Tema 2: EL TIPO DE DATOS ENTERO. INSTRUCCIÓN DE ASIGNACIÓN Y DE COMPOSICIÓN SECUENCIAL

Cualquier duda sobre el contenido de este tema se puede enviar al foro TEORIA2.

2.1.- El tipo de datos entero (`byte, short, int, long`)

2.1.1.- Dominio de valores:

Es el conjunto de los números enteros Z.

En realidad, se trata de un subconjunto de Z, ya que Z es un conjunto infinito y no se pueden representar todos los números enteros. Qué subconjunto de Z se trata depende de cuántos bits se utilicen para su representación interna.

2.1.2.- Representación interna

Un bit es un dígito binario, el cual puede tomar uno de 2 posibles valores: {0, 1}.

Con n bits se pueden representar 2^n números distintos.

Por ejemplo, con 3 bits:

3 bits → 2^3 valores → { 000, 001, 010, 011, 100, 101, 110, 111}
Representan: { 0, 1, 2, 3, -4, -3, -2, -1}

Se usa una representación denominada *en complemento a 2*, la cual permite representar los números positivos (aquellos cuyo primer bit es 0) y los números negativos (aquellos cuyo primer bit es 1).

Con 8 bits se pueden representar 256 números: {-128, ... , 0 , ... 127}
Con 16 bits se pueden representar 65536 números: {-32768, ... , 0 , ... 32767}
Con 32 bits son 4.294.967.296 números: {-2.147.483.648,....,0,.....2.147.483.647}
Con 64 bits se pueden representar estos: {-9.223.372.036.854.775.808,..,0,..+9.223.372.036.854.775.807}

http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html

2.1.3.- Representación externa y definición en Java

En Java los tipos `byte, short, int` y `long` son enteros de 8, 16, 32 y 64 bits.

Para definir (o declarar) una variable de tipo entero en Java hay que dar el nombre del tipo entero (`int`, `long`) dependiendo de la precisión que se quiera y el nombre de la variable, el cual debe ser un identificador.

Las constantes de tipo `int` son números como `4378, -353`, etc. Las constantes de tipo `long` también, pero deben acabar con la letra ele `"l": 4378l, -353l,…`

Ejemplos de definiciones de variables enteras en Java:

```
int edad;
long cant1, cant2;
```

Al definir una variable se le puede dar o asignar un valor inicial.

```
int num = 28; // Se define la variable entera num con valor inicial 28
```

2.1.4.- Operaciones

Aritméticas

Los operadores +, -, *, / (cociente con operandos enteros), % (resto de la división entera). Los resultados de estas operaciones son también números enteros.

Ejemplo: `4+3 → 7` `9/7 → 1` `9%7 → 2` `3*5 → 15`

Comparación

Hay otros operadores de comparación (<, <=, >, >=, ==, !=) cuyos resultados son valores booleanos o lógicos (que se verán en el siguiente tema)

<	MENOR QUE
<=	MENOR O IGUAL QUE
>	MAYOR QUE
>=	MAYOR O IGUAL QUE
==	IGUAL A
!=	DISTINTO DE

Ejemplo:

`3<=6 →` cierto `(true)` `3>7 →` falso `(false)` `5!=8 →` cierto `(true)`

2.2.- Instrucción de asignación

2.2.1.- Sintaxis: <variable> = <expresión>

2.2.2.- Semántica: Se evalúa la expresión <expresión> y el resultado se asigna (se almacena) en la variable <variable>. El valor anterior que pudiera haber en dicha variable se pierde (ya que se sobrescribe encima). La expresión puede ser una constante entera o cualquier combinación de constantes enteras y/o variables enteras combinada con los operadores aritméticos de los números enteros (+, -, *, %). Se pueden poner paréntesis.

Ejemplo:

```
int a, b, c, d;
a = 5; /* En la variable a se almacena el valor 5 */
b = a; /* En la variable b se almacena el valor que hay en la variable a */
c = a + b*5; /* En la variable c se almacena el valor resultado de evaluar
                la expresión a + b*5 */
d = c - a - b; /* En la variable d se almacena el valor resultado de evaluar
                la expresión c - a - b */
```

2.2.3.- Evaluación de expresiones: prioridad entre operadores

Ejercicio: ¿Cuál es el valor de la expresión a + b * 5? (siendo a=5 y b=5)

Si primero calculáramos a + b y luego se le multiplicara por 5, el resultado sería 50

a + b * 5 → (a + b) * 5 → (5 + 5) * 5 → 10 * 5 → 50

Pero, si primero se multiplicara b por 5 y luego se le sumara a, el resultado sería 30

a + b * 5 → a + (b * 5) → 5 + (5 * 5) → 5 + 25 → 30

Por lo tanto, hay que conocer exactamente cuando se escribe una expresión, en qué orden se va a evaluar: qué operaciones se van a ejecutar antes que otras. Existe una prioridad entre los operadores que es la siguiente (de mayor a menor)

( )		
*	/	%
+		–

Que quiere decir que primero se evalúa lo que está entre paréntesis, después las multiplicaciones, divisiones enteras y restos de divisiones enteros con igual prioridad, y por último, las sumas y restas con igual prioridad.

Ejercicio: ¿Cuánto vale c – a – b ? (siendo c=30, a=5 y b= 5)

Dependiendo de qué resta se haga antes `(c-a)` o `(a-b)`, el resultado puede ser uno u otro.

(c – a) – b → (30 – 5) – 5 → 25 – 5 → 20
c – (a – b) → 30 – (5 – 5) →30 – 0 → 30

Y además, hay que saber que cuando hay operadores de igual prioridad entonces se evalúa de izquierda a derecha.

Ejercicio. Dada la siguiente secuencia de instrucciones, ¿cuánto vale g?

```
int e, f, g;
e=6;
f=7;
g = e * f / 2 + 5 - (e - f);
```

2.2.4.- Errores posibles en instrucciones de asignación

1.- Alguna de las variables que aparecen en la expresión ESTÁN SIN INICIALIZAR

```
int e, f, g;
e=6;
g = e * f / 2 + 5 - (e - f); // Error ya que f NO SE HA INICIALIZADO
```

2.- El tipo de la variable y de la expresión NO COINCIDEN.

```
int e, f, g;
f=7.45;      // Error ya que el tipo de f (int) no es un número real
```

2.3.- Bloque de instrucciones.

Toda instrucción debe terminar con el carácter ; (punto y coma)

Un bloque de instrucciones puede estar formado por una única instrucción, o bien puede estar formado por una serie de instrucciones encerradas entre llaves **{ }. Todas las instrucciones deben terminar en ;**

Un bloque de instrucciones se puede considerar como una instrucción formada por varias que se ejecutan una detrás de otra.

Ejemplos:

```
a = b + 5 * c;
/* Es un bloque de instrucciones formado por una sola instrucción */
```

```
{ a = b + 5;
  a = a * c; }
/* Es un bloque de instrucciones formado por 2 instrucciones */
```

2.4.- Entrada/Salida de enteros.

Las instrucciones de Entrada/Salida son necesarias para que el programa interactúe con el usuario y así trabaje con distintos datos de entrada y produzca distintos resultados de salida, dependiendo de dichos datos de entrada. Recordamos aquí la figura que aparecía en el primer tema, y que intenta mostrar de manera gráfica para qué sirven los programas.

En Java existen muchas formas de obtener datos (entrada) y de mostrar resultados (salida). Algunas de ellas son bastante potentes y sofisticadas, pero, por el contrario, más difíciles de utilizar.

Comenzaremos utilizando una forma sencilla de realizar Entrada / Salida, que va a consistir en:

1) Crear lo necesario para poder realizar operaciones de entrada/salida
2) Preguntar datos de entrada al usuario
3) Mostrar resultados de salida al usuario

1) Crear lo necesario para poder realizar operaciones de entrada/salida

Será suficiente con crear un "objeto" al cual se le puede pedir que haga operaciones de entrada y salida de datos. Se hace de esta manera:

1) Definiendo al comienzo del programa lo siguiente:

```
import acm.io.*; // Para importar todas las clases del paquete acm.io
```

2) Y creando dicho "objeto" que se almacena en una variable (que en este caso llamaremos `entradaSalida`, pero que podría tener como nombre cualquier identificador), para poder utilizarlo más adelante, esto es, para poder pedirle que realice operaciones de entrada y salida de datos.

```
IODialog entradaSalida = new IODialog();
```

A continuación tenéis una explicación un poco más detallada de este paso, que en una primera lectura se puede obviar.

Ese "objeto", en nuestro caso, va a ser un objeto de una clase ya definida llamada `IODialog`, que se encuentra en un paquete llamado `acm.io`. El nombre completo de la clase está formado por el nombre del paquete seguido del nombre de la clase: `acm.io.IODialog`

Lo siguiente crea un objeto de dicha clase: `new acm.io.IODialog()`

La creación del objeto y almacenamiento en la variable `entradaSalida` se hace con esta instrucción:

```
acm.io.IODialog entradaSalida = new acm.io.IODialog();
```

NOTA: el "`acm.io.IODialog`" a la izquierda de `entradaSalida` indica que en la variable se pueden almacenar objetos de la clase `acm.io.IODialog`

Otra manera más sencilla de llamar a las clases es usando solamente su nombre, y no añadiendo el nombre del paquete. La creación y almacenamiento anterior se harían simplemente así:

```
IODialog entradaSalida = new IODialog();
```

Para ello, hay que indicar que se va a trabajar con esa clase (que se va a "importar" esa clase), utilizando una sentencia **`import`**

```
import acm.io.IODialog;
// Para poder referirse a dicha clase como IODialog
```

O bien:

```
import acm.io.*;
// Para importar todas las clases del paquete acm.io
```

Veremos más adelante qué son los objetos y las clases, por lo que no os preocupéis ahora de la sintaxis.

Más info sobre el paquete acm.io en: http://jtf.acm.org/rationale/io-package.html

2) Preguntar datos de entrada al usuario

Al objeto creado anteriormente (almacenado en la variable `entradaSalida`) le podemos pedir que obtenga algún dato de entrada del usuario. Para que el usuario sepa qué dato le están pidiendo, se le puede mostrar un mensaje que se lo indique (en este caso: "Dime tu edad:").

Esto se hace así: `entradaSalida.readInt("Dime tu edad: ");`

El usuario podrá escribir un número, como por ejemplo el 25:

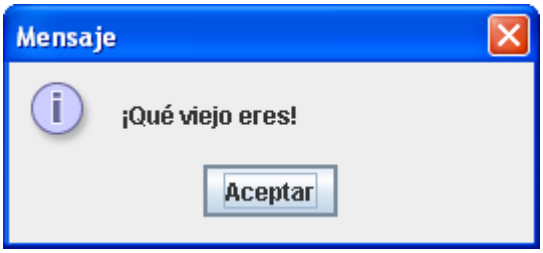
Y ese valor, habrá que almacenarlo en alguna variable, para poder ser utilizado más adelante. Por ejemplo en una variable llamada `edad` de tipo entero (`int`).

Todo esto se realiza de la siguiente manera:

```
int edad = entradaSalida.readInt("Dime tu edad: ");
```

3) Mostrar resultados de salida al usuario

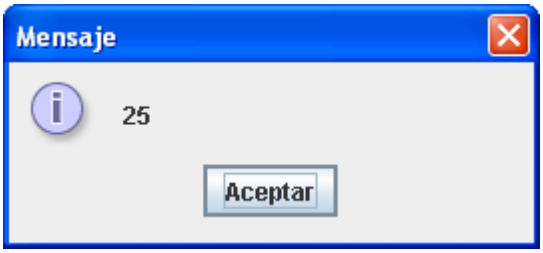
Al objeto creado anteriormente (almacenado en la variable `entradaSalida`) también se le puede pedir que muestre resultados, los cuales pueden ser textos como el siguiente:



Se haría de la siguiente manera, donde el texto a mostrar se pone entre comillas:

```
entradaSalida.println("¡Qué viejo eres!");
```

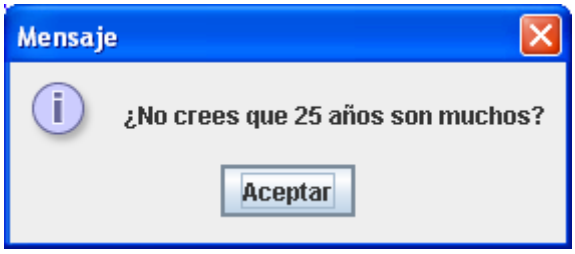
Pero también se puede mostrar el contenido de variables:



Se haría de la siguiente manera, indicando el nombre de la variable cuyo contenido se quiere mostrar:

```
entradaSalida.println(edad);
```

Y se puede mostrar una mezcla de texto y contenido de variables como la siguiente:

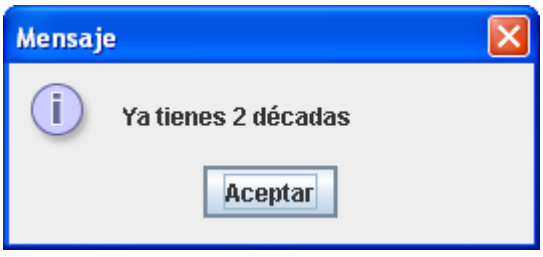



```
entradaSalida.println("¿No crees que "+edad+" años son muchos?");
```

donde el operador + sirve en este caso para concatenar textos, contenidos de variables, etc.

Por último, hay que tener en cuenta que los resultados de salida pueden requerir realizar un cierto procesamiento con los datos de entrada. Para ello, podemos utilizar los operadores aritméticos explicados anteriormente, y hacer uso de variables para almacenar los resultados que se deseen.

Si conocida la edad quisiéramos indicar al usuario cuántas décadas tiene:



Podríamos conseguirlo de esta manera:

```
int decadas = edad / 10;
entradaSalida.println("Ya tienes "+decadas+" décadas");
```

2.6.- Otra forma sencilla de realizar salida de datos: System.out

Para mostrar resultados de una manera más sencilla, sin necesidad de usar un objeto de la clase `IODialog`, se puede hacer sobre otro objeto, que es el `System.out`, que los muestra en la consola.

Todas las escrituras anteriores realizadas sobre el objeto almacenado en `entradaSalida`

```
entradaSalida.println("¡Qué viejo eres!");
entradaSalida.println(edad);
entradaSalida.println("¿No crees que "+edad+" años son muchos?");
entradaSalida.println("Ya tienes "+decadas+" décadas");
```

Podrían haberse hecho directamente sobre la consola.

```
System.out.println("¡Qué viejo eres!");
System.out.println(edad);
System.out.println("¿No crees que "+edad+" años son muchos?");
System.out.println("Ya tienes "+decadas+" décadas");
```

NOTA: Si usamos `System.out.print` no salta de línea tras escribir el texto correspondiente.

2.7.- Primer programa en Java

El siguiente programa lee un número entero y escribe el número siguiente al leído.

```
import acm.io.IODialog;

public class PrimerPrograma {

	public static void main(String[] args) {

		IODialog entradaSalida = new IODialog();
		int edad = entradaSalida.readInt("Dime tu edad: ");
		int edadSiguiente = edad + 1;
		entradaSalida.println("El año que viene tendrás "
				+edadSiguiente+" años");
	}
}
```

Un programa Java está formado por al menos una clase (en este caso llamada `PrimerPrograma`), que, al ejecutarse comienza a ejecutar las instrucciones que hay en el bloque de instrucciones del método principal `main`. De momento, no explicaremos exactamente qué es eso. Lo dejaremos para el tema de clases, métodos y objetos.

```
	public static void main(String[] args) {



	}
```

Dentro de ese método principal (`main`) hay que definir las variables que se necesiten y las instrucciones que se desee ejecutar.