

# 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! (En el zip generado exportando tu proyecto de NetBeans y que has nombrado incluyendo tu nombre y apellidos, añade el pdf que has obtenido rellenando el modelo de ficha, como en el documento de ejemplo).



## Código que debes depurar

Imaginemos que introducimos una opción incorrecta, nos la vuelve a pedir e introducimos la correcta.

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 LUIS NAVARRO ORTIZ
 */
public class Ejercicio01 {

    public static void main(String[] args) {
        //-----
        //          Declaración de variables
        //-----
        // Constantes
        final int MIN_LARGOS = 0;
        final int MAX_LARGOS = 50;
        final byte MAX_INTENTOS = 3;

        // Variables de entrada
        int largosTotales;

        // Variables de salida
        String resultado = "DESCONOCIDO";

        // Variables auxiliares
        boolean entradaValida;
        byte intentos = 0;

        // Clase Scanner para petición de datos
        Scanner teclado = new Scanner(System.in);

        System.out.println("Ejercicio 1. Uso del depurador");
        System.out.println("-----");
        System.out.println("PRUEBA DEL PROFESOR LUIS NAVARRO ORTIZ");
        System.out.println("-----");
        //-----
        //          Entrada de datos
        //-----
        // Bloque 1. Solicitud del número de largos.
        // Validación de entrada
        System.out.println("ENTRENAMIENTO DE NATACIÓN");
        System.out.println("-----");
        entradaValida = false;
        do {
            System.out.print("Introduzca el número de largos realizados (entre " + MIN_
                largosTotales = teclado.nextInt();
            if (largosTotales >= MIN_LARGOS && largosTotales <= MAX_LARGOS) {
                entradaValida = true;
            }
        } while (!entradaValida && ++intentos < MAX_INTENTOS); // Si se alcanza el máxi

        //-----
        //          Procesamiento
        //-----
        // Sólo si la entrada ha sido válida llevamos a cabo los cálculos
        if (entradaValida) {
            // Inicializamos el resultado
            resultado = "{ ";
            // Construimos la cadena largo a largo
            for (int i = 1; i <= largosTotales; i++) {
                if (i % 2 == 0) {
                    resultado += "Espalda "; // Si el largo es par se hace a espalda
                } else if (i % 4 == 1) { // Si se trata del primer largo de cada serie

```

```

        resultado += "Crol "; // Entonces se hace a crol
    } else if (i % 4 == 3) { // Si se trata del tercer largo de cada serie
        resultado += "Braza "; // Entonces se hace a braza
    }
}
resultado += "}";
}

//-----
//          Salida de resultados
//-----
System.out.println();
System.out.println("DESARROLLO DE LARGOS");
System.out.println("-----");
if (entradaValida) {
    System.out.println(resultado);
} else {
    System.out.println("Entrada no válida.");
}
}
}

```

## 1.2.- Ejercicio 2: Máquina expendedora.

Vamos a programar el sistema de control de una máquina expendedora de bebidas. Para ello, la máquina informará al usuario de las bebidas de que dispone así como su precio a través de un menú de opciones (de valores enteros). Se solicita al usuario el número del menú correspondiente a la bebida elegida.

- **Debe existir una opción "0" para salir.** Mostrando el mensaje de despedida "**Gracias por usar la Máquina Expendedora. ¡Hasta luego!**" y no haciendo nada más
- **Si la opción elegida por el usuario no es correcta** se debe volver a mostrar el menú de opciones y solicitar una nueva opción.
- **Si la opción seleccionada es válida**, el usuario introducirá una cantidad de dinero para realizar el pago de la bebida seleccionada.
  - **Si la cantidad de dinero es inferior** al precio se mostrará el mensaje: "Dinero insuficiente. Su dinero será devuelto." y el programa finalizará.
  - **Si la cantidad de dinero es igual o superior al precio** se calculará la diferencia entre la cantidad introducida y el precio de la bebida. Se informará del cambio devuelto y el programa finalizará. El **cambio devuelto debe tener dos decimales obligatoriamente**, pues representa euros y céntimos de euro.



[Crystaltalks](#). Imagen de Japón, Máquina expendedora y Bebidas. (Pixabay License)

**NOTA:** Utiliza un tipo enumerado para definir los tipos de bebidas (COCACOLA, PEPSI, AGUA, ZUMO, OTRO)

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;
- ✓ Para determinar las acciones en cada caso dentro del menú de usuario puedes aprovechar el uso de la estructura "switch";
- ✓ **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;
- ✓ **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

Para una ejecución correcta con la opción 0:

Mostrar retroalimentación

## Ejercicio 2: Máquina expendedora de Bebidas

Bienvenido a la Máquina Expendedora de Bebidas

Seleccione una bebida:

1. COCACOLA - 1.50€
2. PEPSI - 1.50€
3. AGUA - 1.00€
4. ZUMO de naranja - 2.00€
0. Salir

Seleccione una opción:

0

Gracias por usar la Máquina Expendedora. ¡Hasta luego!

Para una ejecución correcta en la que introducimos dinero suficiente y nos devuelve cambio:

Mostrar retroalimentación

## Ejercicio 2: Máquina expendedora de Bebidas

Bienvenido a la Máquina Expendedora de Bebidas

Seleccione una bebida:

1. COCACOLA - 1.50€
2. PEPSI - 1.50€
3. AGUA - 1.00€
4. ZUMO de naranja - 2.00€
0. Salir

Seleccione una opción:

2

Ha seleccionado una PEPSI. El precio es 1,50 €

Ingrese la cantidad de dinero en euros: 2

Compra exitosa. Su cambio es: 0,50 €

Disfrute su PEPSI!

Introducimos una opción incorrecta, luego una correcta pero con dinero insuficiente:

Mostrar retroalimentación

## Ejercicio 2: Máquina expendedora de Bebidas

Bienvenido a la Máquina Expendedora de Bebidas

Seleccione una bebida:

1. COCACOLA - 1.50€
2. PEPSI - 1.50€
3. AGUA - 1.00€

4. ZUMO de naranja - 2.00€  
0. Salir

Seleccione una opción:

6

Opción no válida. Seleccione una bebida válida.

Bienvenido a la Máquina Expendedora de Bebidas  
Seleccione una bebida:

1. COCACOLA - 1.50€  
2. PEPSI - 1.50€  
3. AGUA - 1.00€  
4. ZUMO de naranja - 2.00€  
0. Salir

Seleccione una opción:

2

Ha seleccionado una PEPSI. El precio es 1,50 €

Ingrese la cantidad de dinero en euros: 1

Dinero insuficiente. Su dinero será devuelto.

## 1.3.- Ejercicio 3: Formación romana a partir del número de soldados.

Vamos a desarrollar una aplicación para **simular formaciones militares para una legión romana**, donde automáticamente se **genere un gráfico con el número de soldados y el tipo de formación** que elija el usuario.

Cuando empiece el programa se le darán las instrucciones al usuario y se **solicitarán dos datos**:

- **Número de soldados**: que debe ser al menos 1 (Se debe controlar que no se introduzcan valores negativos obviamente). En caso contrario se vuelve a pedir.
- **Tipo de formación**: Debe ser **"LINEA"**, **"CUADRADO"** o **"TRIANGULO"** (El programa debe ignorar las mayúsculas y minúsculas). Se recomienda no introducir tildes en la entrada de datos. Si no es una de ellas se indicará "Opción NO CORRECTA" y finalizará el programa.



[Sprachprofi](#). Imagen de Legión, Romano y Ejército (Dominio público)

En cuanto a los posibles formaciones se tendrá:

- **La formación en línea dispondrá todos los soldados en la primera fila.** Es decir se dibujarán tantos asteriscos como número de soldados haya introducido el usuario por teclado. Por tanto, debe verse al final de la ejecución, una línea horizontal con tantos asteriscos como número de soldados introducidos.
- **En cuanto a la formación en cuadrado**, primero se debe **calcular el número de filas y columnas (debe ser el mismo) máximo** que se pueden representar con el número de legionarios romanos introducidos (evidentemente el número de soldados restantes no se representarán en la formación). Para ello haremos **uso de la raíz cuadrada**, utilizando el método estático de la librería Math (**Math.sqrt()**). Nota: Este método devuelve un valor double, por tanto habrá que realizar una conversión de tipos. Una vez que tenemos ese valor, **simplemente representaremos con asteriscos (soldados o legionarios romanos), tantas filas de asteriscos y columnas como necesite nuestro cuadrado.**
- Por último, **para la formación en triángulo con los soldados asignados completamos triángulo perfecto invertido (terminando en la última fila con un soldado)** y el resto de soldados no van a la formación. En este caso, debemos calcular el número máximo de filas que podemos representar (este número de filas coincidirá con el número de soldados de la primera fila). Para calcular este número de filas, podemos utilizar la siguiente fórmula:  $((\text{Math.sqrt}(1 + 8 * \text{numSoldados}) - 1) / 2)$ . Así se deberán ir reduciendo en cada fila un soldado sobre la primera fila, además de ir añadiendo espacios en blanco para que se dibuje correctamente como un triángulo invertido.
- Si al hacer la mayor formación posible del tipo indicado con los soldados introducidos sobran soldados, se debe indicar cuantos.

Ten en cuenta que en este programa:

- ✓ Se deberá usar un tipo de bucle apropiado para ir recorriendo las filas y las columnas.
- ✓ Se deberá determinar si en una fila o columna va un soldado o no.
- ✓ Se valorará que se **minimice el número de líneas**.
- ✓ Las únicas clases y objetos que puedes usar en el programa son `System`, `Math` y `Scanner`, junto con sus métodos, que utilizamos para la E/S por consola, y cálculo de la raíz cuadrada. 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 adecuadas para discriminar entre un tipo de formación u otra.
- ✓ El **código deberá incluir diferentes tipos de comentarios internos** tanto de una línea como multilínea (los comentarios JavaDoc no son necesarios)





## Ejemplos de ejecución

Aquí tienes algunos posibles ejemplos de ejecución del programa.

**Formación en LINEA con 36 soldados:**

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
36  
Introduce el tipo de formación  
LINEA  
  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```

**Primero intento meter un número incorrecto de soldados y después le indico que son 8 en línea.**

Mostrar retroalimentación

```
Introduce el número de elementos de la formación (debe ser mayor que cero)  
0  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
8  
Introduce el tipo de formación  
LINEA  
  
* * * * *
```

**Le indico que 8 soldados y luego que en ROMBO (incorrecta)**

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
8  
Introduce el tipo de formación
```

ROMBO  
Opción NO CORRECTA

Dispongo 36 soldados en formación 'CUADRADO'

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
36  
Introduce el tipo de formación  
CUADRADO  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *  
* * * * *
```

Dispongo 36 soldados en formación 'TRIANGULO'

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
36  
Introduce el tipo de formación  
TRIANGULO  
* * * * * * *  
* * * * * *  
* * * * *  
* * * *  
* * * *  
* * *  
* *  
*  
*
```

Dispongo 13 soldados en formación 'CUADRADO'

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
13  
Introduce el tipo de formación  
CUADRADO  
* * *  
* * *  
* * *  
  
De los 13 soldados asignados, una vez hecha la mayor formación posible del tipo indicado, sob
```

Dispongo 13 soldados en formación 'TRIANGULO'

Mostrar retroalimentación

```
Ejercicio03: Formación romana a partir de un número de soldados.  
Introduce el número de elementos de la formación (debe ser mayor que cero)  
13  
Introduce el tipo de formación  
TRIANGULO  
* * * *  
* * *  
* *  
*  
  
De los 13 soldados asignados, una vez hecha la mayor formación posible del tip
```

## 1.4.- Ejercicio 4: Simulador de máquina tragaperras.

Se desea desarrollar una aplicación para simular los premios en una máquina tragaperras.

Dicha máquina obtiene en cada jugada una secuencia de tres símbolos y cada símbolo se obtendrá de forma aleatoria, con la misma probabilidad que los demás símbolos

Los símbolos son P, F, M, N y C. Representan una fruta de las siguientes respectivamente:

- Plátano (P)
- Fresa (F)
- Manzana (M)
- Naranja (N)
- Cereza (C)

El simulador nos debe decir el número de jugadas que se ha tardado en obtener un premio y en caso de obtenerlo, qué premio ha obtenido.

Los premios son:

- Premio 1: PPP
- Premio 2: FFF
- Premio 3: MMM
- Premio 4: NNN
- Premio 5: CCC

Para realizar esta simulación se deben generar de forma aleatoria números de 0 a 4 por ejemplo, y asociar cada número a una de las letras indicadas antes. Una vez obtenida una secuencia de tres elementos (letras o frutas) en cada tirada tendremos una terna que debemos comprobar con cada uno de los premios, en caso de coincidir la terna con uno de los premios, no se realizan más tiradas y se presentan tanto el número de intentos como el premio obtenido.

Ten en cuenta las siguientes indicaciones:

- 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.



[khfalk](#), Máquina Tragaperras (Dominio público)

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

Se obtiene premio en 14 jugadas con secuencia CCC:

Mostrar retroalimentación

Ejercicio 4. Simulador de Máquina Tragaperras.

Voy a generar secuencias de 3 frutas entre Plátano, Fresa, Naranja, Manzana y Cereza hasta co

1-CMP

2-PNM

3-NMM

4-PFF

5-MNM

6-PFM

7-FPM

8-CMC

9-NPP

10-MNN

11-CCN

12-CMN

13-NCM

14-CCC

Has conseguido el premio 5 en el intento 14 con la secuencia: CCC

Ejemplo 2, en 27 intentos obtengo el premio correspondiente a FFF (secuencia de tres fresas):

Mostrar retroalimentación

Ejercicio 4. Simulador de Máquina Tragaperras.

Voy a generar secuencias de 3 frutas entre Plátano, Fresa, Naranja, Manzana y Cereza hasta co

1-FCM

2-PMP

3-CFM

4-FFC

5-PFP

6-MNN

7-PFN

8-FCM

9-NMM

10-MMN

11-PCM

12-FCM

13-MNF

14-FNM

15-FMC

16-PNN

17-CMC  
18-FMN  
19-MMN  
20-MNM  
21-MFC  
22-MPP  
23-MNF  
24-PMP  
25-PPM  
26-PPM  
27-FFF

Has conseguido el premio 2 en el intento 27 con la secuencia: FFF




## 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.
- ✓ **Indenta** 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</p>	9,09%

<p>las mismas y su correcta interacción con diferentes 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 %