Unidad 2

2.1 Identificadores

Tomemos como ejemplo el siguiente programa que calcula la media de tres números (tipo double) introducidos por teclado, calcula la media almecenándola en una variable y luego muestra el resultado:

```
import java.util.Scanner; // Scanner is in the java.util package
public class ComputeAverage {
    public static void main(String[] args) {
        // Create a Scanner object
        Scanner input = new Scanner(System.in);
        // Prompt the user to enter three numbers
        System.out.print("Enter three numbers: ");
        double number1 = input.nextDouble();
        double number2 = input.nextDouble();
        double number3 = input.nextDouble();
        // Compute average
        double average = (number1 + number2 + number3) / 3;
        // Display results
        System.out.println("The average of " + number1 + " " + number2 + " " +
number3 + " is " + average);
    }
}
```

Los **identificadores** son los nombres que identifican los elementos tales como las clases, métodos y las variables de un programa.

Según el programa anterior, **ComputeAverage**, **main**, **input**, **number1**, **number2**, **number3** son identificadores. Los cuales deben de cumplir las siguientes normas:

- Un **identificador** es una secuenca de caracteres que consisten en letras, dígitos, guiones bajos (_) y signo dollar (**$**).
- Un **identificador** de comenzar por una letra, guión bajo o dollar, nunca con un dígito o espacio en blanco.
- Un identificador no podrá ser ni **true**, ni **false** ni **null**.
- Un identificador puede tener cualquier longitud.

Por ejemplo: **$2, ComputeArea, ara, radius, input** son identificadores legales, mientras que **2A, d+4, mi variable** no lo son.

> Recuerda que Java es *case sensitive*: **area** no es igual a **Area**.

Los identificadores se utilizan para nombrar variables, métodos, clases y otros items de un programa. Identificadores descriptivos hacen programas fáciles de leer.

- Evita usar abreviaciones para identificadores
- Utiliza palabras completas: **numeroDeAlumno** es mejor que **numAlu,numDeAlu**.
- A veces es mejor utilizar nombres sencillos: **i, j, k, x, y, s** pero como método de aprendizaje ya que es más breve.

- No poner un **$** delante de un identificador normalmente ya que esto es para cuando se genera código automáticamente.la almacena en una variable

2.2 Variables

Las variables repesentan valores que pueden cambiar en un programa, dichos valores se almacenan en posiciones de memoria.

```
1  // Compute the first area
2  radius = 1.0;                                          radius:  1.0
3  area = radius * radius * 3.14159;                        area:  3.14159
4  System.out.println("The area is " + area + " for radius " + radius);
5
6  // Compute the second area
7  radius = 2.0;                                          radius:  2.0
8  area = radius * radius * 3.14159;                        area:  12.56636
9  System.out.println("The area is " + area + " for radius " + radius);
```

Podemos ver en el programa anterior, que `radius` es una variable con valor `1.0`, y con esa variable se puede realizar un cálculo que se asigne a la variable `area`. Y luego se imprime el resutlado de ambas variables. Esto lo puodemos hacer las veces que queramos, pero los valores pueden ir cambiando.

Para utilizar una variable tiene que declararla, de esta forma le dice al compilador que busque un espacio en memoria para esa variable según del tipo que sea.

La sintaxis de declaración de una variable es:

`TipoDeDatos NombreVariable;`

Ejemplos de declaraciones:

```
int count;          // Declaramos un contador tipo entero
double radius;      // Declaramos un radio de tipo doble
double tasaInteres; // Declaramos una tasa de interés de tipo double
```

Si las variables son del mismo tipo se pueden declarar juntas separadas por comas:

```
tipoDatos variable1, variable2, ...., variablen;
```

```
int i, j, k;
```

Se pueden inicializar las variables de la siguiente forma:

```
int count = 1;
//equivalente a
int count;
count = 1;
```

También puedes declarar e inicializar variables de forma conjunta para el mismo tipo:

```
int i = 1, j = 2;
```

> **Una variable debe ser declarada antes de que se le asigne el valor, y en general, antes de que pueda utilizarse en cualquier parte del programa. Cuando sea posible declarar la variable e inicializarla en un paso.**

Cada variable tiene lo que se llama **alcance de la variable** que es donde se puede utilizar o acceder. Es un concepto que iremos viendo en siguientes unidades.

¿Cuál es el error de este programa?:

```
1  public class Test {
2    public static void main(String[] args) {
3      int i = k + 2;
4      System.out.println(i);
5    }
6  }
```

2.3 Constantes

Una constante es un identificador que representa un valor permanente.

El valor de una variable puede cambiar, pero el de una constante nunca cambia. También se llama **variable final**.

En un programa que calcule el área se utiliza mucho el valor PI, el cual es constante, lo podemos utilizar como constante.

Al declarar una constante:

```
final TipoDeDatos NOMBRE_DE_LA_CONSTANTE = valor;
```

- Las constantes deben ser siempre inicializadas.
- La palabra `final` es para decir que es una variable.
- Por convención, se utilizan mayúsculas para declarar variables.

En el siguiente ejemplo se puede ver la declaración de la variable **CM_PER_INCH** :

```
public class Constants
{
   public static void main(String[] args)
   {
      final double CM_PER_INCH = 2.54;
      double paperWidth = 8.5;
      double paperHeight = 11;
      System.out.println("Tamaño del papel en cm: "
         + paperWidth * CM_PER_INCH + " por " + paperHeight * CM_PER_INCH);
   }
}
```

Existen tres beneficios de utilizar constantes:

- No tienes que repetir el mismo valor cuando lo uses.
- Si tienes que cambiar el valor de la variable, lo cambias en un sólo sitio.

- Los nombres descriptivos de las constantes ayudan a entender mejor el programa.

2.4 Convenciones en el uso de nombres

Seguir las convenciones de los nombres en Java hace los programas más fáciles de leer y evita errores.

- Utiliza **lowerCamelCase** para nombrar variables si es largo el nombre: `numberOfStudents`
- Para nombres sencillos de variables en minúscula: `area, radio`
- Utiliza la primera en mayúscula (**UpperCamelCase**) para nombrar las clases: `public class CalcularArea {...}` O también: `public class Sistema {...}`
- Capitalizar las constantes y separarlas si fuera necesario por `_`: `PI, VALOR_MAX`

2.5 Tipos de Datos Numéricos y sus operaciones

Java tiene seis tipos de datos numéricos y de tipo punto flotante con los operadores +, *, -, / y %

Tipos Numéricos

Cada tipo de datos tiene un rango de valores. El compilar asigna espacio de memoria para cada variable o constante de acuerdo a su tipo de datos. Java provee **ocho tipos de datos primitivos para valores numéricos, caracteres y booleanos: byte, short, int, long, float, double, char, boolean**.

En la siguiente tabla se ven los tipos de datos numéricos:

TABLE 2.1 Numeric Data Types

Name	*Range*	*Storage Size*	
`byte`	-2^7 to 2^7-1 (-128 to 127)	8-bit signed	byte type
`short`	-2^{15} to $2^{15}-1$ (-32768 to 32767)	16-bit signed	short type
`int`	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	32-bit signed	int type
`long`	-2^{63} to $2^{63}-1$	64-bit signed	long type
	(i.e., -9223372036854775808 to 9223372036854775807)		
`float`	Negative range: $-3.4028235E+38$ to $-1.4E-45$	32-bit IEEE 754	float type
	Positive range: $1.4E-45$ to $3.4028235E+38$		
`double`	Negative range: $-1.7976931348623157E+308$ to $-4.9E-324$	64-bit IEEE 754	double type
	Positive range: $4.9E-324$ to $1.7976931348623157E+308$		

> El estándar **IEEE 754** se utiliza para representar números en punto flotante. Este estándar se utiliza ampliamente. Java utiliza 32 bit IEEE 754 para los datos de tipo **float** y 64-bit IEEE 754 para el tipo **double**.

Java utiliza cuatro tipos de datos enteros: byte, short, int, long. Básicamente es el tamaño ocupado en memoria lo que va dar la elección de un tipo u otro.

Java utiliza dos tipos de números en punto flotante: float y double. El **double** es dos ves más grande que el **float**. También se dice que el double es de doble precisión y el float de simple precisión. Se suele utiizar el double para mayor precisión.

> Noticia de cómo existen grandes errores en los tipos de datos elegidos: link

Números enteros (literales enteros)

Un entero puede ser asignado a una variable de tipo entero pero tiene que tener el espacio suficiente. Por ejemplo un entero de tipo byte se puede declarar: `byte b = 128` y nos daría un error de compilación ya que si vemos el límite que tiene el tipo byte es de **-128 a 127**.

Un tipo **int** tiene los límites 2^{31} (-2147483648) y 2^{31-1} (2147483647) . No podrías exceder estos límites en una variable de tipo int.

Un tipo **long** se denota con una **L** ó **l (l minúscula)** al final del valor. Por ejemplo para escribir el entero 2147483648 en un programa Java tienes que escribir `2147483648L` ó `21474836481`.

Por defecto un entero es un número decimal entero. Para denotar otras bases:

- Entero binario: utilizar `0b` ó `0B`
- Entero octal: utilizar el `0` como prefijo del número.
- Entero Hexadecimal: usar `0x` ó `0X`

```
System.out.println(0B1111); // Muestra 15
System.out.println(07777); // Muestra 4095
System.out.println(0XFFFF); // Muestra 65535
```

Para mejorar la lectura de números Java permite utilizar guiones bajo:

```
long ssn = 232_45_4519;
long creditCardNumber = 2324_4545_4519_3415L;
```