Ten en cuenta que: en este tema se habla mucho de cómo se almacenan los datos internamente, es decir, a nivel de bit. Es un tema complejo y que aquí está muy resumido, leyendo esto no puedes conseguir una comprensión absoluta de la representación interna de los datos. El objetivo que tienes que marcarte es saber hacer y entender lo mejor posible los ejercicios del final.

SISTEMA DE NUMERACIÓN

Es un concepto necesario para trabajar con este boletín. Lo veréis o visteis en profundidad en otro módulo. Puedes echarle un vistazo rápido en cualquier página web por ejemplo:

http://platea.pntic.mec.es/~lgonzale/tic/binarios/numeracion.html

LOS TIPOS

En java que "una cosa" tenga un ***tipo*** indica :

- como se representa en memoria, esto es, con cuantos bits y en base a qué código (complemento a 2, unicode,punto flotante …)
- qué operaciones se pueden hacer con ese tipo

Los componentes del lenguaje java que tienen tipo son:

- Las variables. Recuerda que se indica su tipo al declararlas
  Ej: char c;
  c es una variable de tipo char
- los literales. Un literal es un valor constante que aparece en el código, su tipo se deduce de como está escrito.
  Eje: 8.5 es un literal double. También se podría escribir 8.5d y también 8.5D.
  Eje: 8.5f es un literal float

sobre el término "constante": ojo, en programación, la palabra "constante" según el contexto se puede a referir a una variable constante o a un valor constante (literal). En java a los valores constante se les prefiere llamar siempre literales, dejando el término "constante" para las variables constantes.

- las expresiones. Una expresión se evalúa (se calcula) y el resultado de la evaluación es un valor, ese valor tendrá un tipo como los literales.
  Eje: 7 + 1.1 se evalúa y se obtiene 8.1 que tiene tipo double

LOS TIPOS EN JAVA

Sin ningún rigor, pero para poder empezar a explicar los tipos en java, vamos a decir que en Java hay dos grandes clases de tipos:

- Los tipos primitivos
- Los tipos objeto

Por el momento veremos sólo los tipos primitivos

RESUMEN TIPOS DE DATOS PRIMITIVOS

Dato	Tipo	Bits	Rango
carácter	**char**	16	0 a 65535
entero	**byte**	8	-128 a 127
	short	16	-32768 a 32767
	int	32	-2147483648 a 2147483647
	long	64	-9223372036854775808 a 9223372036854775807
real	**float**	32	$-3.4x10^{38}$ a $-1.4x10^{-45}$, $1.4x10^{-45}$ a $3.4x10^{38}$
	double	64	$-1.7x10^{308}$ a $-4.9x10^{-324}$, $4.9x10^{-324}$ a $1.7x10^{308}$
booleano	**boolean**	1	true, false

LAS VARIABLES DE TIPO ENTERO

Los números matemáticos *enteros* se pueden representar en java con más o menos bits. Disponemos en java de los siguientes tipos para valores enteros:

- byte(8 bits)
- short(16 bits)
- int(32 bits)
- long(64 bits)

Al declarar en java una variable, debemos indicar su tipo, si lo que queremos es almacenar un valor entero debemos escoger entre uno los tipos anteriores.
Ejemplo: Ejecuta el siguiente programa

```
public class App {
    public static void main(String[] args) {
      byte midato1 = 1;
      short midato2 = 100;
      int midato3 = 10000;
      long midato4 = 100000000;

      System.out .println("la variable byte vale:  " + midato1);
      System.out .println("la variable short vale:  " + midato2);
      System.out .println("la variable int  vale:  " + midato3);
      System.out .println("la variable long vale:  " + midato4);

    }
}
```

LOS LITERALES DE TIPO ENTERO

Los literales enteros sólo pueden ser de tipo int o long
Ejemplo: Ejecuta el siguiente programa

```
public class App {
    public static void main(String[] args) {
        long i1 = 5L;// 5L es un literal long
        long i2 = 5l;//5l literal long pero l minúscula se confunde con 1 mejor usar L
        int i3 = 5;//literal int
        short s;
        byte b;
        s=3; //3 es un literal int no existen literales short
        b=3; //3 es un literal int no existen literales byte
    }

}
```

Puede resultar chocante que funcione la siguiente instrucción
s=3;
ya que 3 se almacena como un int de 32 bits, ¿Como es posible que "quepa" en un short de 16 bits?
Esto es debido a que el compilador hace un cast automático y la instrucción anterior sería equivalente a
s=(short) 3;
que lo que hace es quedarse con los 16 primeros bits y despreciar los 16 superiores
más sobre cast en otro boletín ...

Por lo tanto: se puede asignar a una variable byte y short un literal entero, siempre cuando dicho valor entero esté en el rango del tipo:

```
byte b;
b=3; //bien
b=127; //bien
b=128; //mal
b=1328; //mal
```

¡Comprueba esto compilando!

Diversas formas de escribir un literal entero (int y long) en el código java

Lo habitual es escribirlos en base 10, pero es también posible escribirlos en base 8 (octal), 16 (hexadecimal) y binario.
Ejemplo: Ejecuta este código

```
class App {
    public static void main(String[] args) {
        System.out.println("15 en decimal: "+15);
        System.out.println("15 en octal: "+017);
        System.out.println("15 en hexadecimal: "+0xF);
        System.out.println("15 en binario: "+0B1111); //se puede usar b minúscula
    }
}
```

Tienes que tener claro que en el ejemplo anterior los bits de los literales anteriores son exactamente los mismos ya que estamos escribiendo a alto nivel el mismo valor pero de distintas formas matemáticas. Internamente sólo se maneja base 2.
Por otro lado, a println() le pasamos el entero 15 escrito en distintas bases, pero él siempre lo imprime en base 10.

Guión bajo con enteros

Para mejorar la legibilidad de los números enteros muy largos se puede usar el underscore (guión bajo), no influye en absoluto en el valor del número.

```
class App {
    public static void main(String[] args) {
        int tresMillones= 3_000_000;
        System.out.println("valor de tresmillones es: "+tresmillones);

    }

}
```

Representación en bits de los números enteros

Echa un vistazo rápido a este vídeo para ver que el sistema que se utiliza para representar un número entero. Hazte una idea de por dónde van los tiros. No es necesario entenderlo al 100%

https://www.youtube.com/watch?v=B7SpmkW0lTs&list=TLPQMDExMDIwMjI4nL0vHJjZwQ&index=3

LOS TIPOS REALES.

https://www.youtube.com/watch?v=HcjXH9WGmAU

Este sistema tiene un problema de precisión inevitable, no es un problema de java, lo tienen todos los lenguajes ya que la precisión está condicionada por las instrucciones máquina del procesador, no del lenguaje de alto nivel. En java podemos representar un número real con dos tipos:

- float => utiliza 32 de bits (para representar signo, exponente y parte no exponencial)
- double => utiliza 64 bits.

En java, todos los literales de coma flotante son del tipo double salvo que se especifique lo contrario, por eso si se intenta asignar un literal en coma flotante a una variable de tipo float el compilador nos dará un error (tipos incompatibles):

Ejecuta el siguiente programa

```
public class App {
    public static void main(String[] args) {
      float valor;
      //la siguiente instrucción da error si la descomentas
      //valor = 2.6;
      //la siguiente instrucción es correcta
      valor = 2.6f;
      System.out .println("Valor del dato= " + valor);
    }
}
```

Por tanto, para indicar que el valor 2.6 se almacene como un tipo float tenemos que añadir una f al final 2.6f

Con double puedo representar números en un rango mayor y con más decimales

Ejemplo: comprobamos que con double hay más precisión
Sabemos que ⅓ es un valor infinito pero el ordenador tiene que tener una representación interna con un número finito de bits.

```
D:\programacion>java App
1/3 con valores float  0.33333334
1/3 con valores double  0.3333333333333333
```

Te puede sorprender que el float contenga al final un 4 y no así el double. La representación interna (el número en bits y como se manejan) de float y double es compleja y mayores profundizaciones se sale de nuestro objetivo. Lo que tienes que "sobreentender" es que hay infinitos números reales y es imposible representarlos todos pues disponemos de un número finito de bits para la representación. Ya que hay infinitos números que representar, para la mayor parte de ellos realmente estaremos trabajando con representaciones aproximadas.
Es fácil entender que el resultado de 1.0/3.0 ya que es un número infinito es imposible representarlo internamente de forma exacta y se trabaja por tanto con un valor aproximado. Observa en el ejemplo anterior que cuando trabajamos con double la aproximación es mejor.

Lo que es más difícil de entender es que valores como 0.1 en realidad no se pueden representar exactamente, de hecho, 0.1 se representa internamente con un conjunto de bits que se corresponden con el siguiente valor decimal
0.1000000000000000055511151231257827021181583404541015625.
De nuevo recuerda que hay infinitos números reales y que hay que representar con un número finito de bits. La solución es guardar muchos de ellos de forma aproximada

¿Es posible almacenar un valor entero en una variable float o double?. Sí, pero esto tiene como como inconvenientes:

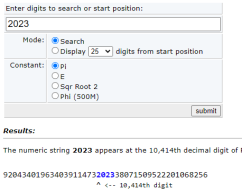
- gasto innecesario de memoria
- al hacer operaciones aritméticas, posible pérdida de precisión en el resultado

Encontrar tu dni en el número PI.
Como sabes PI es un número infinito, así que cuando lo usamos desde los lenguajes de programación realmente usamos aproximaciones del número PI.

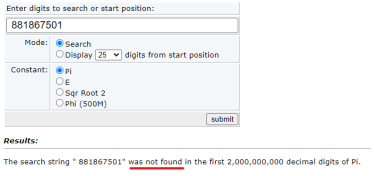
Además PI es un número irracional, sus decimales no se repiten con una secuencia fija, y así, dentro de sus infinitos decimales podremos encontrar cualquier número, por ejemplo, nuestro número de teléfono o nuestro dni. En la siguiente página(entre otras) podemos comprobar si aparecen estos números en PI y cuando aparecen.

https://matematicascercanas.com/2016/03/14/buscar-numero-decimales-%CF%80/

Pero, ojo, la página, lógicamente, no puede manejar infinitos decimales, sólo es capaz de mirar en los primeros dos mil millones de decimales(casi nada!). Por ejemplo el número 2023 aparece "pronto"



Pero por ejemplo el teléfono del sanclemente no está en los primeros dos mil millones de decimales



Mi DNI y mi teléfono sí aparecen ¿Y los tuyos?

Representación en bits de los números reales
Los números reales se representan internamente con el sistema de "punto flotante" (coma flotante en castellano). Este standard tiene que ver con las instrucciones máquina del ordenador. No depende ni del lenguaje de programación ni del sistema operativo. Por ejemplo los PC lo usan, así que windows y linux tienen que usarlo y por extensión todos los lenguajes de programación que uso en un PC.

Echa un vistazo rápido a este vídeo para ver que el sistema que se utiliza para representar un número real tiene su complejidad. Hazte una idea de por dónde van los tiros. No es necesario entenderlo al 100%

https://www.youtube.com/watch?v=HcjXH9WGmAU&list=TLPQMDExMDIwMjI4nL0vHJjZwQ&index=1

EL TIPO CHAR Y LOS CÓDIGOS DE CARACTERES.

Los códigos de caracteres

Internamente los caracteres se representan en base a unas tablas que llamamos códigos. En estas tablas se asigna a cada carácter una combinación de bits.
Java utiliza, generalizando y simplificando mucho, el código UNICODE

El código ascii

El código ascii es "un antepasado" de Unicode. Es un código muy antiguo (1964), que tuvo mucha repercusión en computación. Todavía hoy en día se hacen referencias constantes al código ASCII
En este video puedes obtener un explicación asequible de lo que pretende un código de caracteres, en el ejemplo explicado con ASCII

Como ejercicio, buscamos una página web que me contenga una tabla ASCII, por ejemplo http://es.wikipedia.org/wiki/ASCII y localizamos la *n* en la tabla ascii

En este video explica de forma sencilla cómo se traduce texto a binario
https://www.youtube.com/watch?v=uKpgIL-RzVE

El codigo unicode

EL código ASCII es muy sencillo, con muy pocos caracteres, pensado para ordenadores de hace más de 40 años. Actualmente este código está incluido en el propio UNICODE de forma que a los primeros 128 caracteres de unicode se le siguen llamando "los caracteres ASCII"

http://www.rapidtables.com/code/text/unicode-characters.htm

Ejercicio: comprueba que la *n* tiene el mismo valor decimal que en una tabla UNICODE que en una tabla ASCII

El tipo char en java

Para asignar un carácter a una variable char puedo utilizar 3 métodos:

- el carácter entre comillas simples, ej. letra_n='n'
- el valor unicode en decimal , ej. letra_n= 110.
  Hay un cast automático (más adelante veremos qué es *cast*) y por tanto es equivalente a letra_n=(char) 110
- el valor unicode (\u seguido de 4 dígitos hexadecimales) , ej. letra_n='\u006E'. Si pasas 6e a decimal obtendrás 110

```
class App{
   public static void main(String[] args){
      char letra_n;

      letra_n='n';
      System.out.println("El valor de la variable letra_n  es "+letra_n);
```

```
      letra_n=110;
      System.out.println("El valor de la variable letra_n es "+letra_n);
      letra_n='\u006E';
      System.out.println("El valor de la variable letra_n es "+letra_n);
   }
}
```

Ya que los caracteres unicode son códigos de 16 bits, el mayor literal entero que se puede usar para asignar a un char se corresponde a 2^{16}-1 o sea 65536-1 es decir 65535
En el siguiente ejemplo observa que si descomentamos la última instrucción compila con error

```
class App{
   public static void main(String[] args){
      char c;
      c=90;
      System.out.println("El entero 90 es el carácter "+c);
      c=97;
      System.out.println("El entero 97 es el carácter "+c);
      c=225;
      System.out.println("El entero 225 es el carácter "+c);
      c=65535;
      System.out.println("El entero 65535 es el carácter "+c);

      //c=65536;
   }
}
```

ESPECIFICAR ALGUNOS CARACTERES ESPECIALES: SECUENCIAS DE ESCAPE

Idea intuitiva de secuencia de escape.

```
class App{
   public static void main(String[] args){
      System.out.println("-Papá, ¿somos amigos no?");
      System.out.println("-Si");
      System.out.println("-Y siempre estaremos juntos, ¿verdad?");
      System.out.print("-Simba, te voy a contar algo que un día me dijo mi padre. ");
      System.out.println("Mira las estrellas. Los grandes reyes del pasado nos observan desde esas estrellas.");
      System.out.println("-¿De veras?");
      System.out.println("-Si. Y cuando te sientas solo, recuerda que esos reyes estarán ahí para guiarte. Y yo también.");
   }
}
```

Podemos eliminar, si nos place, instrucciones println() si podemos de alguna forma indicar que queremos cambiar de línea. Para eso introducimos en el texto el caracter especial \n para indicar donde queremos comenzar una nueva línea, por ejemplo, simplificamos las tres primeras instrucciones en una:

```
class App{
   public static void main(String[] args){
      System.out.println("-Papá, ¿somos amigos no?\n-Si\n-Y siempre estaremos juntos, ¿verdad?");
      System.out.print("-Simba, te voy a contar algo que un día me dijo mi padre. ");
      System.out.println("Mira las estrellas. Los grandes reyes del pasado nos observan desde esas estrellas.");
      System.out.println("-¿De veras?");
      System.out.println("-Si. Y cuando te sientas solo, recuerda que esos reyes estarán ahí para guiarte. Y yo también.");
   }
}
```

Entonces, una secuencia de escape se forma con \caracter y no se tratan como texto normal si no que tienen un significado especial.

Secuencias de escape más usadas y su significado

Secuencia de escape	Significado
\b	Retroceso
\n	Salto de línea
\t	Tabulación horizontal
\\	Barra invertida \
\'	Comilla simple
\"	Comilla doble

Hay combinaciones \caracter que no son secuencia de escape, es decir, detrás de la \ sólo pueden ir ciertos caracteres.
El compilador reconoce una secuencia de escape porque comienza por una barra diagonal invertida. Cuando el compilador se encuentra con una \ sabe que el carácter que viene a continuación "escapa" de su significado normal y toma un significado especial. Hay dos situaciones en las que se usan:

- Para trabajar con un carácter no imprimible, por ejemplo un tabulador
  ch = '\t' ;
  observa que con la barra \ precediendo al carácter *t*, la *t* no se refiere al código ascii asociado a la letra *t* minúscula, si no a el carácter no imprimible *tabulador*
- Para poder almacenar caracteres que ya tienen de por sí un uso especial, por ejemplo:
  ch = '\\'
  para poder almacenar el carácter \ lo precedo del indicado carácter de secuencia de escape que también es una \
  Otro ejemplo: Almacenar en una variable la comilla simple
  ch = '\''
  así almaceno en ch una comilla simple,
  Si hubiera escrito:
  ch = ''' el compilador interpreta la segunda comilla como una comilla de cierre, y la tercera como una de apertura sin su correspondiente de cierre y por tanto habrá un error de compilación.

Ejemplo con '\t'

```
class App{
   public static void main(String[] args){
      System.out.print('A');
      System.out.print('\t');
      System.out.print('B');
      System.out.print('\t');
      System.out.print('C');
      System.out.print('\t');
       System.out.print('D');
      System.out.print('\t');


   }
}
```

Ejemplo: imprimir la barra \ y la comilla

```
class App{
   public static void main(String[] args){
      System.out.print('\\');
      //System.out.print('\');
      System.out.print('\'');
   }
}
```

CADENAS DE CARACTERES

El término *String* se traduce por *cadena*. Ambos términos se utilizan profusamente.
String es realmente un objeto no un tipo primitivo, pero se usan inevitablemente desde el principio y adelantamos una mini explicación.
Una cadena de caracteres es un conjunto de caracteres encerrados entre comillas dobles.
Observa que 'a' y "a" son cosas distintas. Observa que ‘a’ es un char, pero “a” es una cadena de caracteres

Ejemplo de variables string.

```
class App{
    public static void main(String[] args){
        String saludo="hola";
        System.out.println(saludo);
        System.out.println(saludo+" mundo" );
    }
}
```

El operador + y Strings

El operador + con números suma y con string concatena las cadenas de caracteres (pega una con otra). Si mezclamos cadenas y números pueden ocurrir varias cosas pero por el momento para simplificar asumimos que siempre concatena.
Las secuencias de escape también se usan tanto en literales char como en literales String
Ejemplo:

```
class App{
    public static void main(String[] args){
        System.out.println("Primera línea \n Segunda linea");
        System.out.println("A\tB\tC");
        System.out.println("D\tE\tF");
        System.out.println("la barra \\ sirve para representar secuencias de escape ");
    }
}
```

EL TIPO BOOLEAN

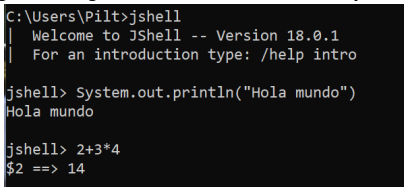
dos valores posibles true y false

```
class App{
    public static void main(String[] args){
        boolean b;
        b=false;
        System.out.println(" b es " +b);
        System.out.println("hola mundo");
        b=true;
        System.out.println(" b es " +b);
    }
}
```

Observa que el método println() “entiende" todos los tipos de datos primitivos (int, float,char, boolean, ...)

SOBRE jshell

Es una utilidad de consola para ejecutar bloques de código cortos sin necesidad de usar javac/java. Lo vemos aquí como curiosidad pero su uso es prescindible.



Se pueden declarar variables y muchas más cosas pero realmente sólo es cómodo para cosas muy breves como ver el valor que generan expresiones.

```
jshell> int i=0
i ==> 0

jshell>  i=4
i ==> 4

jshell> i*2
$8 ==> 8

jshell> /exit
|  Goodbye

C:\Users\Pilt>
```

Calculadoras para programadores

Utilizar una calculadora en el modo de programación puede ser útil para hacernos una idea de cómo se almacenan internamente los números enteros en bits y para razonar el cast que estudiaremos en el siguiente boletín. Podemos utilizar por ejemplo la calculadora de windows o la siguiente de chrome:
https://chrome.google.com/webstore/detail/programmers-calculator/pgkgdlpegifkoofoioopnbkkfhjociaj

EJERCICIO U1_B3_E1.

Cada tipo primitivo tiene asociada una clase Java. Aún no vimos lo que es una clase, pero para simplificar digamos por el momento que estas clases Java empaquetan para cada tipo primitivo un conjunto de utilidades. A estas clases se les conoce como Wrapper Classes (clases envoltorio) y a lo largo del curso entenderás el porqué de este nombre.

Wrapper Classes for Primitive Data Types

Primitive Data Types	Wrapper Classes
int	Integer
short	Short
long	Long
byte	Byte
float	Float
double	Double
char	Character
boolean	Boolean

Todas las clases excepto Boolean dan acceso al máximo y mínimo valor posible de cada tipo a través de las constantes MAX_VALUE y MIN_VALUE. Por ejemplo:

```
jshell> Integer.MAX_VALUE
$5 ==> 2147483647
```

Se pide que escribas un programa que genere la siguiente tabla

```
Tipo    Min.            Max.
int     -2147483648     2147483647
byte    -128            127
short   -32768          32767
float   1.4E-45         3.4028235E38
double  4.9E-324        1.79769313486231570E308
```

Completando el siguiente código

```
class App{
   public static void main(String[] args){
      System.out.println("Tipo\tMin.\t\tMax.");
      System.out.println("int\t"+Integer.MIN_VALUE+"\t"+Integer.MAX_VALUE);
      ETC.
```

Observa que excluimos de la tabla a Boolean que no tiene las constantes indicadas y a Character porque hasta el boletín que viene en el que estudiamos el operador de cast no entenderíamos bien la salida ya que los valores están asociados a caracteres no imprimibles. Para alinear la tercera columna según el largo de la segunda a veces tendrás que usar un \t y a veces \t\t.

EJERCICIO U1_B3_E2. En el jshell crea una variable int i y luego asígnale el valor 08. Razona el error.

```
jshell> int i
i ==> 0

jshell> i=0123
i ==> 83

jshell> i=08
|  Error:
|  ';' expected
|  i=08
|     ^
```

EJERCICIO U1_B3_E3. Utiliza calculadora para observar el valor de 17 en decimal/octal/hexadecimal/binario

EJERCICIO U1_B3_E4.
Explica porqué el siguiente código da error de compilación

```
class App{
   public static void main(String[] args) {
      int miVar=2147483648;
   }
}
```

EJERCICIO U1_B3_E5.
Explica porque la primera declaración de variable no da error de compilación y la segunda sí

```
class App{
   public static void main(String[] args) {
      int miInt=128;
      byte miByte=128;
   }
}
```

EJERCICIO U1_B3_E6.
Explica porque la primera declaración da error de compilación y la segunda no

```
class App{
   public static void main(String[] args) {
      float miFloat=10.0;
      double miDouble=10.0;
   }
}
```

EJERCICIO U1_B3_E7.

Explica porque hay error de compilación en

```
class App{
   public static void main(String[] args){
      System.out.println("Esto no es una secuencia de escape \p y da error");
   }
}
```

EJERCICIO U1_B3_E8.

Explica porqué no se imprime en la salida el carácter \ y el carácter n

```
class App{
   public static void main(String[] args){
      System.out.println("si pongo \n no se imprime la barra y la n");
   }
}
```

EJERCICIO U1_B3_E9.

Escribe un println que genere la siguiente frase por pantalla:

Me gusta mucho la barra \ y las comillas "

EJERCICIO U1_B3_E10.

Consulta una tabla Unicode (o ASCII) y consigue el mismo resultado que este código pero utilizando el código decimal del carácter A en lugar del literal char 'A'

```
class App{
   public static void main(String[] args){
      char miLetra_A='A';
      System.out.println(miLetra_A);
   }
}
```

EJERCICIO U1_B3_E11.

Escribe un programa que almacene en variables y luego imprima por pantalla, 4 características de una de una persona: Su nombre completo, su sexo, su edad y su peso. Elige para cada variable el tipo de dato más económico en bits. Obtén una salida similar a la siguiente:

```
D:\programacion>java App
nombre: Juán López Salvado
sexo: v
edad: 49
peso: 70.8
```

EJERCICIO U1_B3_E12.

Indica que tipo de literal es cada uno de los siguientes

178

2L

077L

0xBAACL

37.266D

87.363F

'c'
'\t'
true
false
'\u00E1'
"hola"