

## Tarea online 02

### 1.- Descripción de la tarea.

*1.1.- Ejercicio 1:  
Utilización del  
depurador del IDE*

*1.2.- Ejercicio 2:  
Signos del Zodiaco.*

*1.3.- Ejercicio 3:  
Juego de las siete y  
media.*

*1.4.- Ejercicio 4:  
Histograma Vertical*

### 2.- Información de interés.

### 3.- Evaluación de la tarea.

Anexo. Licencia de recursos.

## Tarea online 02

**Título de la tarea:** PROG02. Uso de estructuras de control de flujo.

**Unidad:** 02

**Ciclo formativo y módulo:** DAM/DAW, Programación.

**Curso académico:** 2022/2023

## ¿Qué contenidos o resultados de aprendizaje trabajaremos?

El principal resultado de aprendizaje que se va a trabajar con esta tarea será:

- ✓ RA3. Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje.

Esto significa que se trabajará esencialmente con los siguientes contenidos:

- ✓ Uso de estructuras de selección.
- ✓ Uso de estructuras de repetición.
- ✓ Depuración de programas.

# 1.- Descripción de la tarea.



## Caso práctico



**María** continúa con su aprendizaje y su entrenamiento.

Le siguen asignando pequeños problemas a resolver, que luego serán integrados en aplicaciones más complejas. María se los toma como pequeños retos.

Poco a poco va aumentando la dificultad de las tareas que le proponen, y como se ve insegura todavía, ha decidido que va a repasar un poco las estructuras de control de flujo resolviendo algunos problemas típicos, cogiendo soltura en la depuración de los mismos, etc.

## ¿Qué te pedimos que hagas?

Esta tarea consiste en resolver una pequeña relación de ejercicios en los que se usan las estructuras de control de flujo. Antes de empezar a trabajar con los ejercicios, **lee bien los enunciados**, así como los apartados "**Información de interés**" y "**Evaluación de la tarea**". Es importante que revises todo esto antes de empezar a resolver los ejercicios, y también antes de llevar a cabo la entrega para asegurarte de que **cumples con todo lo que ha pedido**. Revisa bien el apartado de evaluación y asegúrate de que estás entregando lo que se te pide y de la forma que se te pide (tipos y formato de los documentos, uno o varios proyectos, clases o archivos Java, etc.).

No debe haber más que un único archivo fuente .java por cada ejercicio. Dada la cantidad de líneas que se necesita para construir cada programa, de momento no necesitamos más. Todo lo que no sea eso es una complicación gratuita. Y las complicaciones gratuitas, en programación, penalizan.

Como siempre, **se valorará en todos los casos la corrección ortográfica y gramatical de los mensajes para comunicarnos con el usuario, así como la presentación clara de cualquier información que se muestre por pantalla.**



# 1.1.- Ejercicio 1: Utilización del depurador del IDE

En este ejercicio **no se pide que elabores código en Java**, pues ya se te proporciona en el archivo `Ejercicio01.java` que se encuentra en el proyecto base que se os proporciona con la tarea. Tu misión consistirá en **seguir una serie de pasos** que se te indican en un documento que tendrás que usar como **ficha de la tarea**. Para **documentar cada paso tendrás que realizar una captura de pantalla de tu entorno Netbeans** que servirá como evidencia de que has realizado apropiadamente ese paso.



[Shafin Protic](#) (Licencia Pixabay)

Para llevar a cabo cada uno de esos pasos deberás utilizar algunas de las funcionalidades que ofrece **la herramienta Netbeans como entorno de depuración** para programas escritos en Java.

Aquí tienes el documento con el **modelo de ficha**, en [versión .odt](#) o  [versión .docx](#), que debes rellenar. Además, se te proporciona un  [documento de ejemplo](#) en el que podrás observar cómo debería quedar tu ficha una vez cumplimentada con todas las capturas. Debes usarlo como modelo para generar tus propias capturas.

No debes modificar el código del programa que se te proporciona (`Ejercicio01.java`) salvo la **línea 40**, donde en lugar de dejar el texto: `"PRUEBA DEL ALUMNO NOMBRE APELLIDO1 APELLIDO2"` **deberás indicar tu nombre y apellidos**.

**En la primera captura debes incluir también, como fondo, tu login en la plataforma**, donde se pueda observar tu nombre y tu foto, para así comprobar que el trabajo lo has realizado tú. En el resto de capturas no es necesario, pero sí debes capturar siempre la **línea 40** (siempre que se trate de una captura de código) para que se pueda observar tu nombre y comprobar que todas las capturas las has ido realizando tú.

Procura ceñirte a la **estructura y las páginas del documento modelo** y a incluir las capturas en cada página que se indica tal y como se muestra en el PDF proporcionado. Tu documento debe intentar parecerse al máximo al PDF salvo por la aparición de tu nombre en el código (línea 40).

Una vez hayas terminado de cumplimentar el documento, genera un PDF a partir de él para evitar problemas de formato a la hora de corregirlo. ¡Y no olvides incluir ese documento en el paquete junto con el resto de tu proyecto!



## Código que debes depurar

Aunque el código que debes depurar se te proporciona ya en el archivo `Ejercicio01.java` que se encuentra en el proyecto base, aquí lo tienes nuevamente por si quieres echarle un vistazo.

[Mostrar retroalimentación](#)

```

/*
 * To change this license header, choose License Headers in Project Properties
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package tarea02;

import java.util.Random;
import java.util.Scanner;

/**
 *
 * @author Programación DAM y DAW IES Trassierra
 */
public class Ejercicio01 {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        //-----
        // Declaración de variables y constantes
        //-----
        Scanner teclado = new Scanner(System.in);
        Random aleatorio = new Random();
        final byte NOTA_MAXIMA = 10;
        final byte NOTA_MINIMA = 0;

        // Variables de entrada (aquí se definen las variables que recibirán los datos)
        float valorIntroducido;

        double suma = 0;
        float notaObtenida;
        double notaFinal;
        int numeroNotas = 0;
        int mediaEntera;
        String calificacionFinal = "";
        System.out.println("Ejercicio 1. Uso del depurador");
        System.out.println("-----");
        System.out.println("PRUEBA DEL ALUMN@ JUAN MATA LOPEZ");
        System.out.println("-----");
        //-----
        // Entrada de datos
        //-----
        // Bloque 1. Solicitud de la calificación deseada.
        // Validación de entrada
        do {
            System.out.println("Introduce la calificación que te gustaría obtener (entre 0 y 10):");
            valorIntroducido = teclado.nextFloat();

        } while (valorIntroducido < NOTA_MINIMA || valorIntroducido >= NOTA_MAXIMA);

        //-----
        // Procesamiento
        //-----
    }
}

```

```

// Bloque 2. Obtención de notas aleatoriamente
do {
    notaObtenida = aleatorio.nextFloat() * 10;
    System.out.printf("\nHas obtenido una nota de: %.2f\n", notaObtenida);
    suma += notaObtenida;
    numeroNotas++;
} while (notaObtenida < valorIntroducido);

// Bloque 3. Calculo de notas medias (valor real y valor entero)
notaFinal = (suma / numeroNotas);

//Obtención del redondeo de calificación a entero
if (notaFinal - ((int) (suma / numeroNotas)) < 0.5) {
    mediaEntera = (int) (notaFinal);
} else {
    mediaEntera = (int) (notaFinal) + 1;
}

System.out.printf("\nLa nota media con decimales obtenida es: %.2f\n", notaFinal);
System.out.printf("\nLa nota media redondeada es obtenida es: %d\n", mediaEntera);

// Bloque 4. Cálculo de la Calificación del Acta
switch (mediaEntera) {
    case 0:
    case 1:
    case 2:
    case 3:
    case 4:
        calificacionFinal += "Insuficiente";
        break;
    case 5:
        calificacionFinal += "Suficiente";
        break;
    case 6:
        calificacionFinal += "Bien";
        break;
    case 7:
    case 8:
        calificacionFinal += "Notable";
        break;
    case 9:
    case 10:
        calificacionFinal += "Sobresaliente";
        break;
}
System.out.println("La calificación final del acta es: " + calificacionFinal);
}
}

```

## 1.2.- Ejercicio 2: Signos del Zodíaco.

Escribe un programa en Java que solicite al usuario un **número de día** y un **número de mes**. Evidentemente el mes debe encontrarse entre 1 y 12 ambos inclusive, y el número de día debe estar entre 1 y el número máximo de días que pueda tener cada mes en concreto (para el mes de Febrero tomaremos como máximo 29; para los meses de Abril, Junio, Septiembre, y Noviembre tomaremos 30; y para el resto de meses 31 ). Si mes o el día introducido no es válido (no está en ese rango), se indicará que la fecha no es correcta, y finalizará el programa mostrando el correspondiente mensaje de error por pantalla.

**Si la fecha es correcta**, el programa procederá a **calcular el signo zodiacal correspondiente a la fecha introducida**:



[Quique](#)

- ✓ Aries. 21 de marzo - 20 de abril.
- ✓ Tauro. 21 de abril – 21 de mayo.
- ✓ Géminis. 22 de mayo – 22 de junio.
- ✓ Cáncer. 23 de junio – 23 de julio.
- ✓ Leo. 24 de julio – 23 de agosto.
- ✓ Virgo. 24 de agosto - 23 de septiembre.
- ✓ Libra. 24 de septiembre - 23 de octubre.
- ✓ Escorpio. 24 de octubre - 22 de noviembre.
- ✓ Sagitario. 23 de noviembre - 21 de diciembre.
- ✓ Capricornio. 22 de diciembre - 20 de enero.
- ✓ Acuario. 21 de enero - 19 de febrero.
- ✓ Piscis. 20 de febrero - 20 de marzo.

Una vez calculada toda esa información, se mostrará por pantalla con el siguiente formato:

- ✓ **si la fecha no era válida**, no se realizará ningún cálculo ni ninguna comprobación más y se indicará por pantalla que no es una fecha válida;
- ✓ **si la fecha es válida** se mostrará por pantalla:
  - ◆ **La fecha actual introducida por teclado con formato de dos dígitos para el día y para el mes;**

Debes tener en cuenta que:

- ✓ se valorará que se **minimice el número de líneas y condiciones**. Intenta utilizar los **operadores lógicos** siempre que te sea posible;
- ✓ antes de comenzar a analizar la fecha, deberías **primero comprobar cuál es el número máximo de días para cada mes**. Para ello puedes aprovechar el uso de la estructura "switch";
- ✓ **una vez que se establezca que la fecha es correcta, debemos comprobar según el mes en el que nos encontremos las dos posibles opciones que tenemos de signo en función del día, según la relación de fechas detallada antes.**
- ✓ **las únicas clases y métodos que puedes usar en el programa** son los de System y Scanner, que utilizamos para la E/S por consola. Aparte de estas, no puedes usar ninguna otra clase, método o herramienta, ni de la API de Java ni propia, para resolver el ejercicio;
- ✓ se deberán utilizar estructuras "if/else" junto con "switch" obligatoriamente;
- ✓ **para obtener la salida con formato de dos dígitos no se permite utilizar ninguna clase de la API de Java que no se haya utilizado hasta el momento**, una buena opción podría ser utilizar algún operador visto en la unidad 1.
- ✓ El **código deberá incluir diferentes tipos de comentarios internos** tanto de una línea como multilínea (los comentarios JavaDoc no son necesarios)



## Ejemplos de ejecución (I)

Aquí tienes algunos ejemplos de ejecución del programa fechas válidas:

Para el año **12 de Enero**:

Mostrar retroalimentación

```
Ejercicio 2. Signos del Zodíaco
-----
Introduce número del MES
1
Introduce el DÍA del mes
12

RESULTADO
-----
El signo correspondiente al 12/01 es:   Capricornio
```

Para el **25 de Abril**:

Mostrar retroalimentación

```
Ejercicio 2. Signos del Zodíaco
-----
Introduce número del MES
4
Introduce el DÍA del mes
25

RESULTADO
-----
El signo correspondiente al 25/04 es:   Tauro
```

Para el **10 de Abril**:

Mostrar retroalimentación

```
Ejercicio 2. Signos del Zodíaco
-----
```

```
Introduce número del MES
4
Introduce el DÍA del mes
10

RESULTADO
-----
El signo correspondiente al 10/04 es:   Aries
```

Para el 21 de Diciembre:

Mostrar retroalimentación

```
Ejercicio 2. Signos del Zodíaco
-----
Introduce número del MES
12
Introduce el DÍA del mes
21

RESULTADO
-----
El signo correspondiente al 21/12 es:   Sagitario
```

Para el 22 de Mayo:

Mostrar retroalimentación

```
Ejercicio 2. Signos del Zodíaco
-----
Introduce número del MES
5
Introduce el DÍA del mes
22

RESULTADO
-----
El signo correspondiente al 22/05 es:   Géminis
```



## Ejemplos de ejecución (II)

Y aquí algunos otros con fechas no válidas:



Para el 40 de Mayo:

Mostrar retroalimentación

Ejercicio 2. Signos del Zodíaco

Introduce número del MES

5

Introduce el DÍA del mes

40

RESULTADO

Fecha incorrecta

Para el -7 de Junio:

Mostrar retroalimentación

Ejercicio 2. Signos del Zodíaco

Introduce número del MES

6

Introduce el DÍA del mes

-7

RESULTADO

Fecha incorrecta

Para el día 15 del mes 14:

Mostrar retroalimentación

Ejercicio 2. Signos del Zodíaco

Introduce número del MES

14

Introduce el DÍA del mes

15

RESULTADO

Fecha incorrecta

Para el día 15 del mes -5:

Mostrar retroalimentación

Ejercicio 2. Signos del Zodíaco

# 1.3.- Ejercicio 3: Juego de las siete y media.

Estamos pensando en desarrollar una aplicación para simular el juego de naipes conocido como "Las Siete y Media". El objetivo del juego consiste en sumar puntos hasta conseguir 7,5 puntos y medio solicitando cartas, pero "SIN PASARSE" y se emplea la baraja española de cartas.

En este juego cada naipe de la baraja tiene un valor, que es el siguiente:



kevberon (Pixabay License)

Valor de los naipes en el juego "Siete y Media"

Naipes	Puntuación
As (1)	1 puntos
Dos (2)	2 puntos
Tres (3)	3 puntos
Cuatro (4)	4 puntos
Cinco (5)	5 puntos
Seis (6)	6 puntos
Siete (7)	7 puntos
Sota (10)	0,5 puntos
Caballo (11)	0,5 puntos
Rey (12)	0,5 puntos
Resto de cartas (8 y 9)	0 puntos- No cuentan

**NOTA:** Para el valor de la banca se generará un valor entre 4 y 7,5 (este código para generar dicho valor vendrá dado en el proyecto base).

Escribe un programa en Java que genere números aleatorios entre 1 y 12, estos números serán evaluados para calcular su puntuación acorde con la tabla anterior, y sumadas a un acumulador.

En cada iteración de nueva carta se deberá mostrar el tipo de carta obtenida y se mostrará la suma acumulada tras sumar la nueva carta obtenida.

Tras dicha suma se deberá preguntar al usuario (si su suma acumulada aun es inferior al tope sin pasarse, es decir inferior a 7,5 puntos), si desea una nueva carta. Esta consulta deberá preguntar mediante un booleano si desea seguir.

- Si desea seguir se repetirá el proceso de generar una nueva carta,
- Si no desea seguir saldremos del bucle.

Una vez "Plantados" o "Pasados", se deberá comprobar quién ha ganado comparando el valor obtenido por la banca al principio del juego, o el propio jugador. Ganará el que más se aproxime a 7,5 puntos sin pasarse, en caso de empate a puntos "gana la banca".

Ten en cuenta que en este programa:

- ✓ se deberá algún tipo de bucle apropiado para ir generando sucesivamente números ("cartas") y comprobar si se desea seguir;
- ✓ se valorará que se **minimice el número de líneas**.
- ✓ las únicas clases y objetos que puedes usar en el programa son `System`, `Random` y `Scanner`, junto con sus métodos, que utilizamos para la E/S por consola, y generación de números aleatorios. Aparte de estas, no puedes usar ninguna otra clase, objeto o método, ni de la `API` de Java ni propio, para resolver el ejercicio;
- ✓ Se deberán **presentar los resultados como aparecen en los ejemplos de ejecución**.
- ✓ Se deberán utilizar estructuras de control `if/else` o/y `if/elseif/else`, pudiéndose utilizar también el operador ternario visto en la unidad anterior, siempre y cuando su simplicidad mejore la estructura del programa.
- ✓ El **código deberá incluir diferentes tipos de comentarios internos** tanto de una línea como multilínea (los comentarios `JavaDoc` no son necesarios)

Recuerda que para obtener un **número aleatorio utilizamos la clase `Random`**, para ello definimos un objeto de esta clase **`Random aleatorio = new Random();`**, y posteriormente se utiliza el método `aleatorio.nextInt()` o `aleatorio.nextInt(int límite)`. El primero de ellos entre 0 y  $2^{32}$ , y el segundo entre 0 y el límite -1.

Ejemplo:

```
Random aleatorio = new Random();

int numAleatorio = aleatorio.nextInt(20); // Genera un número entre 0 y 19.

numAleatorio = 1 + aleatorio.nextInt(20) // Genera un número entre 1 y 20.
```



## Ejemplos de ejecución

Aquí tienes algunos posibles ejemplos de ejecución del programa.

**Partida en la que GANA el JUGADOR**, con una sola carta jugada por obtener mayor puntuación sin pasarse de 7.5

Mostrar retroalimentación

```
Ejercicio 4. Juego Siete y Media
-----
La banca ha jugado, hasta donde te atreves a apostar

JUEGO
-----
Ha obtenido 6
La suma total de sus cartas es: 6.0
Desea seguir (true=false)
false

RESULTADO
```

```
-----
Has ganado: banca(4.5) vs (6.0) jugador
```

**Partida en la que PIERDE el JUGADOR por pasarse**, con dos cartas jugadas. (La banca nunca se pasa tal y como está programado).

Mostrar retroalimentación

```
Ejercicio 4. Juego Siete y Media
-----
La banca ha jugado, hasta donde te atreves a apostar

JUEGO
-----
Ha obtenido 5
La suma total de sus cartas es: 5.0
Desea seguir (true-false)
true
Ha obtenido 6
La suma total de sus cartas es: 11.0

RESULTADO
-----
Ha ganado la banca, te has pasado
```

**Partida en la que GANA el JUGADOR**, con varias cartas jugadas, una de ellas sin valor (no cuenta para el conteo), obteniendo el jugador mayor número de puntos sin pasarse de 7.5

Mostrar retroalimentación

```
Ejercicio 4. Juego Siete y Media
-----
La banca ha jugado, hasta donde te atreves a apostar

JUEGO
-----
Ha obtenido Carta sin valor (0)
La suma total de sus cartas es: 0.0
Desea seguir (true-false)
true
Ha obtenido 2
La suma total de sus cartas es: 2.0
Desea seguir (true-false)
true
Ha obtenido Una figura (0.5)
La suma total de sus cartas es: 2.5
Desea seguir (true-false)
true
Ha obtenido 4
```

```
La suma total de sus cartas es: 6.5
Desea seguir (true-false)
false
```

RESULTADO

-----

Has ganado: banca(4.0) vs (6.5) jugador

**Partida en la que PIERDE el JUGADOR**, por obtener menos puntuación que la banca con una carta jugada.

Mostrar retroalimentación

Ejercicio 4. Juego Siete y Media

-----

La banca ha jugado, hasta donde te atreves a apostar

JUEGO

-----

Ha obtenido 4

La suma total de sus cartas es: 4.0

Desea seguir (true-false)

false

RESULTADO

-----

Ha ganado la banca: banca(6.5) vs (4.0) jugador

**Partida en caso de empate con la banca, PIERDE el JUGADOR**, con varias cartas jugadas.

Mostrar retroalimentación

Ejercicio 4. Juego Siete y Media

-----

La banca ha jugado, hasta donde te atreves a apostar

JUEGO

-----

Ha obtenido 4

La suma total de sus cartas es: 4.0

Desea seguir (true-false)

true

Ha obtenido 1

La suma total de sus cartas es: 5.0

Desea seguir (true-false)

true

Ha obtenido Una figura (0.5)

La suma total de sus cartas es: 5.5

```
Desea seguir (true-false)
false
```

RESULTADO

-----

Ha ganado la banca: banca(5.5) vs (5.5) jugador

## 1.4.- Ejercicio 4: Histograma Vertical

Estamos planteando el desarrollo de una aplicación para simular la generación de un histograma por consola, donde automáticamente se genere una especie de gráfico con diferentes símbolos.

Escribe un programa en Java que solicite al usuario un número filas que deseemos que tenga el programa. Este número deberá ser mayor o igual que 12. Posteriormente, ajustaremos dicho número de filas introducido para que sea el mayor número múltiplo de 4. Es decir, si obtenemos un número que no es múltiplo de 4, debemos ajustarlo al múltiplo menor más próximo, por ejemplo:

- si introducimos el 13, se ajustará a 12.
- si introducimos el 15, se ajustará a 12.
- si introducimos el 16, se ajustará a 16.
- si introducimos el 31, se ajustará a 28.
- etc.

Una vez resuelto el número de filas que tendrá nuestro histograma debemos aplicar el número máximo de símbolos que tendrá cada fila, para ellos calcularemos la mitad del número de filas que tengamos.

Para generar el histograma, seguiremos la secuencia de símbolos siguiente:

1. Símbolo: \*
2. Símbolo: #
3. Símbolo: @
4. Símbolo: >

Es decir, la primera fila presentará "\*" (asteriscos), la segunda fila presentará "#"(almohadillas), la tercera fila presentará "@" (arroba), la cuarta fila presentará ">" (mayor que), la quinta fila presentará nuevamente "\*" (asteriscos), la sexta fila presentará nuevamente "#"(almohadillas) y así sucesivamente hasta el número de filas que le hayamos indicado inicialmente por teclado.

Para cada fila se deben incluir tantos símbolos del mismo tipo como se generen automáticamente para cada fila (entre 3 y la mitad del número máximo de filas, número máximo de símbolos calculado al principio) . Es decir, en cada fila imprimiremos tantos símbolos como nos indique un número generado automáticamente (para cada fila), que estará comprendido entre 3 y la mitad del número de filas que tenga finalmente nuestro histograma, como antes se ha comentado

Cada cuatro filas, es decir, cuando se termine cada iteración con 4 símbolos se debe imprimir por pantalla una línea de guiones como se ve en los ejemplos de ejecución.

- Se valorará el **uso del tipo de bucle más apropiado** en cada caso.
- Como siempre, se valorará que se **minimice el número de líneas**.
- El **código deberá incluir diferentes tipos de comentarios internos** tanto de una línea como multilinea (los comentarios JavaDoc no son necesarios).
- Debes utilizar bucles anidados.



[Pixbay \(inspire-studio\)](#). Histograma (Pixbay (inspire-studio))

Recuerda que para obtener un número aleatorio utilizamos la clase Random, para ello definimos un objeto de esta clase Random aleatorio =new Random(); , y posteriormente se utiliza el método aleatorio.nextInt() o aleatorio.nextInt(int límite). El primero de ellos entre 0 y 232, y el segundo entre 0 y el límite -1.

Ejemplo:

```
Random aleatorio =new Random();
```

```
int numAleatorio=aleatorio.nextInt(20); // Genera un número entre 0 y 19.
```





## Ejemplos de ejecución

Ejemplo 1, con 14 filas de las que se generan 12 finalmente.

Mostrar retroalimentación

```
Ejercicio 4. Histograma
-----
Introduce el número de filas del histograma (Mayor que 10)
14

RESULTADO
-----
El histograma va a presentar 12 filas.
-----
*****
####
@@@@@
>>>>
-----
****
#####
@@@@
>>>
-----
***
#####
@@@@@
>>>>
-----
```

Ejemplo 2, con 19 filas de las que se generan 16 finalmente.

Mostrar retroalimentación

```
Ejercicio 4. Histograma
-----
Introduce el número de filas del histograma (Mayor que 10)
19

RESULTADO
-----
El histograma va a presentar 16 filas.
-----
*****
#####
```

```

@@@@
>>>>
-----
***
#####
@@@@
>>>>
-----
***
###
@@@@@
>>>
-----
***
#####
@@@@
>>>>>>
-----

```

Ejemplo 3, con 20 filas de las que se generan 20 finalmente.

Mostrar retroalimentación

```

Ejercicio 4. Histograma
-----
Introduce el número de filas del histograma (Mayor que 10)
20

RESULTADO
-----
El histograma va a presentar 20 filas.
-----
*****
#####
@@@@@@
>>>>>>
-----
*****
#####
@@@@
>>>
-----
*****
#####
@@@@@@@
>>>>>>
-----
***
#####
@@@@
>>>>>
-----
*****
#####
@@@@
>>>>>>
-----

```

## 2.- Información de interés.

### Recursos necesarios y recomendaciones

A continuación se os proporcionan algunas recomendaciones y consejos importantes sobre la realización de la tarea para que podáis llevarla a cabo con éxito:



mohamed Hassan (Licencia Pixabay)

- ✓ El código de vuestros programas debe seguir la **estructura de programa** que ya se vio en la unidad anterior y su tarea. Salvo que se indique lo contrario, siempre **será obligatorio** escribir vuestros programas usando esa estructura. Recordad todas las indicaciones al respecto que se os dieron en la tarea anterior y no olvidéis descargar el proyecto que debéis usar como guía. En él dispondréis de esa estructura preparada para cada ejercicio: 📁 [Proyecto base para la tarea 2](#).
- ✓ Uso de los **casos de prueba**. ¡Muy importante! Ya se ha hablado sobre esta cuestión en la tarea anterior. Si no lo recuerdas, vuelve a leer lo que se dice al respecto en esa tarea. Y ten siempre en cuenta que **los ejemplos y casos de prueba son para usarlos y probar nuestro programa**. No son un "adorno" para "rellenar" el texto del ejercicio.
- ✓ Para llevar a cabo las **capturas (screenshots)** que es necesario realizar en el **ejercicio 1**, os recomendamos la aplicación 🖼️ [Greenshot](#) o cualquier otra similar en **Sistemas Operativos Windows**. Dispones de unas instrucciones detalladas de las capturas que debes realizar en el documento de ficha en el que debes incluir esas capturas.
- ✓ En caso de que estés trabajando en una plataforma **Mac OSX**, puedes utilizar las siguientes combinaciones de teclas (puedes encontrar más información [aquí](#)):
  - ➡ Para realizar una captura de toda la pantalla en Mac OSX pulsa simultáneamente las teclas **Shift + Command + 3**
  - ➡ Para realizar una captura de parte de la pantalla en Mac OSX pulsa simultáneamente las teclas **Shift + Command + 4**
  - ➡ Para realizar la captura de una ventana o menú en Mac OSX pulsa simultáneamente las teclas **Shift + Command + 4 + espacio**
- ✓ Para realizar capturas de pantalla en **Linux** puedes utilizar las siguientes combinaciones de teclas más usuales, aunque dependiendo de la distribución de linux o actualizaciones pueden no funcionar (puedes encontrar más información [aquí](#)):
  - ➡ **ImprPant**: Guarda una captura de todo el escritorio en la carpeta «Imágenes».
  - ➡ **Shift + ImprPant**: nos permite seleccionar un trozo de la pantalla y guarda la captura en «Imágenes».
  - ➡ **Alt + ImprPant**: Guarda una captura de la ventana activa en «Imágenes».
  - ➡ **Ctrl + ImprPant**: Copia la captura de toda la pantalla en el portapapeles.
  - ➡ **Shift + Ctrl + ImprPant**: Copia la captura de un trozo de la pantalla en el portapapeles.
  - ➡ **Ctrl + Alt + ImprPant**: Copia la captura de la ventana activa en el portapapeles.
- ✓ **Utilizad el depurador** para descubrir los errores que se producen en tus programas. De hecho, tendréis que usarlo explícitamente para resolver el primer ejercicio, pero que os recomendamos que utilicéis normalmente en vuestro trabajo como programadores para ayudaros a localizar y corregir errores en vuestro código. Podríamos parafrasear la famosa expresión "*El perro es el mejor amigo del hombre*" como "*El depurador es el mejor amigo del programador*". Es fundamental que aprendáis a utilizarlo pues vuestro rendimiento como desarrolladores de aplicaciones se verá incrementado exponencialmente.
- ✓ Siempre que lo consideréis oportuno, **includ comentarios útiles en el código**. Os resultará de utilidad en el futuro si tenéis que revisar ese código para llevar a cabo algún tipo de modificación.
- ✓ Los enunciados de cada ejercicio suelen contener toda la información necesaria para su resolución. **Cualquier cosa que no tengáis clara, se la preguntáis "al cliente"** que en este caso es el profesorado, aunque se valorará también mostrar cierta soltura interpretando los enunciados de una manera natural. Siempre tendréis a vuestra disposición los **foros** y el **correo** de la plataforma, así como la asistencia telefónica si fuera necesario. En cualquier caso, toda suposición que se haga al resolver el

problema debe estar correctamente documentada en el código mediante comentarios y debe ser razonable y razonada.

- ✓ Recuerda que **las estructuras de salto incondicional deben ser evitadas** (a no ser que sea totalmente imprescindible, y no es el caso en estos ejercicios). Se trata de elementos no deseables en un programa estructurado y normalmente pueden ser sustituidas por las otras estructuras de control que sí cumplen las reglas básicas de la programación estructurada. En estos ejercicios no vas a tener que usarlas. Su uso podría llevarte a la anulación completa del ejercicio en el que se utilicen.
- ✓ **Indentar** tu código de forma adecuada. Es algo que se tendrá en cuenta a la hora de evaluar la tarea. Recuerda que NetBeans puede hacer esto por ti (selecciona el código a "*formatear*" o "*embellecer*" y pulsa **Alt+Mayús+F**).



## Indicaciones de entrega

Una vez completados todos los ejercicios de la tarea, el envío se realizará a través de la plataforma. Comprime la carpeta del proyecto NetBeans en un fichero .zip y nómbralo siguiendo las siguientes pautas:

**Apellido1\_Apellido2\_Nombre\_PROG02\_Tarea**

¡No olvides incluir en ese paquete de entrega el documento PDF que se te pide elaborar en el **primer ejercicio** (*uso del depurador*)!

## 3.- Evaluación de la tarea.

### Criterios de evaluación implicados

En esta unidad se va a evaluar el resultado de aprendizaje RA3. "Escribe y depura código, analizando y utilizando las estructuras de control del lenguaje." Ello se hará mediante los siguientes Criterios de Evaluación correspondientes al citado resultado de aprendizaje

- ✔ a. Se ha escrito y probado código que haga uso de estructuras de selección.
- ✔ b. Se han utilizado estructuras de repetición.
- ✔ c. Se han reconocido las posibilidades de las sentencias de salto.
- ✔ e. Se han creado programas ejecutables utilizando diferentes estructuras de control.
- ✔ f. Se han probado y depurado los programas.
- ✔ g. Se ha comentado y documentado el código.



[Peggy\\_Marco](#) (Pixabay License)

Nota: El criterio de evaluación (d. Se ha escrito código utilizando control de excepciones) correspondiente al RA3, se trabajará en la unidad 5.

De manera general en esta tarea podremos comprobar que:

- ✔ Se han escrito programas simples.
- ✔ Se ha utilizado el entorno integrado de desarrollo en la creación y compilación de programas simples.
- ✔ Se ha utilizado el entorno integrado de desarrollo para la depuración de programas simples.
- ✔ Se maneja adecuadamente la estructura de control de flujo "if" simple (sin else).
- ✔ Se maneja adecuadamente la estructura de control de flujo "if-else".
- ✔ Se maneja adecuadamente la estructura de control de flujo if compuesta: if-elseif-else.
- ✔ Se maneja adecuadamente la estructura de control de flujo switch.
- ✔ Se maneja adecuadamente la estructura de control de flujo while.
- ✔ Se maneja adecuadamente la estructura de control de flujo do-while.
- ✔ Se maneja adecuadamente la estructura de control de flujo for.
- ✔ Se manejan adecuadamente los operadores de aritméticos para crear expresiones aritméticas sencillas.
- ✔ Se manejan adecuadamente los operadores aritmético-lógicos para crear expresiones lógicas de complejidad media.
- ✔ Se ha utilizado la consola para realizar operaciones de salida de información.
- ✔ Se ha escrito y probado código que hace uso de estructuras de selección.
- ✔ Se ha comentado y documentado el código.
- ✔ Se ha reconocido la sintaxis, estructura y componentes típicos de un programa simple.

### ¿Cómo valoramos y puntuamos tu tarea?

A continuación mostraremos los elementos que podrán penalizar en la evaluación de la tarea. Tenlos muy en cuenta al resolver los ejercicios:

1. **Los programas deben compilar.** Cualquier funcionalidad pedida en el enunciado debe poderse probar y ha de funcionar correctamente de acuerdo a las especificaciones del enunciado. **Si el programa ni siquiera compila, el ejercicio podría considerarse nulo y su puntuación podría llegar a ser 0.**

2. Se utilizan **estructuras de salto incondicional** para el **control del flujo**, o bien herramientas como `return`, `exit(0)` o cualquier otro mecanismo que no sea el **fin de la ejecución natural del programa**.
3. No se respeta la **estructura de bloques** propuesta en la plantilla: **entrada de datos, procesamiento y salida de resultados**.
4. **Algunos identificadores no cumplen con el convenio sobre "asignación de nombres a identificadores"** establecido para el lenguaje Java para constantes, variables, métodos, clases, etc.
5. **Se declaran identificadores que no tienen nombres significativos o descriptivos** que representen de alguna manera la información que están almacenando para que el código quede lo más claro, legible y autodocumentado posible.
6. No se observa una **corrección ortográfica y gramatical**, así como la **coherencia en las expresiones lingüísticas**, tanto en los **comentarios en el código** como en los **textos de los mensajes** que aparezcan en pantalla para pedir información de entrada al usuario o para mostrar resultados de salida. Deben evitarse **mensajes de entrada de datos y/o salida de resultados** inapropiados, descontextualizados, insuficientes o incorrectos.
7. **El código no está correctamente indentado. Es fundamental para poder observar e intuir rápidamente, y de forma visual, la estructura de los programas. Recuerda que NetBeans puede hacer esto por ti (selecciona el código a "formatear" o "embellecer" y pulsa Alt+Mayús.+F).**

En esta otra tabla puedes observar la puntuación máxima asignada a cada ejercicio de la tarea así como los principales elementos que se tendrán en cuenta a la hora de corregir cada uno de ellos:

Rúbrica de la tarea	
<p>P1: Se establece la forma de utilizar de forma correcta y coherente las estructuras de selección, incluso combinando con otras estructuras de control, realizando pruebas para verificar su correcto funcionamiento y definiendo casos por defecto cuando sea necesario.</p> <ul style="list-style-type: none"> <li>• CE a) Se ha escrito y probado código que haga uso de estructuras de selección.</li> </ul>	9,09%
<p>P2: Se han utilizado estructuras de repetición, distinguiendo el uso de las mismas en función del número mínimo de iteraciones así como el conocimiento a priori del número de iteraciones a realizar, optimizando el número de sentencias a ejecutar así como comprobando y verificando mediante pruebas los límites de dichas estructuras. Se aplica en ser caso de ser necesario los bucles anidados y la combinación de estas con otras estructuras de control.</p> <ul style="list-style-type: none"> <li>• CE b) Se han utilizado estructuras de repetición.</li> </ul>	13,64%
<p>P3: Se han reconocido las posibilidades de las sentencias de salto, controlando los casos límite, contemplando todas las casuísticas y dominio de los valores a ejecutar, así como la anidación de las mismas y optimización de casos para realizar un procesamiento óptimo.</p> <ul style="list-style-type: none"> <li>• CE c) Se han reconocido las posibilidades de las sentencias de salto.</li> </ul>	4,55%
<p>P4: Se han creado programas ejecutables utilizando diferentes estructuras de control, la combinación de las mismas y su correcta interacción con diferentes</p>	9,09%

<p>casos de prueba y funcionamiento, combinando estas estructuras para solucionar de manera coherente y óptima los problemas planteados</p> <ul style="list-style-type: none"> <li>• CE e) Se han creado programas ejecutables utilizando diferentes estructuras de control.</li> </ul>	
<p>P5: Se han probado y depurado los programas, realizando diferentes pruebas en casos límites de estructuras de control identificando y subsanando dichos posibles errores sobre baterías de pruebas planteadas por el/la alumn@s sobre todos los posibles casos.</p> <ul style="list-style-type: none"> <li>• CE f) Se han probado y depurado los programas.</li> </ul>	9,09%
<p>P6: Se ha comentado y documentado el código. Además de utilizar correctamente las convenciones del lenguaje Java para identificadores, clases y métodos. Presentando la salida por pantalla como solicitan los requisitos del enunciado, con coherencia, limpieza y corrección lexicográfica.</p> <ul style="list-style-type: none"> <li>• CE g) Se ha comentado y documentado el código.</li> </ul>	4,55%
<p>P7: La aplicación funciona correctamente de acuerdo a las especificaciones requeridas en el enunciado</p> <ul style="list-style-type: none"> <li>• CE a) Se ha escrito y probado código que haga uso de estructuras de selección.</li> <li>• CE b) Se han utilizado estructuras de repetición.</li> <li>• CE c) Se han reconocido las posibilidades de las sentencias de salto.</li> <li>• CE e) Se han creado programas ejecutables utilizando diferentes estructuras de control.</li> <li>• CE f) Se han probado y depurado los programas.</li> <li>• CE g) Se ha comentado y documentado el código.</li> </ul>	50 %