2.- Fundamentos y uso de variables.

2.1.- Declaración de variables y los tipos de variables primitivos.

Las variables son los espacios en memoria que se utilizan para almacenar la información que se procesa. Se comportan al igual que una memoria de una calculadora, almacenan información y si se vuelve a utilizar la misma memoria la información ahí almacenada se actualiza borrándose la información anterior.

Antes de utilizar una variable es necesario declararla, al declarar una variable especificamos su tipo y su nombre. Los tipos primitivos de Java simplifican la manipulación de información. Cuando se declara una variable de tipo primitivo automáticamente se le asigna espacio en la memoria del programa y no se requiere ninguna acción adicional por parte del programador. Sin embargo si la variable no es de tipo primitivo es necesario solicitar la memoria antes de utilizarla.

Los tipos simples y más comunes son los mostrados en la tabla 2.1.

TIPO	DESCRIPCION	CARACTERISTICAS
int	Se utiliza para almacenar variables enteras, por ejemplos numero de personas, numero de autos y los valores de las variables discretas.	primitivo
long	Es lo mismo que **int** pero con capacidad de almacenar números más grandes.	primitivo
float	Este tipo nos sirve para declarar variables cuyo valor tiene decimales, como por ejemplo medidas de pesos, alturas y cantidades fraccionarias.	primitivo
double	Es lo mismo que float, pero con capacidad de almacenar números más grandes.	primitivo
char	Es utilizado para declarar variables cuyo valor puede ser una sola letra o símbolo.	primitivo
boolean	Este tipo de variables solo admite dos valores **true** y **false** es muy útil para almacenar respuestas a preguntas.	primitivo
String	Se utiliza para almacenar cadenas de caracteres, normalmente palabras frases o códigos de identificación.	Compuesto pre definido por Java

Tabla 2.1 Tipos de variables más comunes en Java.

Cuando declaramos una variable seguimos la siguiente sintaxis:

<tipo> <nombre de la variable> ;

Por ejemplo para declarar la variable numHijos entera escribimos:
int numHijos;
Para declarar la variable salario como flotante escribimos:
float salario;
Para declarar e inicializar la variable pesoKgs como flotante escribimos:
float pesoKgs=0;
Si vamos a leer una variable con la facilidad Ip.wlee es necesario que se le haya asignado un valor previamente.

Java proporciona una gran cantidad de tipos predefinidos, pero además el programador puede definir sus propios tipos también llamados clases.

El siguiente ejemplo que se encuentra en el programa "DeclaracionVariables" ilustra como declaramos una variable y utilizando el inovaProg leemos variables tecleadas por el usuario del programa y posteriormente las despliega en una ventana.

```
int i=0;
float x=0;
long j=0;
double f=0;
 i = Ip.wlee("Valor de i ",i);
 x = Ip.wlee("Valor de x ",x);
 j= (long) Ip.wlee("Valor de j ",i);
 f = Ip.wlee("Valor de f ",f);
 Ip.wescribeMensaje("Valor de i = " + i + " valor de x = " + x
                + "\nValor de j = " + j + " valor de f = " + f);
```

2.2.- Nombres de variables y comportamiento de las mismas.

Es recomendable utilizar nombres ilustrativos del valor y uso que se les piensa dar a las variables, iniciar su nombre con minúscula y poner mayúscula cuando se quiere separar una palabra de otra, por ejemplo: numeroHijos.

No se permite iniciar nombres de variables con números aunque si se permite utilizar números en el nombre de una variable siempre y cuando se inicie con una letra, por ejemplo: trimestre2 sería un nombre válido. Estas mismas reglas aplican para los nombres de los programas.

La memoria ram es un enorme arreglo de celdas que pueden almacenar un dato o una parte de un dato cada una de ellas, para almacenar una variable se puede requerir una o más celdas.

Una metáfora de mucha ayuda para aprender el manejo y la manipulación de las variables es la siguiente: se puede pensar en las variables como si fueran recipientes y en los datos como si fueran líquidos que se guardan en dichos recipientes. Por ejemplo no podemos poner leche en un recipiente que se ha utilizado para aceite (respetar el tipo de la variable) y no podemos poner más líquido que lo que marca la capacidad del recipiente, si tenemos tres litros de leche requeriremos de un recipiente de tres litros para almacenarla si la ponemos en un recipiente más pequeño la leche se derramaría esto en programación se llama *overflow*.
Si tenemos en un recipiente leche fresca y en otro recipiente agua y deseamos intercambiar los líquidos de tal manera que el primer recipiente que actualmente tiene leche contenga el agua y el segundo recipiente que actualmente contiene agua quede conteniendo la leche. Evidentemente vamos a necesitar un tercer recipiente para hacer el intercambio de tal manera que podemos poner la leche en el tercer recipiente en forma temporal para posteriormente completar el intercambio. Lo mismo se hace en un programa java cuando queremos intercambiar el valor de dos variables, por ejemplo si tenemos las variables a y b declaradas como enteros tendríamos:

```
int a=5;
int b=20;
```

se requiere una tercer variable temporal (el tercer recipiente involucrado)

```
int temp;
```

para llevar a cabo el intercambio

```
temp=a;
a=b;
b=temp;
```

Una vez ejecutadas estas instrucciones el intercambio de valores entre a y b se ha completado, la analogía con recipientes y contenidos es evidente. Para comprobarlo compile y ejecute el programa adjunto llamado “Intercambio”.

2.3.- Expresiones aritméticas y operadores.

Para asignar valores a una variable (a la celda de memoria que contiene el valor de una variable) se cuenta con el operador **=** éste operador permite asignar el contenido de una literal, de otra variable o el resultado de un cálculo. Veamos las siguientes instrucciones:
Asumiendo que tenemos las siguientes declaraciones

```
int a = 3;
int b = 5;
int c = 4;
```

podemos escribir

```
c = 8;
```

cambiando el valor de c a 8, 8 es una literal

tambien podemos escribir

```
c = a ;
```

cambiando el valor de c a 3 que es el valor actual de a
o bien

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

que cambia el valor de c a 8 que es el resultado de la operación de sumar a más b,

Hay algunas reglas sencillas para formar las expresiones aritméticas del tipo a + b las mas importantes son:
Primera Regla: Los operadores permitidos son: **+**, **-** , ***** y **/** también se permite el cálculo del residuo de una división con el operador **%**
Segunda Regla: El orden en el que se hacen las operaciones (también llamada jerarquía de los operadores) es: ***** , **/**, **%** posteriormente **+**, **-** en caso de igual jerarquía se procede de izquierda a derecha.
Tercera Regla: Se pueden utilizar paréntesis para alterar el orden de las operaciones cuando sea necesario.
Cuarta Regla: Las operaciones aritméticas en los cálculos deben hacerse de tal manera que los tipos de las variables de la derecha sean de menor o igual capacidad que los de la izquierda, el orden de los tipos empezando por el de mayor capacidad es:

```
double, float, long, int
```

Esto es se puede asignar valores **int** a variables de tipo **long, flota o double** sin ningún problema, pero no viceversa. Volviendo a la analogía de los recipientes, el recipiente más grande son los **double** y los más pequeños son los **int**.

En caso de requerir asignar un valor tipo **long** a uno tipo **int** rompiendo la regla, es necesario utilizar una instrucción de moldeo o *cast* como se muestra:

```
int a = 3;
long b = 5;
b= (long) a;
```

Note el tipo al que se va asignar entre paréntesis

Veamos los siguientes ejemplos de expresiones aritméticas con variables enteras:

```
int a = 3;
int b = 5;
int c = 4;
int r;
r=a+b+c;
r=a*b/c + 2;
r=c + a*b;
```

r toma los valores de 12, 5 y 19. Cuando se utiliza aritmética entera los resultados se truncan eso explica el resultado de 5 en la segunda expresión.

Por otra parte los paréntesis alteran el orden de la evaluación de la expresión y tenemos que la expresión **`r=(c + a)*b;`** nos da 35 como resultado utilice el programa "ExpresionesAritmeticasEnteras" para comprobar lo anterior.
Intente cambiar la expresión **`r=a*b/c + 2;`** a
`r=a*b/c + 2.0;`
Observe que Java envía un mensaje de error en la compilación lo cual se debe a que el 2.0 es interpretado como de doble precisión o tipo **double**. Para evitar este error no utilice puntos decimales cuando esté utilizando aritmética entera o flotante sencilla. Si requiere hacerlo tendrá que utilizar el moldeo.

Veamos los siguientes ejemplos de expresiones aritméticas con variables flotantes:

```
float a = 3;
float b = 5;
float c = 4;
float r;
r=a+b+c;
r=a*b/c + 2;
r=c + a*b;
```

Los resultados obtenidos respectivamente son: 12.0, 5.75 y 19.0
Para **`r=(c + a)*b;`** se obtiene 35.0 observe el punto decimal en los resultados y que no hay truncamiento sino un calculo completo.
Si cambia la expresión **`r=a*b/c + 2;`** por **`r=a*b/c + 2.0;`**
Sigue obteniendo un error de compilación, el cual desaparece si declara r como **`double`**.
Utilice el programa "ExpresionesAritmeticasFlotantes" para comprobar lo anterior.

2.4.- Expresiones lógicas y operadores.

Los resultados de las expresiones lógicas conducen a resultados que se almacenan en variables boleanas, los operadores que actúan sobre las variables boleanas son los siguientes:

Operador "**`o`**" que se escribe como || y es un operador binario o sea que necesita dos operandos, proporciona **`true`** como resultado si uno de sus operandos es **`true`**

Operador "**`y`**" que se escribe **`&&`** y es un operador binario o sea que necesita dos operandos, proporciona **`true`** como resultado si los dos operandos tienen el valor de **`true`**

Operador "**`no`**" que se escribe **`!`** y es un operador unitario o sea que solo necesita un operando, proporciona el valor contrario del operando esto es **`true`** si el operando es **`false`** y **`false`** si el operando es **`true`**.

La jerarquía de los operadores es la siguiente: primero el operador **"no" !**, posteriormente el operador **"y" &&** y finalmente el operador **"o"** ||
Pueden utilizarse paréntesis tal como en las expresiones aritméticas para alterar el orden en el que se aplican los operadores.

Por ejemplo teniendo las declaraciones que se muestran veamos los siguientes ejemplos:

```
boolean a = true;
boolean b = false;
boolean c = true;
boolean r;
r = a || b;
r = a && b;
r = !b;
r = !a;
```

Obteniendo como resultados **true,false,true y false** respectivamente
Estas expresiones conducen a valores que son utilizables en la expresión condicional **if**
De la siguiente forma

```
If (r)
//expresión que se ejecuta si el valor de r es true
else
//expresión que se ejecuta si el valor de r es false
```

Puede utilizarse el programa "ExpresionesBoleanas" para probar los ejemplos mostrados.
Las comparaciones aritméticas conducen también a resultados boleanos los operadores permitidos en las operaciones aritméticas son:

- **>** mayor que,
- **<** menor que,
- **>=** mayor o igual,
- **>=** menor o igual
- **==** igual que

Note que se utilizan dos signos de igual para el operador de comparación por igualdad ya que para la asignación se utiliza solo uno. Estos operadores solo funcionan con tipos primitivos. Observemos los siguientes ejemplos:

```
int a = 3;
int b = 3;
int c = 4;
boolean r;
r = a == c;
if (r)
    Ip.wescribeMensaje(" a es igual  b ");
else
    Ip.wescribeMensaje(" a no es igual  b ");
```

```
if (a == c)
    Ip.wescribeMensaje(" a es igual  b ");
else
    Ip.wescribeMensaje(" a no es igual  b ");
r = a >= c;
if (r)
    Ip.wescribeMensaje(" a es mayor o igual c ");
else
    Ip.wescribeMensaje(" a es menor  c ");
if (a >= c)
    Ip.wescribeMensaje(" a es mayor o igual c ");
else
    Ip.wescribeMensaje(" a es menor  c ");
```

El ejemplo anterior detecta que a no es igual a b y que a es menor que c.
Estas comparaciones también funcionan para long, float,doublé y char, pero no para el tipo String ya que este no es un tipo primitivo. Pude utilizar el programa "ComparacionesAritmeticas" para comprobar estos ejemplos, haga el cambio a variables flotantes y dobles para observar el funcionamiento, intente hacer lo mismo con variables tipo *String* para observar los problemas que se presentan.

2.6.- Expresiones de cadenas de caracteres y operadores

Las variables de cadenas de caracteres son de tipo *String*, por lo que se comportan diferente a las variables de tipo primitivo. Cuentan con un grupo de rutinas llamadas métodos que nos apoyan en su manipulación. La operaciones más comunes de este tipo de variables son la concatenación, comparación por igualdad y comparación general de dos variables de este tipo:

La concatenación es la unión consecutiva de dos o más variables, su operador es el signo **+** y la ilustramos a continuación:

```
String nombre = "Juan ";
String apellido = "Perez";
String nombreCompleto = "";
nombreCompleto = nombre + " " + apellido;
```

El valor de la variable nombre completo queda como "Juan Perez", con un espacio en blanco entre el nombre y el apellido.

La comparación por igualdad de dos variables tipo *String* tiene como resultado un valor tipo boleano o sea **true** o **false** se lleva a cabo por medio del uso de uno de los métodos previamente definidos para el tipo *String*. Por ejemplo:

```
String nombre1 = "Juan ";
String nombre2 = "Pedro";
boolean mismaPersona;
mismaPersona = nombre1.equals(nombre2);
if(mismaPersona)
     Ip.wescribeMensaje(nombre1 + " y " + nombre2 +
         " son iguales");
else
      Ip.wescribeMensaje(nombre1 + " y " + nombre2 +
         " no son iguales");
```

La comparación general alfabética de dos variables tipo *String* tiene como resultado un numero entero, si es menor a 0 la posición en el alfabeto de la primera variable es menor que la posición en el alfabeto de la segunda, si es 0 las dos variables son iguales y si es mayor a cero la posición en el alfabeto de la primera variable es mayor que la posición en el alfabeto de la segunda. Esta comparación se lleva a cabo por medio del uso de otro método previamente definidos para el tipo *String*. Por ejemplo:

```
String nombre1 = "Juan ";
String nombre2 = "Pedro";
Int    resultadoComparacion;
resultadoComparacion = nombre1.compareTo(nombre2);
if(resultadoComparacion < 0)
     Ip.wescribeMensaje(" La posición de " +
 nombre1 + " antecede a la de " + nombre2) ;
if(resultadoComparacion == 0)
     Ip.wescribeMensaje(" La posición de " +
 nombre1 + " y la posición de " + nombre2
+ " son iguales ") ;
if(resultadoComparacion > 0)
     Ip.wescribeMensaje(" La posición de " +
 nombre1 + " procede a la de " + nombre2) ;
```

El programa anterior ejemplifica este tipo de comparaciones. Puede comprobarlo utilizando el programa "ExpresionesCadenasCaracteres".

2.7.- Alcance de la existencia de las variables.

Una vez declarada una variable tiene una vida, esto es un alcance definido, una variable puede existir durante todo el programa o solo dentro de un trozo de el.

La regla del alcance de las variables es muy sencilla, una variable existe y puede ser utilizada después de haber sido declarada en el código que se encuentre dentro del par de llaves { } en el que se encuentra su declaración.

Veamos la figura 2.1.

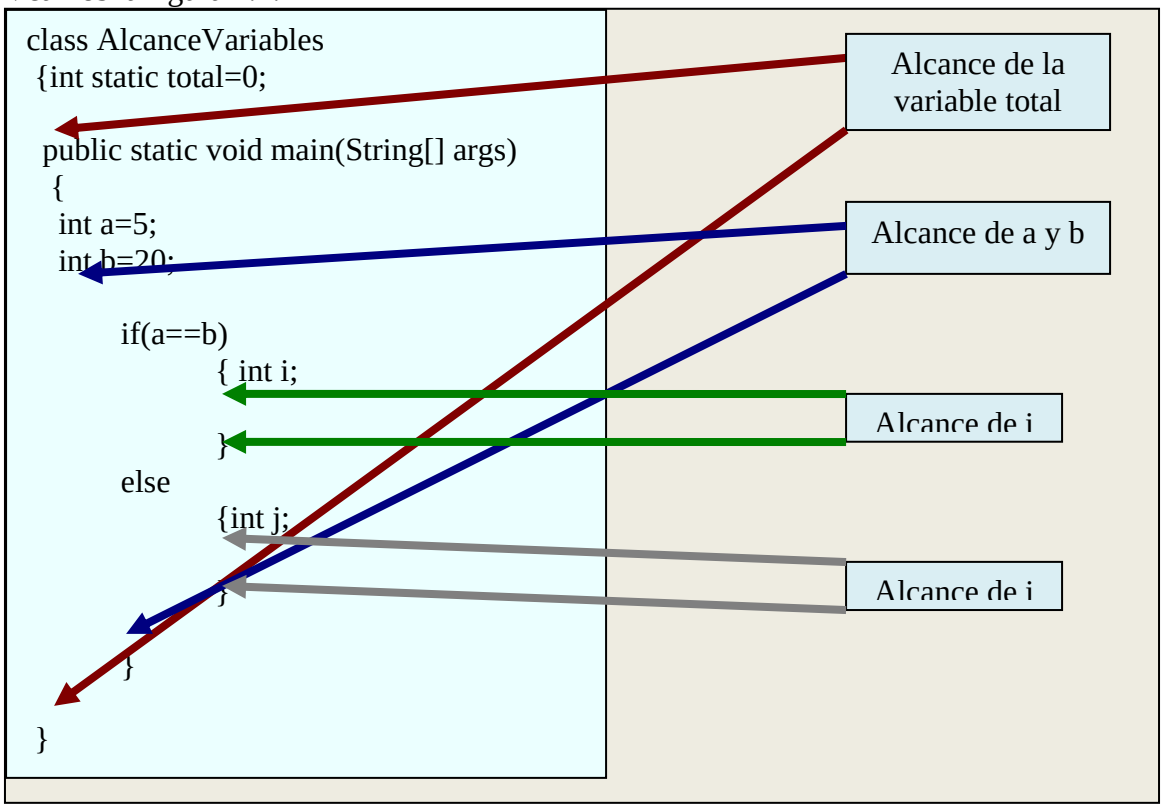

Figura 2.1 Alcance de variables

Cualquier instrucciones que use una variable fuera de su alcance produce un error de compilación, utilice la plantilla "AlcanceVariables" para agregarle instrucciones que utilicen las variables declaradas y observe los errores de compilación que se pueden provocar.

2.8.- Partes de una variables: nombre, valor, tipo , referencia y alcance.

En resumen podemos concluir que una variable esta formada por las siguientes partes:
Nombre .- Es el mnemónico o símbolo o conjuntos de símbolos que nos permite referirnos a la variable en el programa para utilizar su valor en las instrucciones. Es constante durante todo el programa.
Valor .- Es la cantidad o el contenido de una variable en un momento dado puede variar durante el programa.
Tipo .- Son las características de los valores que se pueden almacenar en una variable, permanece constante durante todo el programa.
Referencia .- Es la dirección o número de la primera celda en la que se guarda una variable, al igual que el valor puede variar durante la ejecución de un programa, normalmente el programador no tiene acceso a ella en forma absoluta, o lo que es lo mismo nunca manipula su valor.
Alcance .- Es el trozo de código en el cuál se puede utilizar una variable.

Gráficamente podemos representar las siguientes relaciones entre los distintos universos de nombres, tipos, valores y referencias como se muestra en la figura 2.2.

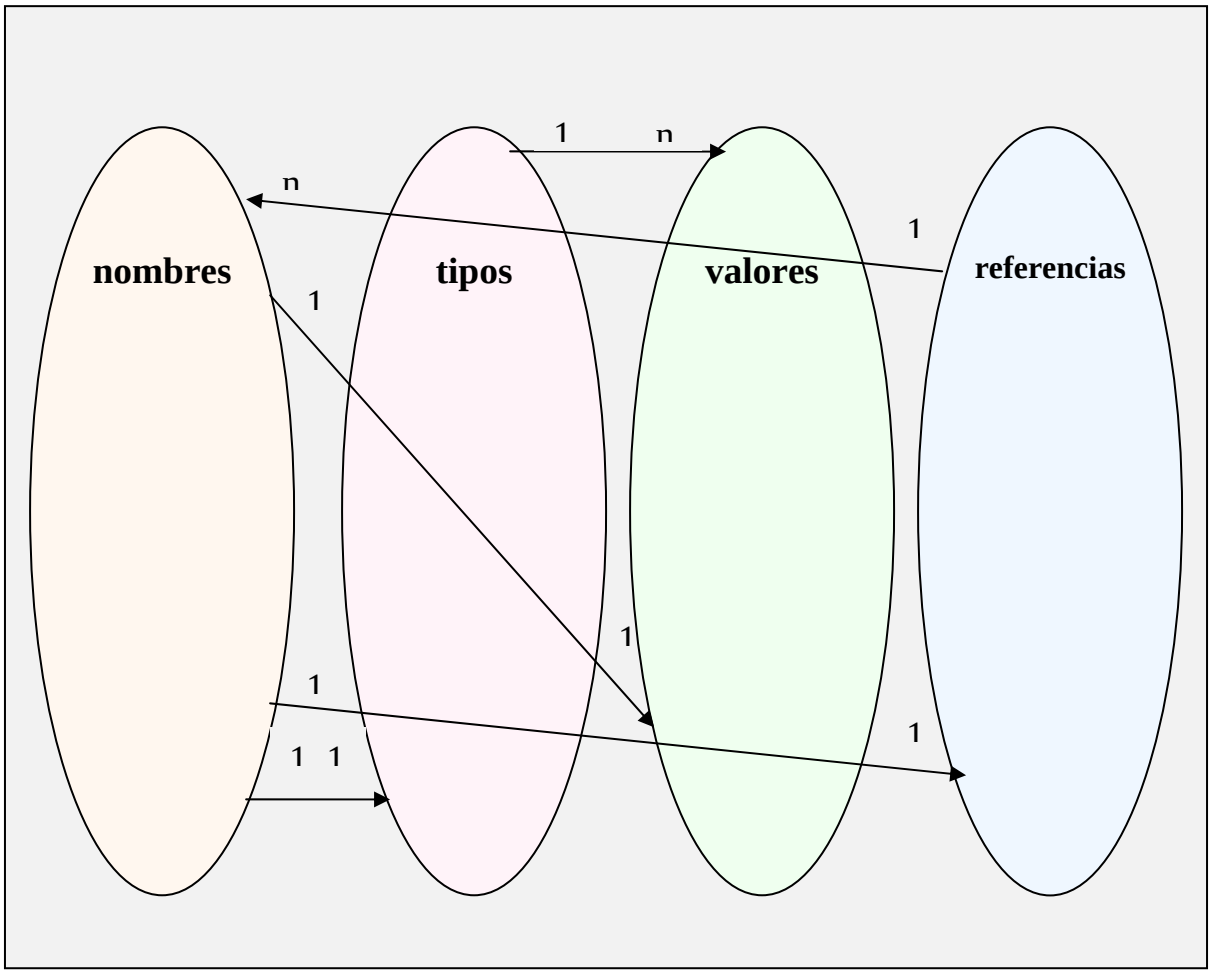

Figura 2.2 Relaciones entre los conjuntos de nombres, tipos, valores y referencias de las variables de un programa.

Como se observa en la figura 2.2 un nombre puede estar ligado a un tipo, a una referencia y a un valor. Un tipo puede estar relacionado con n valores y una referencia puede ser utilizada por n nombres a estos se les llama sinónimos, o lo que es lo mismo dos o mas nombres para una misma celda de memoria.

Hemos cubierto los bloques que forman los cimientos de la programación, sobre ellos construiremos los programas.

2.9.- Actividades:
Compile y ejecute los ejemplos adjuntos.

2.10.- Ejercicio Propuesto:
Escriba un programa que defina variables con su nombre su edad, su peso y su altura utilizando variables de tipos adecuados para cada dato y para terminar despliegue sus datos en una ventana por dato.