

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Clase String. Introducción a los flujos

© ADR Infor SL

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad	3
Clase String. Introducción a los flujos.	4
Objetivos.	4
Clase String.	4
Clase asociada	7
Constructores	7
Métodos principales	8
Introducción a los flujos (streams)	10
Captura de datos desde el teclado	10
Ejercicios	13
Ejercicio 1. Clase String.	13
Recomendaciones.	13
Datos a mostrar por consola.	13
Ejercicio 2. Contar letras.	13
Datos a mostrar por consola.	14
Ejercicio 3. Contar letras II.	14
Datos a mostrar por consola.	14
Ejercicio 4. Comprobar NIF. (Ejercicio obligatorio)	14
Pistas.	15
Recomendaciones.	15
Lo necesario para comenzar.	15
Datos a mostrar por consola	16
Ejercicio 5. Volúmenes.	17
Recomendaciones.	17
Pistas.	17
Lo necesario para comenzar.	17
Datos a mostrar por consola.	18
Ejercicio 6. Adivinar número.	18
Recomendaciones.	19
Datos a mostrar por consola.	19
Recursos	20
Enlaces de Interés	20
Glosario.	20

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:

Manejar correctamente la clase String dentro de un programa Java.

Resultados de Aprendizaje:

- Conocer la clase `java.lang.String` perteneciente a Java.
- Crear y utilizar la clase `java.lang.String`
- Usos más comunes de la clase `java.lang.String` dentro de un programa Java
- Utilizar los métodos más comunes del API `java.lang.String`.

Clase String. Introducción a los flujos.

Objetivos.

- Conocer la clase `java.lang.String` perteneciente a Java.
- Aprender a crear y utilizar una variable de tipo `String`.
- Entender sus usos más normales.
- Conocer los métodos más comunes del API `java.lang.String`.

Clase String.

Vocabulario: String

Un `String` es una **variable referenciada asociada a un objeto** de la clase `java.lang.String`.

Se emplea para almacenar cadenas de caracteres y son ampliamente utilizadas en la programación en Java.

Los `Strings` tienen una característica que los diferencia del resto de objetos: **son inmutables**, es decir, **cuando se intenta modificarlas**, por ejemplo al aplicarles un método, no se modifican sino que **se crea otra `String` nueva**.

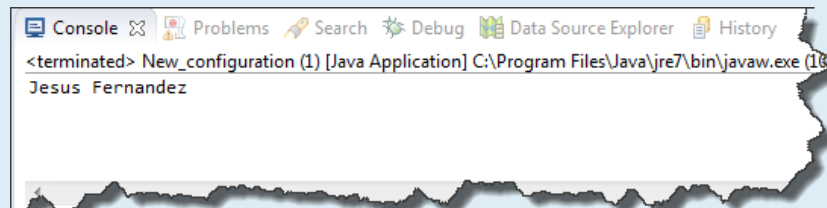
Inmutabilidad 1

Se observa que siendo un String una **variable referenciada** no se emplea constructor para crearla sino que se siguen las mismas pautas que en la variables primitivas. Cuando se vean los constructores de String se explicará el motivo.

```
/* Creación de un objeto String al que apunta la variable
 * referenciada s
 */
String s = "Jesus";

/* Parece que con la siguiente sentencia se modifica la String s
 * añadiéndole "Fernandez" pero, en realidad, no es así. Ocurren
 * dos cosas:
 *   · Se crea un nuevo String cuyo contenido es "Jesus Fernandez".
 *   · La variable s deja de apuntar a la String cuyo contenido es
 *     "Jesus", con lo cual dicha String se desreferencia. La variable s
 *     pasa a apuntar al nuevo String de contenido "Jesus Fernandez".
 */
s = "Jesus Fernandez";

/* Mostrar por consola */
System.out.println(s);
```



Los objetos desreferenciados son candidatos a ser eliminados de la memoria mediante un recolector de basura o **garbage collector**, que trabaja en un segundo plano de forma transparente para el programador. Esto hace que la gestión dinámica de la memoria no sea una preocupación de primer nivel para un programador Java y que pueda focalizar toda su atención en desarrollar buenos códigos.

Inmutabilidad 2

```

class InmutabilidadStrings {
    public static void main(String args[]){
//String a la que apunta la variable referenciada s.
String s="Jesus";

//Al aplicarle el método concat(..), se crea otra String,
//que contiene Jesus Fernandez, sin modificar la original.
//Ocurre, que la nueva String está desreferenciada, es decir,
//no hay ninguna variable que apunte a la misma.
s.concat(" Fernandez");

//Por consola: Jesus
System.out.println(s);

/*
 * La clase StringBuffer se estudiará más adelante. Basta saber que
 * se emplea para almacenar cadenas de texto que cambian con mucha
 * frecuencia. Se introduce para comprobar que sólo las Strings
 * son inmutables, no el resto de objetos.
 */

//StringBuffer a la que apunta la variable referenciada sb.
StringBuffer sb=new StringBuffer("Jesus");

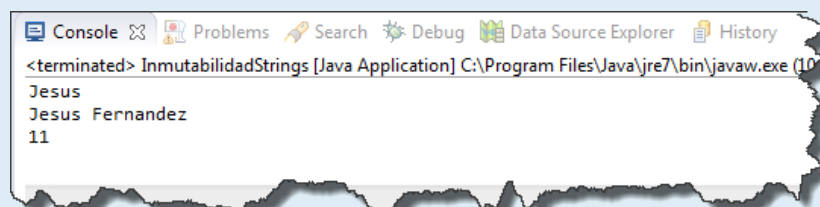
//Al aplicarle el método append(..), NO se crea otra,
//sino que se modifica la original agregándole " Fernandez"
sb.append(" Fernandez");

//Por consola: Jesus Fernandez
System.out.println(sb);

/* Si se trabaja con variables primitivas */
int x=10;
x++;

//Por consola: 11
System.out.println(x);
    }
}

```



Todos los códigos de este tema se guardarán en c:/cursojava/unidad3 o en el workspace correspondiente si se emplea Eclipse.

Otras dos características de los Strings son:

- Un String está **indexado**, es decir, cada uno de sus caracteres tiene asociado un índice: 0 para el primero, 1 para el segundo, etc.
- La cadena de caracteres almacenada por un String **siempre se escribe entre comillas dobles**.

Clase asociada

La clase String pertenece al paquete **java.lang**. De aquí en adelante se nombrará a las clases precedidas de su paquete, **java.lang.String**. La definición de los paquetes se introducirá más adelante.

Constructores

- **String()**: construye un objeto de la clase String sin inicializar.
- **String(String texto)**: construye un String con otro que se le pasa al argumento del constructor. Resulta un poco paradójico que para construir un String se necesite otro String. Ocurre lo siguiente:

Toda variable referenciada debe crearse con un constructor salvo el caso de la variable referenciada String, que es la única que tiene la propiedad de poder construirse como una variable primitiva.

En el segundo constructor se aprovecha esta peculiaridad.

Cuando se trabaja con Strings no suele emplearse ningún constructor sino que se hace uso de la propiedad anteriormente comentada. Así, si se desea crear una String y mostrarla por consola, el código habitual sería:

```
String cadena="Esto es una cadena de texto";
System.out.println(cadena);
```

Así es como se ha trabajado en el ejemplo inicial del tema. Es la forma habitual de crear Strings.

Métodos principales

Para poder aplicar estos métodos es necesario crear un objeto String. Además de estos métodos, la clase String cuenta con otros muchos (consultar la API para más información).

- **int length():** devuelve la longitud del String, incluyendo espacios en blanco. La longitud siempre es una unidad mayor que el índice asociado al último carácter del String.
- **int indexOf(String str, int indice):** devuelve el índice en el que aparece por primera vez el String del primer argumento en el que se aplica el método, a partir del índice especificado en el segundo argumento. Recordar que un String está indexado. Si el índice a partir del que se inicia la búsqueda no existe o el String no aparece, devuelve -1. **MUY USADO.**
- **int indexOf(char ch):** devuelve el índice en el que aparece por primera vez el carácter que se le pasa al argumento. Si no se encuentra el carácter devuelve -1. Se observa que el nombre de este método es igual al anterior aunque su número de argumentos es distinto además de su tipo. A esto, en Java, se le llama **sobrecarga de métodos: mismo nombre pero distinto nº de argumentos o distinto tipo de argumentos o distinto orden**. Ir a la API para comprobar que hay más con este mismo nombre. Este concepto se tratará más en profundidad en temas posteriores.
- **String replace (char viejoChar, char nuevoChar):** cambia el carácter asociado al primer argumento por el que se le pasa al segundo, de la String sobre la que se aplica el método generando una nueva. El String sobre el que se aplica el método no cambia, simplemente se crea otra nueva en base al String sobre el que se aplica el método.
- **String toLowerCase():** devuelve un nuevo String convirtiendo todos los caracteres del String sobre el que se aplica el método, en minúsculas.
- **String toUpperCase():** devuelve un nuevo String convirtiendo todos los caracteres del String sobre el que se aplica el método, en mayúsculas.
- **boolean equals(String str):** investiga si dos String tienen los mismos caracteres y en el mismo orden. Si es así devuelve true y si no false. **MUY USADO**
- **boolean equalsIgnoreCase(String str):** investiga si dos String tienen los mismos caracteres y en el mismo orden sin tener en cuenta las mayúsculas. Si es así devuelve true y si no false. **MUY USADO**
- **boolean startsWith(String str):** devuelve true si el String sobre el que se aplica comienza por el del argumento; false si esto no ocurre.

- **boolean startsWith(String str, int indice):** devuelve true si el String sobre el que se aplica comienza por el del argumento a partir de un determinado índice asociado al segundo argumento; false si esto no ocurre.
- **boolean endsWith(String str):** devuelve true si el String sobre el que se aplica acaba en el del argumento; false si esto no ocurre.
- **String trim():** devuelve un String en base al que se le pasa al argumento, pero sin espacios en blanco al principio ni al final. No elimina los espacios en blanco situados entre las palabras.
- **String substring(int indiceIni, int indiceFin):** devuelve un String obtenido a partir del índice inicial incluido y del índice final excluido; es decir, se comporta como un intervalo semiabierto [indiceIni, indiceFin). Si el índice final sobrepasa la longitud del String, lanza una `IndexOutOfBoundsException`. **MUY USADO.**
- **char charAt (int indice):** devuelve el carácter asociado al índice que se le pasa como argumento del String sobre el que se aplica el método. Si el índice no existe se lanza una `StringIndexOutOfBoundsException` que hereda de `IndexOutOfBoundsException`. **MUY USADO.**

los métodos anteriores que devuelven un String **no modifican** el String sobre la que se aplican; lo que hacen es **crear otra nueva**, en base a la que se emplea para aplicar el método. Recordar la **inmutabilidad** de las Strings.

Ahora que ya se está un poco más familiarizado con los objetos, métodos y variables usados en Java se puede entender mejor la línea de código que se ha venido empleando para mostrar mensajes por consola “`System.out.println(String str)`”. La explicación para esta línea sería la siguiente:

Se está aplicando a un objeto de la clase `PrintStream` el método “`void println(String str)`”. El objeto de `PrintStream` se obtiene mediante la variable de campo estática `out` (representa la salida estándar del sistema, es decir, la consola del DOS en el caso de que el sistema sea un PC) de la clase `System`.

En la API se observa que el método está sobrecargado (este concepto fundamental de la Programación Orientada a Objetos, se estudiará más adelante) y por ello no va a ser necesario transformar sus argumentos en String ya que, en base a lo que le llegue se empleará una versión del método u otra. Por eso no se producen errores cuando se pasa al método una variable primitiva ya sea booleana, entera o real. El programador no debe preocuparse por el tipo de dato que le pasa al método ya que existen múltiples versiones del mismo.

Otra cuestión que se va a introducir y cuyo uso se verá más adelante es la siguiente: si se quiere convertir a String un objeto de cualquier clase, puede emplearse el método “`void toString()`” de `java.lang.Object`. Ojo pues este método sólo es aplicable a objetos, NO a variables primitivas.

Ahora conviene realizar los cuatro primeros ejercicios del tema.

Introducción a los flujos (streams)

Los flujos surgen por la necesidad de las aplicaciones Java de interactuar con el exterior de dos posibles formas:

- Generando salida a la consola del DOS, a un fichero, etc.
- Capturando datos procedentes del teclado, de ficheros, de páginas web, etc.

Vocabulario: Concepto de flujo

Los flujos de datos son **secuencias de bytes**, es decir, un conjunto de bytes en un orden específico que pueden representar cualquier cosa: texto, un archivo, un objeto, una imagen.

Es como un río. El agua en movimiento es el flujo, su contenido son los datos. Lo que permite que esos datos viajen de un origen a un destino es el agua en movimiento, es decir, el flujo. En el caso de la captura, desde un programa Java, de datos introducidos por un usuario mediante teclado, el origen es el teclado, el destino, el programa Java y los datos, lo teclado por el usuario.

Un flujo de dato sirve de **punto** entre la **fuentes de datos** y la **aplicación** y viceversa.

Entre la fuente de datos y la aplicación: lectura de un archivo desde el disco duro, lectura desde el teclado.

Entre la aplicación y la fuente de datos: escritura hacia un archivo del disco duro, visualización en una pantalla.

Java modela flujos mediante clases del paquete **java.io**.

Para comenzar a entender los flujos de datos se va a explicar cómo una aplicación Java captura datos introducidos por el usuario a través del teclado y cómo realiza todo tipo de operaciones sobre los mismos.

Captura de datos desde el teclado

Se necesita lo siguiente:

Obtener un objeto

Obtener un objeto que modele la fuente de entrada de datos “teclado” en un programa Java. Lo primero a tener en cuenta es que el teclado es la entrada de datos estándar de un pc, aunque actualmente hay muchas más (pantallas táctiles, lectores de códigos de barras, etc)

Crear un flujo

Crear un flujo que permita al programador leer datos del teclado. Este flujo va a ser de entrada o lectura de datos y va a modelarse con una objeto de la clase `InputStreamReader`. Se puede acceder al API para estudiar sus constructores.

Crear un filtro

Crear un filtro encargado de leer los datos de forma óptima mediante la aplicación de un método de lectura adecuado. El filtro se modela con un objeto de la clase `java.io.BufferedReader` y el método de lectura adecuado va a ser **`String readLine()`** que lanza una excepción del tipo `IOException` que debe ser gestionada correctamente. En un principio, se gestionará mediante la cláusula `throws IOException` a continuación del método `main`.

El flujo u objeto **`InputStreamReader`** es lo que permite leer datos.

El `System.in` u objeto **`InputStream`** es el **argumento del constructor de `InputStreamReader`** y modela el origen de los datos que se leen, es decir, los datos introducidos por el teclado.

El filtro u objeto **`BufferedReader`** permite, **mediante** la utilización del método **`String readLine()`**, **leer** de forma óptima **del flujo**.

Obtener un objeto que modele la fuente de entrada de datos

El teclado es la entrada de datos estándar de un pc, aunque actualmente hay muchas más (pantallas táctiles, lectores de códigos de barras, etc). Esta entrada genérica de datos a un programa Java se modela mediante un objeto de la clase abstracta **`java.io.InputStream`**. Para obtener un objeto de esta clase utilizaremos la variable `in` de la clase **`java.lang.System`**.

`InputStreamReader isr = new InputStreamReader(System.in);`

La salida de datos se gestiona de la misma manera, la salida de datos estándar en Java es la consola desde donde se ha ejecutado la aplicación, normalmente ms-dos o la consola del editor utilizado. Esta salida genérica de datos se modela mediante un objeto de la clase abstracta **`java.io.OutputStream`**. La variable estática `out` de la clase **`java.lang.System`**, devuelve un objeto de la clase **`PrintStream`**, que hereda de `OutputStream`. Representa la salida estándar de datos de un programa Java. Para escribir datos en la consola podemos utilizar dos métodos de la clase `PrintStream`, **`print(String texto)`** que muestra en la consola el String pasado y **`println(String texto)`** que muestra el texto en la consola y además añade los caracteres de fin de línea y retorno de carro para crear una línea nueva para la siguiente escritura.

`System.out.print("Este texto quiero que aparezca en la consola");`

Crear un flujo que permita al programador leer datos del teclado

Este flujo va a ser de entrada o lectura de datos, es de la clase **`BufferedReader`** y va a modelarse con un objeto de la clase **`InputStreamReader`**.

`BufferedReader br = new BufferedReader(isr);`

Crear un filtro encargado de leer los datos de forma óptima mediante la aplicación de un método de lectura adecuado

El filtro se modela con un objeto de la **clase java.io.BufferedReader** y el **método de lectura adecuado va a ser String readLine()** que, ya se verá, **lanza una excepción del tipo IOException que debe ser gestionada correctamente**. En un principio, se gestionará **mediante la cláusula throws IOException** a continuación del método main.

Ejercicios

Ejercicio 1. Clase String.

20

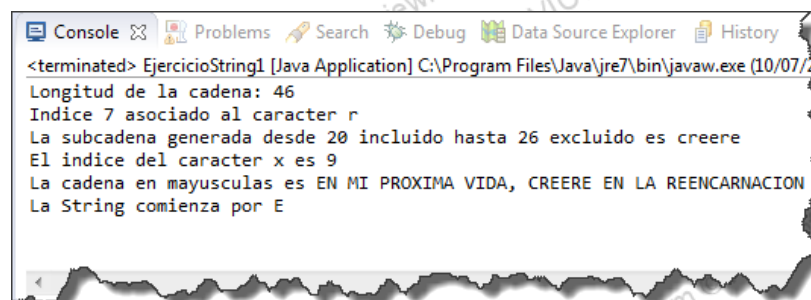
Crear una clase pública de nombre `EjercicioString1` que contenga sólo al método `main` y partiendo de la String **"En mi próxima vida, creere en la reencarnación"** declarada e inicializada como variable de tipo `String`, mostrar por consola lo siguiente:

1. Su longitud
2. El carácter asociado al índice 7
3. La subcadena "creere"
4. El índice que ocupa el carácter 'x'
5. La String transformada en mayúsculas
6. Por último, comprobar si el primer carácter de la String es 'E' y mostrar por consola un mensaje que lo indique.

Recomendaciones.

Todos los código se guardarán en c:\cursojava\unidad3 o en `eclipse_home\MyProjects\unidad3` si se realiza con Eclipse.

Datos a mostrar por consola.

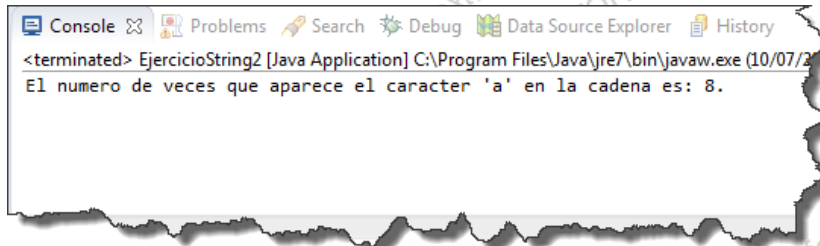


Ejercicio 2. Contar letras.

20

Crear una clase pública de nombre EjercicioString2 que contenga sólo al método main y que muestre por consola el número de veces que aparece la letra "a" en la siguiente String "Esta es una frase de prueba para el ejercicio de contar letras"

Datos a mostrar por consola.



```
<terminated> EjercicioString2 [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10/07/2014)
El numero de veces que aparece el caracter 'a' en la cadena es: 8.
```

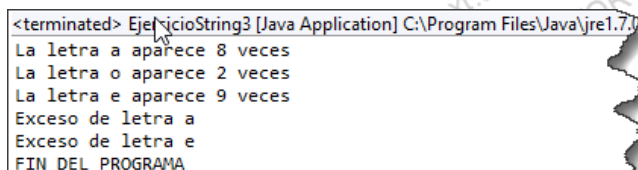
Ejercicio 3. Contar letras II.

20

Crear una clase pública de nombre EjercicioString3 que contenga sólo al método main y mostrar por consola el número de veces que aparecen las letras a, o y e en la String del ejercicio anterior.

Además, si el número de veces que se repite la a es superior a 5 debe aparecer el mensaje "Exceso de a", si el número de veces que se repite la o es superior a 5 debe mostrarse "Exceso de o" y si se repite más de 5 veces la letra e debe mostrarse "Exceso de e"

Datos a mostrar por consola.



```
<terminated> EjercicioString3 [Java Application] C:\Program Files\Java\jre1.7.0_71\bin\javaw.exe (10/07/2014)
La letra a aparece 8 veces
La letra o aparece 2 veces
La letra e aparece 9 veces
Exceso de letra a
Exceso de letra e
FIN DEL PROGRAMA
```

Ejercicio 4. Comprobar NIF. (Ejercicio obligatorio)

40

Verificar si una cadena de texto almacenada en el String nif, **es un NIF correcto o no**.

- Si lo es, se mostrará por consola su parte numérica.
- Si no lo es se mostrará el mensaje "NIF no valido".

Pistas.

- Dos condiciones que debe cumplir el NIF: **tener 9 caracteres y que el último sea una letra**.
- Comprobado esto, verificar que **el resto de caracteres son dígitos**.

Recomendaciones.

- Usar el método **length()** de java.lang.String para conocer el número de caracteres de una cadena de texto.
- Usar el método estático **isLetter(char c)** de java.lang.Character para comprobar que un carácter es una letra.
- Usar el método estático **isDigit(char c)** de java.lang.Character para comprobar que un carácter es un dígito.
- Usar el método **substring(int inicio, int fin)** de java.lang.String para obtener la parte numérica del nif.

Lo necesario para comenzar.

Se aporta un esqueleto para facilitar la creación de la clase.

```
import java.io.IOException;

public class ComprobarNIF {
    public static void main(String args[]) throws IOException {

    }

    /*
     * Método que comprueba si un nif es válido o no. Condiciones que debo comprobar:
     * Longitud cadena = 9
     * Último carácter debe ser letra; el resto dígitos
     * @param nif El nif a comprobar
     * @return true si nif válido
     */
    public boolean comprobar(String nif) {

    }

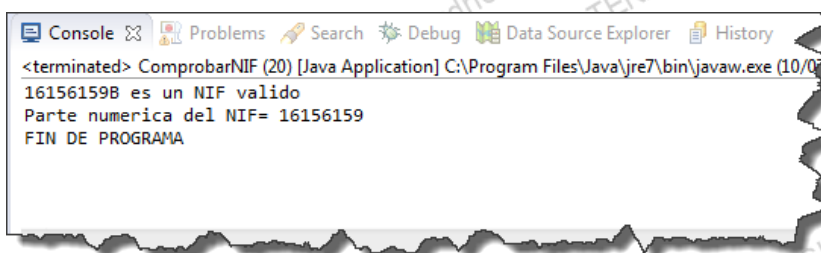
    /*
     * Muestra la parte numérica del nif y un mensaje indicando si es válido
     * @param nif El nif
     * @param nifValido Si es true, el nif es válido
     */
    public void mostrarMensaje(String nif, boolean nifValido) {

    }
}
}
```

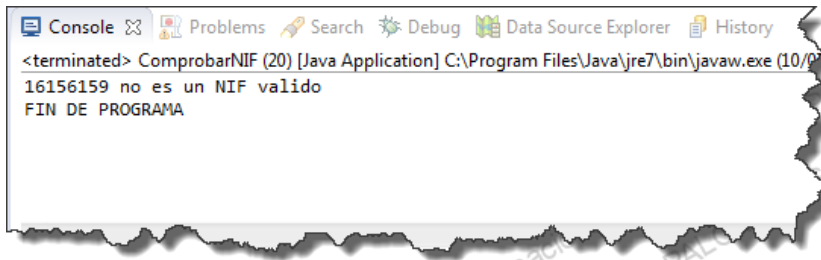
Datos a mostrar por consola

Dos posibles resultados:

- **Si NIF correcto:**



- **Si NIF incorrecto:**



Ejercicio 5. Volúmenes.

30

Calcular el volumen de un cilindro y el de una esfera previa introducción de la altura y radio del cilindro, así como del radio de la esfera.

Recomendaciones.

- Se definirá un método para el cálculo del volumen del cilindro y otro para el de la esfera.
- Se emplearán métodos estáticos de la clase Math y la variable de campo estática que almacena el valor de pi.

Pistas.

- **Volumen esfera** = $(4/3) * \pi * R^3$ **Volumen cilindro** = $\pi * R^2 * H$
- Hay que tener cuidado con las fórmulas que contienen fracciones. Java considera 4/3 como 1 ya que, por defecto, los números enteros se almacenan en una variable int y el cociente de dos enteros, para el programa, es otro entero. Para conseguir que el resultado fuera un double debemos hacer que el numerador sea también un double, por ejemplo, sustituyendo 4/3 por 4.0/3, de este modo se tiene un cociente entre un double y un int cuyo resultado va a ser un double.

Lo necesario para comenzar.

```
import java.io.*;

public class Volumen {
    double PI=Math.PI;
    double calculaVolumenCilindro(double altura,double radio){

    }

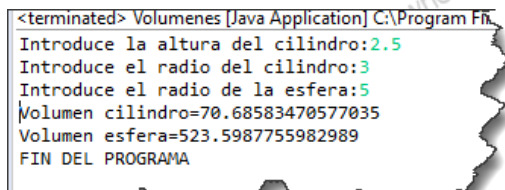
    double calculaVolumenEsfera(double radio){

    }

    public static void main(String args[])throws IOException{

    }
}
```

Datos a mostrar por consola.



```
<terminated> Volumen [Java Application] C:\Program Files\viewnext.adrformacion.com © ADR Infor SL
Introduce la altura del cilindro:2.5
Introduce el radio del cilindro:3
Introduce el radio de la esfera:5
Volumen cilindro=70.68583470577035
Volumen esfera=523.5987755982989
FIN DEL PROGRAMA
```

Ejercicio 6. Adivinar número.

35

Realizar un programa Java compuesto de una clase pública de nombre **AdivinarNumero** que contenga sólo al método main.

Su objetivo será permitir que el usuario averigüe un número entero generado aleatoriamente y comprendido entre [0,100] que se almacenará, dentro del código del programa, en una variable int a la que se llamará numero.

El programa pedirá un número por teclado e informará de si el número que introduce el usuario es mayor o menor que el que se trata de averiguar.

Si no se acierta a la primera, no importa porque **tiene que dejar introducir números de forma ininterrumpida.**

Cuando el usuario acierte, se mostrará un **mensaje de felicitación y el número de intentos** empleados.

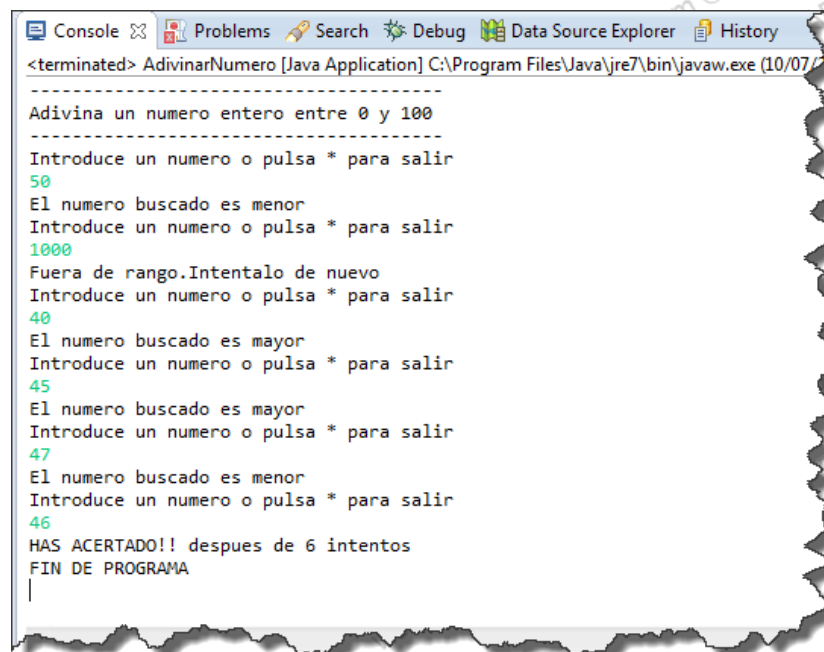
Recomendaciones.

Hay que tener en cuenta que:

- si el usuario introduce un numero no comprendido entre [0,100], el programa mostrará un **mensaje informativo**.
- si el usuario teclea **asterisco**, el programa deberá **finalizar**.
- la generación aleatoria del número a adivinar se realizará con el método estático **void random()** de `java.lang.Math`.

Datos a mostrar por consola.

En esta simulación se interpreta que el usuario mete por consola los números con los que se prueba.



```
<terminated> AdivinarNumero [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (10/07/2012)
-----
Adivina un numero entero entre 0 y 100
-----
Introduce un numero o pulsa * para salir
50
El numero buscado es menor
Introduce un numero o pulsa * para salir
1000
Fuera de rango.Intentalo de nuevo
Introduce un numero o pulsa * para salir
40
El numero buscado es mayor
Introduce un numero o pulsa * para salir
45
El numero buscado es mayor
Introduce un numero o pulsa * para salir
47
El numero buscado es menor
Introduce un numero o pulsa * para salir
46
HAS ACERTADO!! despues de 6 intentos
FIN DE PROGRAMA
|
```

Recursos

Enlaces de Interés



API String

<https://docs.oracle.com/javase/7/docs/api/java/lang/String.html>

En este enlace se puede revisar el API de la clase String.

Glosario.

- **Indexado:** Indexar: Registrar ordenadamente datos e informaciones, para elaborar su índice.
- **Recolector de basura:** Un recolector de basura (del inglés garbage collector) es un mecanismo implícito de gestión de memoria implementado en algunos lenguajes de programación de tipo interpretado o semiinterpretado.
- **Sobrecarga de métodos:** Sobrecarga es la capacidad de un lenguaje de programación, que permite nombrar con el mismo identificador diferentes variables u operaciones. En programación orientada a objetos la sobrecarga se refiere a la posibilidad de tener dos o más funciones con el mismo nombre pero funcionalidad diferente. Es decir, dos o más funciones con el mismo nombre realizan acciones diferentes. El compilador usará una u otra dependiendo de los parámetros usados. A esto se llama también sobrecarga de funciones.