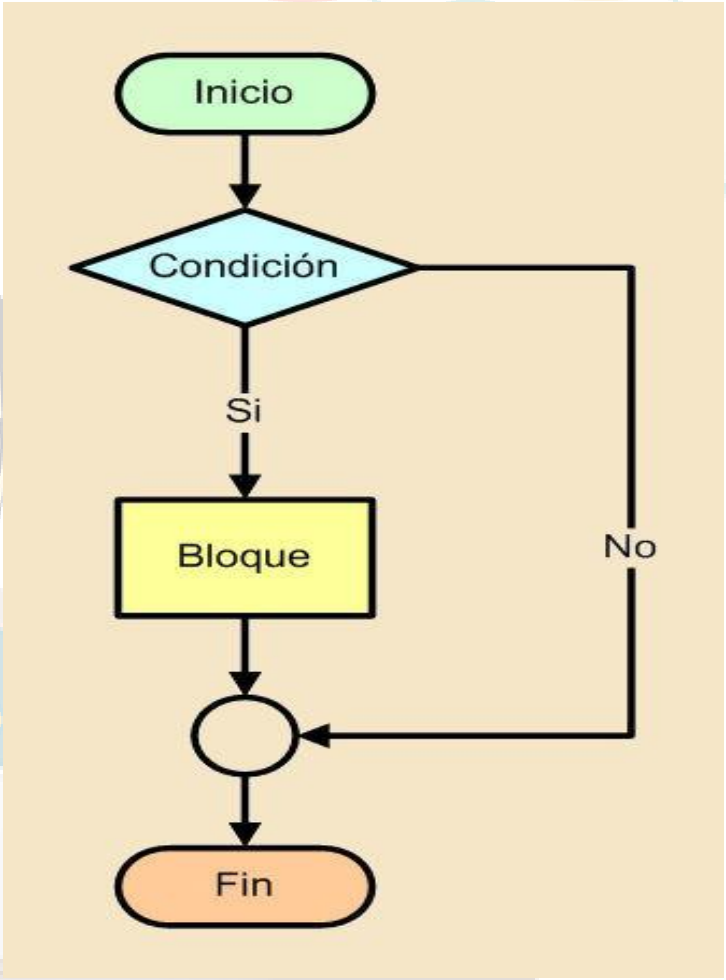
3.2 Estructuras selectivas

- Las estructuras selectivas sirven para seguir una sola de entre varias líneas de ejecución disponibles.



Selectiva simple

- La estructura selectiva simple sirve para seguir una línea de ejecución cuando se cumple una condición.



Selectiva simple…

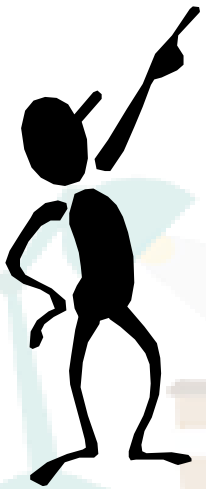
Sintaxis

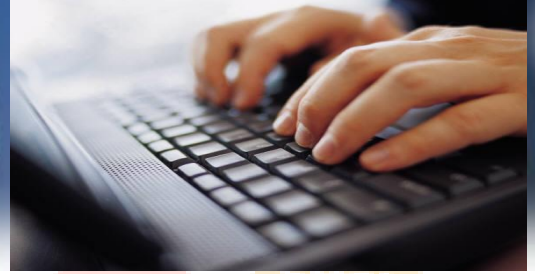
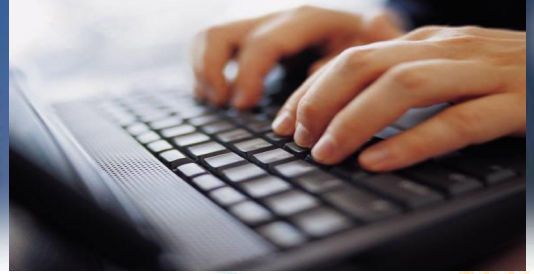
a) Con una instrucción

```
if (condicion(es))
      instrucción;
```

Ejemplos:

```
if (x==y)
      x = y + 2;
if ((a <= b) && (c = d))
System.out.print("ERROR!!");
```



Sintaxis

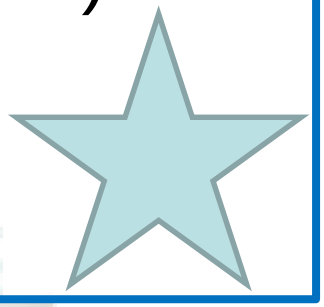
Selectiva simple…

b) Con varias instrucciones

```
if (condición(es))
{ instrucción1;
   instrución2;
      …
   instrucción n;
}
```

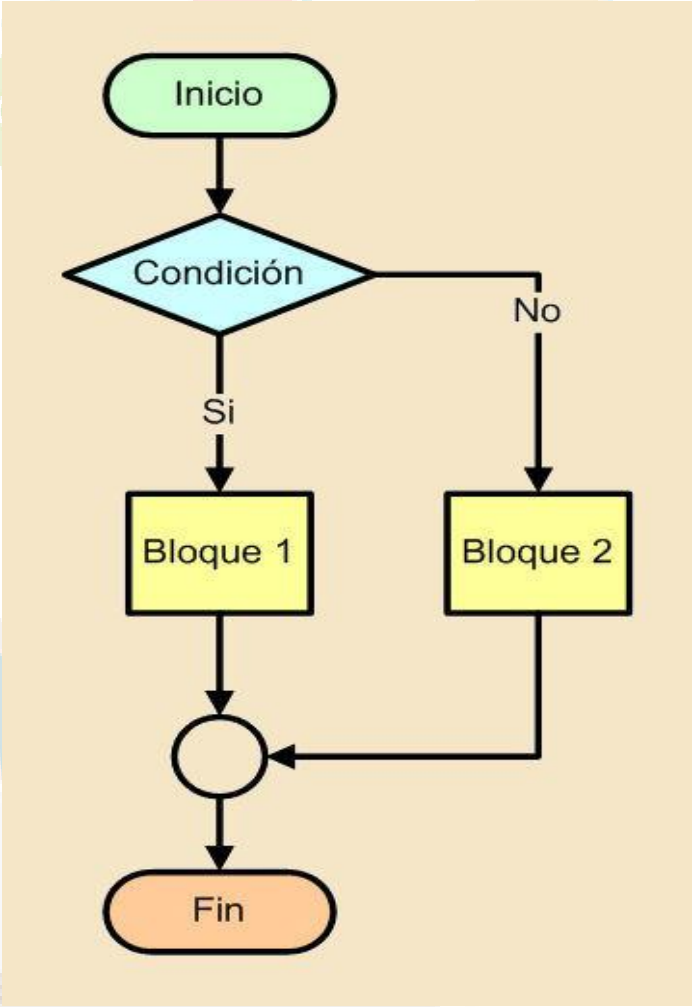
Ejemplos:

```
if (w != 0)
{ System.out.print(“Dato
     válido”);
   x = y + 3;
}
if (valor1 >= valor2)
{ r = z %w;
   t = “A”;
}
```



Selectiva doble

- La estructura selectiva doble sirve para seguir una línea de ejecución cuando se cumple la condición, o seguir otra línea cuando no se cumple la condición.



Selectiva doble…

Sintaxis

a) **Con una instrucción**

```
if (condición(es))
    instrucción;
else
    instrucción;
```

Ejemplos:

```
if (x==y)
    x = y + 2;
else
    y = r / 2;

if ((a <= b) && (c == d))
    System.out.print (“ERROR!!”);
else
    System.out.print (“Datos válidos”);
```

Selectiva doble…

Sintaxis

b) Con varias instrucciones

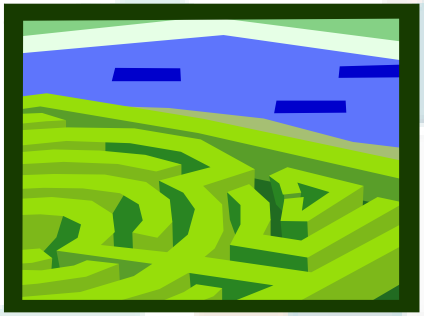
```
if (condición(es))
  { instrucción1;
        …
    instrucción n;
  }
else
  { instrucción1;
        …
    instrucción n;
}
```

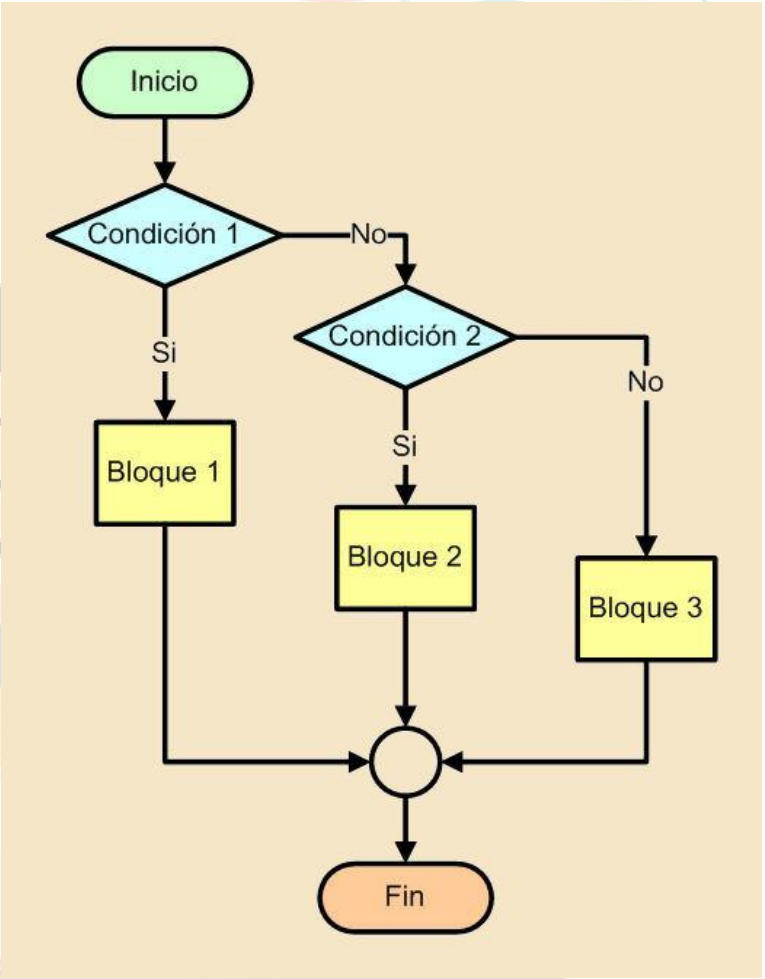
Ejemplo:

```
if ((j!=i)||(i>0))
{ r = z %w;
  System.out.print(“El
     resultado es”+r);
  }
else
{
    t = m*2;
    e = t / 2;
 }
```

Selectiva anidada

- La estructura selectiva anidada es una cadena de estructuras selectivas que se conectan de la parte **else** de la actual con la parte **if** de la siguiente.





Selectiva anidada…

Sintaxis

a) **Con una instrucción**

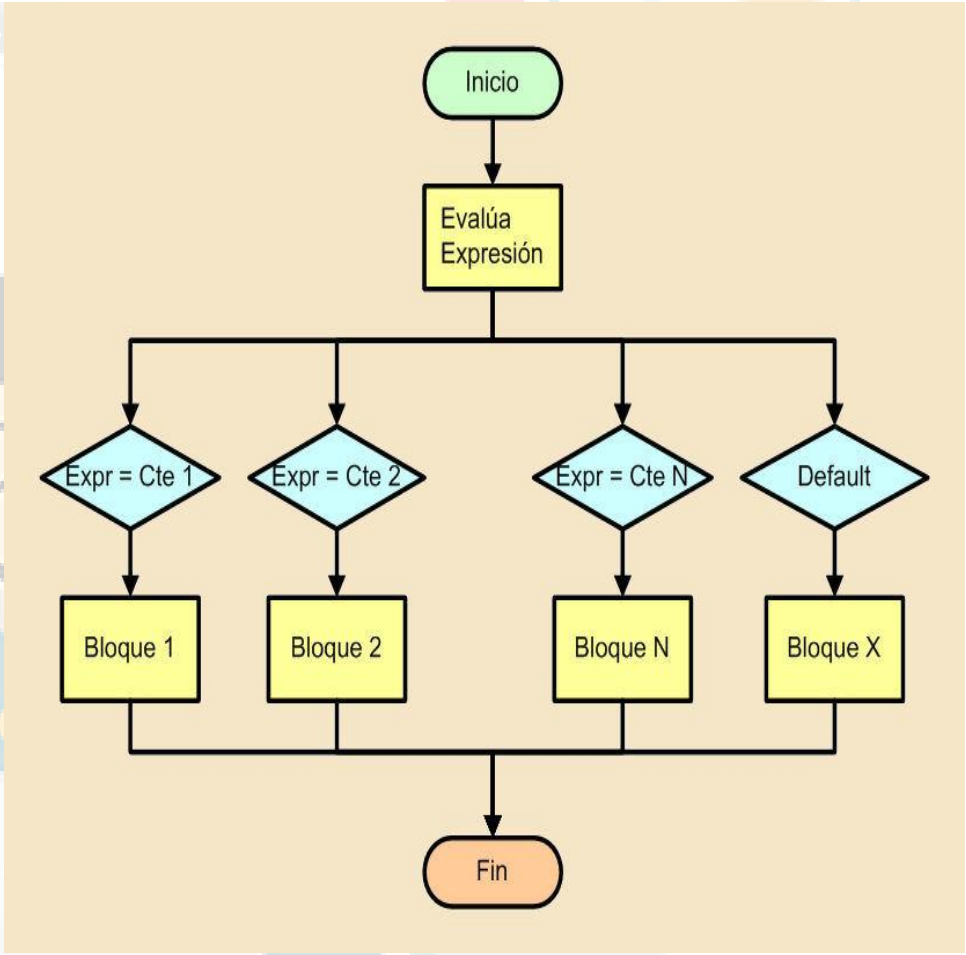
```
if (condición(es))
    if (condición(es))
        Bloque 1
    else
        Bloque 2
else
    Bloque 3
```

```
if (condición(es))
    Bloque 1
else
    if (condición(es))
        Bloque 2
    else
        Bloque 3
```

?

Selectiva múltiple

- La estructura selectiva múltiple es similar a la selectiva anidada, salvo que las condiciones deben ser de alguno de los tipos enteros o de tipo carácter.



Selectiva múltiple…

Sintaxis

```
switch(Expresión){
   case Cte 1: Bloque 1 ; break;
   case Cte 2: Bloque 2 ; break;
   case Cte N: Bloque N ; break;
default: Bloque X ; break;//
      Opcional.
}
```

Selectiva múltiple…

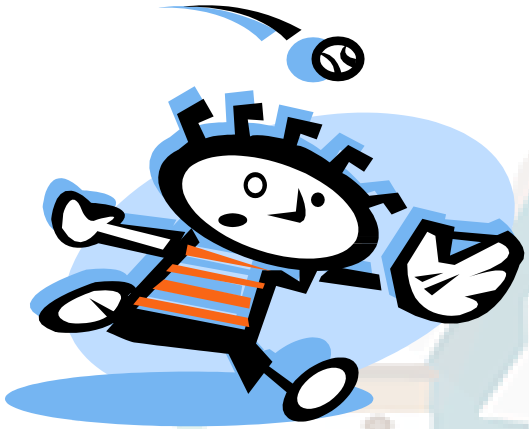
Ejemplo:

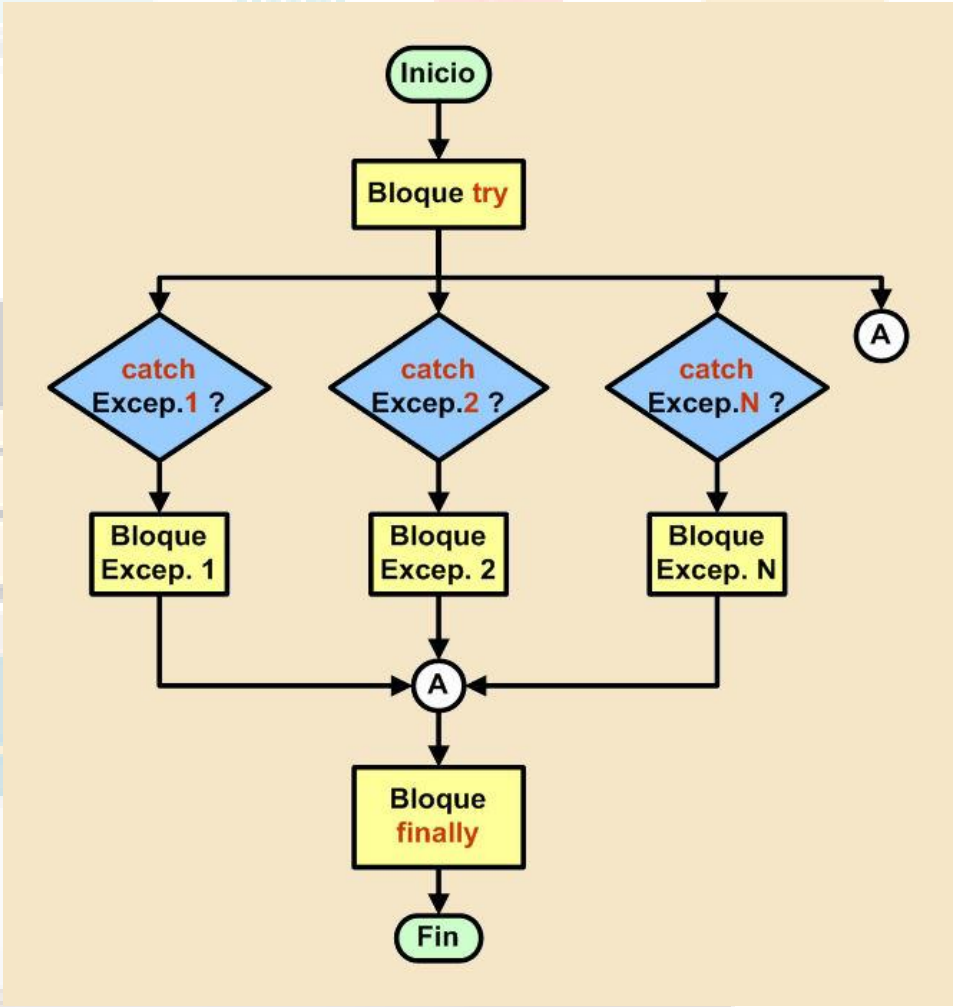
```
switch(opcion) {
  case 'A': System.out.println("Altas");
            break;
  case 'B': System.out.println("Bajas") ;
            break;
  case 'C': System.out.println("Cambios") ;
            break;
default: System.out.println("Opción inválida") ;
      break;
}
```

?

Selectiva intenta

- La estructura intenta (try/catch) se utiliza para atrapar excepciones.





Selectiva intenta…

Sintaxis

```
try
    {
// Bloque de instrucciones que
    pueden generar una o más
    // situaciones excepcionales.
}
catch(Excepción 1)
    {
// Bloque alternativo para el caso
    de existir la Excepción 1
}
```

```
catch(Excepción N )
    {
// Bloque alternativo para el caso
    de existir la Excepción N
}
finally // Opcional
    {
// Bloque de instrucciones que
    siempre se ejecutan
}
```

Try es el bloque de código donde se prevé que se genere una excepción. Es como si decir "intenta estas sentencias y observa si se produce una excepción".

Selectiva intenta…

Ejemplo:

```
int valor;
try {
for( x=0,valor = 100; x < 100; x ++
    )
        valor /= x;
}
catch( ArithmeticException e )
  {
    System.out.println(
    "Matemáticas locas!" );
  }
```

```
catch( Exception e )
  {
    System.out.println( "Se ha
    producido un error" );
}
```

Selectiva intenta…

Ejemplo:

```
try {
        if (temp > 40)
             throw(new
    demasiadoCalor());
       if (dormir < 8) throw(new
    demasiado Cansado());
}
catch(Limites lim)
{
    if (lim instanceof demasiadoCalor)
    {
       System.out.println( "Capturada
    excesivo calor!" );
    return;
    }
```

```
if (lim instanceof
    demasiadoCansado)
{
    System.out.println(
          "Capturada excesivo
          cansancio!" );
    return;
}
}
finally
System.out.println( "En la
    clausula finally" );
```

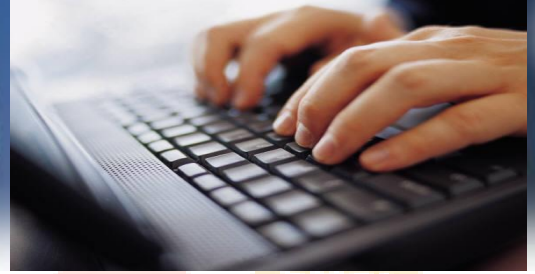
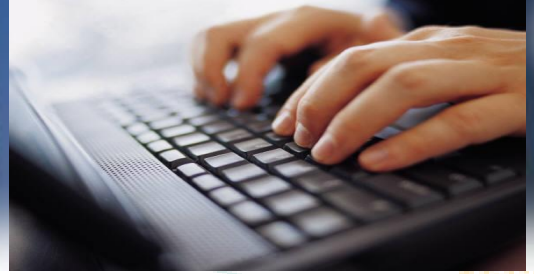
Selectiva anidada…

Ejemplos:

```
if (a != b)
   if (c%2 == 0)
      System.out.print(“Valido”);
  else
      System.out.print(“inválido”);
else
   z = x + y;
```

```
if (k + 2 == 4)
      z = j % 3;
else
      if (k/2 !=0)
            s = cadena1;
      else
            s = cadena2;
```

?

Sintaxis

Selectiva anidada…

b) Con varias instrucciones

```
if (condición(es))
   if (condición(es))
     {
        Bloque 1
     }
   else
     {
        Bloque 2
     }
else
    Bloque 3
```

```
if (condición(es))
    Bloque 1
else
    if (condición(es))
     {
        Bloque 2
     }
    else
     {
        Bloque 3
     }
```

Selectiva anidada…

Ejemplos:

```
if (w != 0)
   if ((j!=i)||(i>0))
   { r = z %w;
    System.out.print (“El
      resultado es”+r);
    }
   else
   { t = m*2;
      e = t / 2;
    }
else
    z =2;
```

```
if (c/2 != 0)
   System.out.print (“Dato válido”);
else
    if ((j!=i)||(i>0))
    { j = t + y;
      System.out.print (“El resultado
       es”+j);
    }
    else
    { r = m / y;
      System.out.print (“Dato
       inválido”);
    }
```

Estructuras Iterativas

Concepto

- Sirven para formar bucles o ciclos en los que se ejecuta repetidamente un bloque de instrucciones.

Iteraciones

- Número de veces que se ejecuta el ciclo

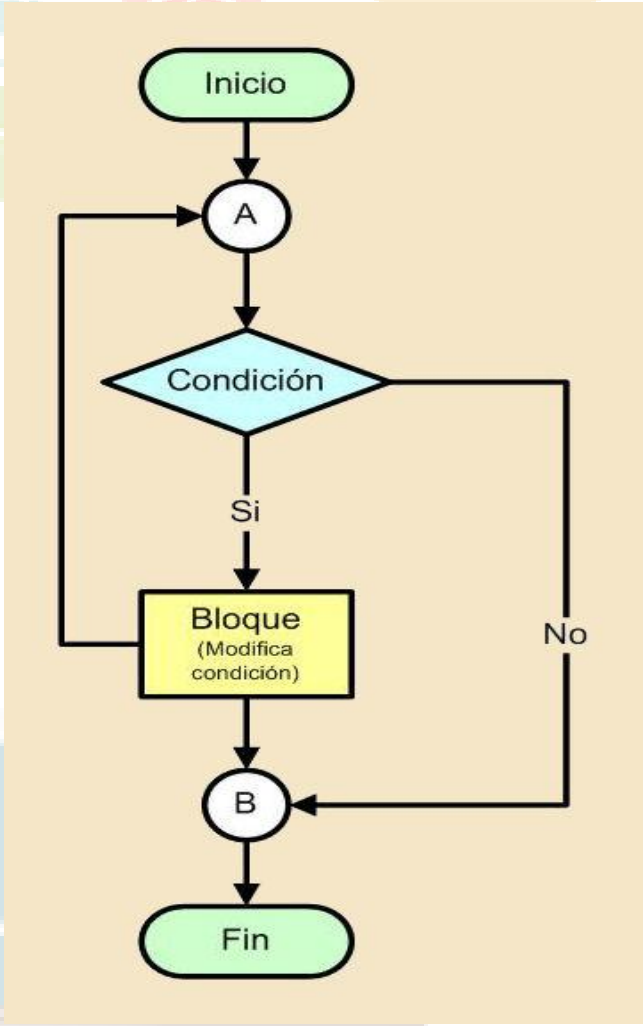
Variable de control de ciclo

- Es la que controla el número de iteraciones de una estructura de repetición.

Repetir mientras (while).

Funcionamiento:

1. Se evalúa una condición.
2. Si es **verdadera**, se ejecuta un bloque de instrucciones, en el cual debe existir una instrucción que modifique la condición, de lo contrario, se ejecutará un ciclo infinito (loop).
3. Si es **falsa**, el bloque de instrucciones no se ejecuta y finaliza la ejecución.



Repetir mientras (while)…

Sintaxis:

```
while( condición )
{
        Bloque;
}
```

Ejemplo 1:

```
int a = 1, b = 5;
while (a < b)
{
   System.out.println (“ a = “+a);
   a = a+1;
}
```

¿Qué pasará si no actualizo la variable de control de ciclo?

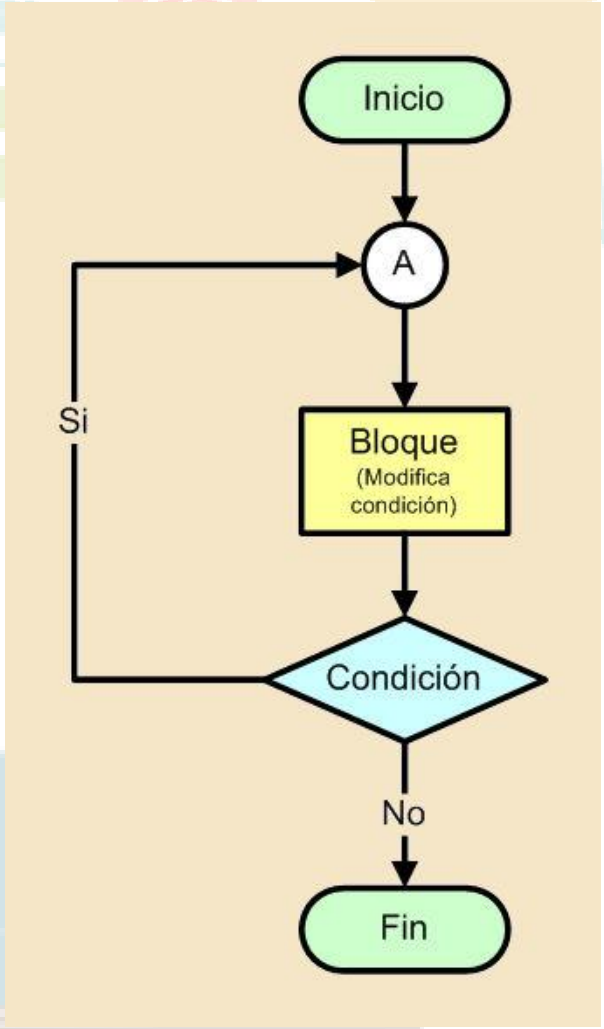
Repetir mientras (while)...

Ejemplo 2

```
boolean x =  true;
System.out.println (“Teclea solo numeros enteros y un 0 para salir”);
while (x== true)
{
  num = Integer.parseInt(entrada.readLine( ));
  if (num == 0)
     x = false;
  else
     if ((num % 2)== 0)
       System.out.println (num+“ es un numero par”);
    else
       System.out.println (num+“ es un numero impar”);
}
```

Repetir hasta (do-while).

- Es similar a while, sólo que primero se ejecuta el bloque de instrucciones y después se evalúa la condición.
- Con esto se asegura que el bloque se ejecutará al menos una vez.



Repetir hasta (do-while)…

Sintaxis:

```
do
{
    Bloque ;
}
while(condición);
```

Ejemplo 1

```
int digito = 0;
do
{
   System.out.println (“        “+digito);
   digito = digito + 1;
} while (digito <= 9);
```

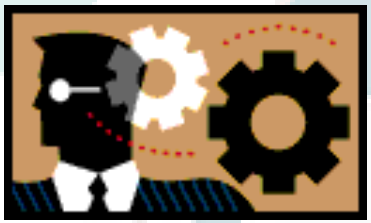
Repetir hasta (do-while)…

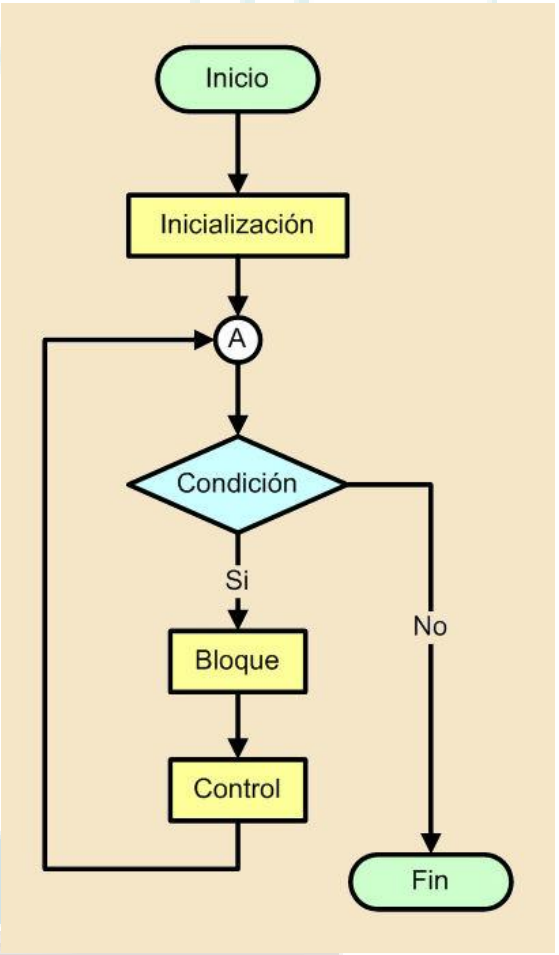
Ejemplo 2

```
int i = 1, N = 5, valor, prom = 0;
do
{
   System.out.println("Teclea el valor = ");
   valor = Integer.parseInt (entrada.readLine( ));
   prom = prom + valor;
   valor = 0;
} while(i <= N);
System.out.println("El promedio de los "+N+" valores dados es = "+prom);
```

Repetir desde (for).

- Es la más versátil de las estructuras de iteración, ya que permite la declaración de variables dentro de su estructura.





Repetir desde (for)…

Sintaxis:

```
for( Inicialización; Condición; Control)
{
        Bloque ;
}
```

donde:

Inicialización es un bloque de instrucciones que puede incluir la declaración de las variables de control de ciclo involucradas y la asignación de valores iniciales.

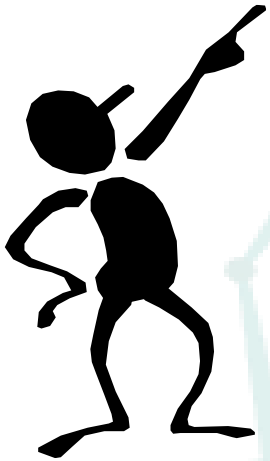
Condición deberá ser verdadera para que se ejecute el Bloque de instrucciones.

Control es una o más instrucciones, separadas por comas, que controlan la variación de los valores de las variables de control de ciclo utilizadas.

Repetir desde (for)…

Ejemplo 1:

```
for(int i = 1 ; i <= 10 ; i++)
    System.out.println("Soy la línea “+ i ) ;
```



Ejemplo 2:

```
int i, calif=0, prom=0;
for(i = 1 ; i <= 30 ; ++i)
{
   System.out.println(“Teclea la  calificación del alumno = “+ i) ;
   calif = Integer.parseInt(entrada.readLine( ));
   prom+= calif;
}
 System.out.println(“El promedio del grupo es = “+ prom /30) ;
```

Ciclos anidados

Ejemplos:

```
for(i = 1 ; i <= 10 ; i++){
   for(j=1; j <= 5; ++j)
         System.out.println("j = "+j );
   System.out.println("i = "+ i );
}
while (x < t)
{
   for(a=5; a >=0; --a)
        m = m*2;
   System.out.println("m = "+ m) ;
   ++x;
}
```

```
do{
    z=10;
   while(z > 0)
  {
    System.out.println("Teclea el salario =");

salario=Integer.parseInt(lectura.readLine())
;
    neto=salario – deducciones;
   System.out.println("Salario Neto
="+neto);
     --z;
  }
  ++depto;
} while (depto < 5)
```