**Estos son los ejercicios que se tendrán que hacer en este trimestre. Están divididos en bloques, que se entregan en una sola entrega de edmodo, en la fecha que se especifique en la entrega.**

**Bloque 1 : Repaso del 1º Trimestre y Argumentos:**

**Entrega 2 Febrero 23:59**

**Para todos los ejercicios de este bloque se usará el mismo main, que se entregará en una sola entrega de Edmodo. Tu puntuación dependerá de qué ejercicios logres hacer que funcionen correctamente. Hay ejercicios, marcados en naranja, que son más difíciles, pero el no hacerlos no bloquea la realización de los ejercicios siguientes.**

**+15 XP por código bien documentado**

**Ejercicio 1: Array Global de Nombres (5xp)**  
En el main, crea un array de 30 posiciones, relleno con 30 nombres distintos de persona a tu elección. “Hardcodea” los nombres: En cada ejecución deben ser los mismos 30.

**Ejercicio 2: Número de veces (5xp)**  
Crea una función pideNumeroPositivo, que reciba un único parámetro llamado maximo. Haz que esta función, en su interior, pida por teclado un número, y se asegure de que es positivo y no mayor al valor maximo pasado por parámetros. Si no lo es, que vuelva a pedirlo. Haz que la función devuelva este número.  
Llama a la función en el main usando el número 30 como máximo. Después de crear el array del Ejercicio 1, y guarda en una variable el resultado de la llamada.

**Ejercicio 3: Array de Nombres Seleccionados (10xp)**  
Crea una función subArray, que reciba por parámetros un número (tamaño) y un array que pueda contener nombres de persona (orig). La función debe devolver un array, que contendrá un subconjunto de nombres del array (orig), tantos como indique el parámetro (tamaño). Lo haremos así:

En esta función, primero comprueba si (tamaño) es mayor que el tamaño de (orig). Si es así, directamente devolver null.

Si es menor o igual:

* Crear un array (res) del tamaño indicado por (tamaño)
* Poner en cada posición de ese array (res) un nombre, seleccionado aleatoriamente de entre todos los nombres del array (orig).
* devolver el array (res).

**Ejercicio 4: Que nadie se repita (20xp)**  
Haz que dentro de la función subArray no se puedan repetir los nombres ya escogidos aleatoriamente en el array (orig) para rellenar el array (res).

**Ejercicio 5: Creando subString (5xp)**  
En el main, llama a la función subArray después de llamar a pideNumeroPositivo. Haz que, si la función subArray devuelve null, se vuelva a pedir el número positivo, tantas veces como haga falta, hasta que se obtenga un subArray relleno.

**Ejercicio 6: pideLetra… (5xp)**  
Crea una función pideLetra, sin parámetros. Que en su interior pida una letra por teclado. Que se asegure que es una letra válida en castellano: Entre la a y la z, y la ñ también incluída. Si no lo es,que la vuelva a pedir las veces que haga falta. Devolver esa misma letra, asegurándose de que se devuelve en minúscula.

**Ejercicio 7: empiezaPor… (10xp)**  
Crea una función empiezaPor, que reciba por parámetros una letra (buscada), y un array con nombres (orig). Debe devolver un array, que contenga todos los nombres que empiezan por la letra (buscada). Si no hay ninguno, que devuelva null.

**Ejercicio 8: empiezaPor Ajustado (20xp)**  
Asegúrate de que el array devuelto en empiezaPor tenga justo el tamaño necesario para contener los nombres de (orig) que empiezan por la letra (buscada). Que no quepa ni uno más, ni uno menos.

**Ejercicio 9: imprimeArray (5xp)**  
-Crea una función imprimeArray, que reciba por parámetros un array de nombres, y devuelva un String con todos los nombres del array separados por | . Ejemplo: “Paco | Juan | Marta | Miguel | Tiburcio”. El último nombre no debe tener un | detrás, ni el primero un | delante.

-Crea otra función imprimeArray (con el mismo nombre) que reciba por parámetros un array de enteros, y devuelva un String con todos los eteros separados por | . Ejemplo: 3 | 79 | 9 | 1. No tenéis ni por qué usarla. Solo declararla.

**Ejercicio 10: Seleccionando nombres (5xp)**  
En el main, hasta ahora deberías de haber obtenido un array tras pedir un número y llamar a la función subArray. Usa imprimeArray para imprimir ese array. A continuación, usa la función pideLetra para pedir una letra. Usa empiezaPor para con la letra pedida, devolver todos los nombres de subArray que empiezan por esa letra, e imprime el array que obtengas de la llamada a empiezaPor.

**Ejercicio 11: Recibir tamaño del array por argumentos (15xp)**  
Modifica el main, para que admita recibir por argumentos una opción -tam, de forma que el siguiente argumento sea un número que indica el tamaño del array que se crea al llamar a la función subArray en el ejercicio 5. Ejemplo de llamada:

java -jar miprograma.jar -tam 12

Si recibe el argumento tam, que no pida por teclado el tamaño, sino que use directamente el tamaño que se ha pasado por argumentos de programa.

**Ejercicio 12: Recibir la letra para empiezaPor por argumentos (20xp)**  
Modifica el main, para que admita recibir por argumentos una opción -letra, de forma que el siguiente argumento sea una letra, que se usará en la llamada a función empiezaPor del ejercicio 10. Si el programa recibe este argumento, no pide por teclado el valor con pideLetra. Ejemplo de llamada:

java -jar miprograma.jar -letra a  
Al final de este ejercicio, se deben poder usar los argumentos -tam y -letra juntos o por separado, en cualquier orden, y el programa también debe ejecutar sin necesitar argumentos.

**Ejercicio 13: Ordenación de valores en un array (35xp)**  
**Hardcodea un array de 30 enteros, donde te asegures que ningún número se repite.** A continuación, vamos a implementar un algoritmo muy sencillo para ordenar los elementos de ese array, no es muy eficiente, pero es sencillo. Los valores se van a ir colocando ordenados, en un nuevo array, del mismo tamaño, llamado destino. Para ello, tienes que:

* buscar el número más grande en el array, y guardarlo en una variable masGrande.
* Asignar una variable int actual al mismo valor que masGrande. Asignar otra variable int mayorQue al valor -1.
* Tantas veces como posiciones tenga el array original:
  + Recorrer el array original, si se encuentra un valor más pequeño que el guardado en actual y más grande que el guardado en mayorQue, se actualiza el valor de la variable actual por el que se ha encontrado.
  + Cuando se acabe de recorrer el array original, se inserta en destino, detrás del último que hayamos insertado, el valor de actual.
  + se hace mayorQue=actual, y actual=masGrande.

**Ejercicio 14: Marquesina (40xp)**  
Seguro que los has visto, los típicos carteles de las ópticas/farmacias… esos textos que se van desplazando un puntito cada vez, y van en movimiento, de derecha a izquierda para leer un texto. (<https://www.youtube.com/watch?v=xrJ1J4A1bW0>) . A nivel de java podemos hacer esto con un array de char (char[]). Te reto a que imprimas por pantalla un array de 10 posiciones varias veces. Cada vez que lo imprimas, el texto tiene que estar desplazado una posición más a la izquierda. Usa el texto “Mira cómo mola esto, una marquesina!”. Ejemplo de cómo quedaría la salida por consola del programa:

M

Mi

Mir

Mira

Mira

Mira c

Mira có

Mira cóm

Mira cómo

Mira cómo

ira cómo m

ra cómo mo

a cómo mol

cómo mola

Etcétera. Como ayuda, te sugiero usar un string auxiliar que contega todo el texto, e ir insertando charAt(int i) en el lugar adecuado del array.

**Bloque 2 : Matrices:**

**Entrega 23 FEBRERO 23:00**

**Para todos los ejercicios de este bloque se usará el mismo main, que se entregará en una sola entrega de Edmodo. Tu puntuación dependerá de qué ejercicios logres hacer que funcionen correctamente. Hay ejercicios, marcados en naranja, que son más difíciles, pero el no hacerlos no bloquea la realización de los ejercicios siguientes.**

**+15 XP por código bien documentado**

**Ejercicio 1:** **Crear matriz con dimensiones por teclado (10xp)**

Pedir por teclado un número de filas y un número de columnas, y crear una matriz rectangular de byte con ese número de filas y columnas.

**Ejercicio 2:** **Valores aleatorios (10xp)**

Rellenar la matriz del ejercicio 1 con valores aleatorios entre 1 y 99 para cada posición de la matriz.

**Ejercicio 3: Que nadie se repita (30xp)**  
Asegúrate, que al poner los números aleatorios en el ejercicio 2, ninguno se repita. Es decir, que todas las posiciones de la matriz tengan un número único.

**Ejercicio 4:** **Función imprimeMatriz (15xp)**

crea una función imprimeMatriz, que reciba por parámetros una matriz cualquiera de byte, y devuelva concatenada en un string la representación de una matriz. Que al hacerle sout al resultado, imprima algo como:

1 2 3

4 5 6

7 8 9

**Ejercicio 5:** ¡**Jugamos al bingo! (20xp)**

Usaremos la matriz que tenemos de los ejercicios anteriores para jugar al bingo. De forma automática, sin que tengamos que introducir ningún dato por teclado. Hasta que no se acierten todos los valores contenidos en nuestra matriz, se irán sacando números de forma aleatoria. Un ejemplo de ejecución de todo el programa hasta ahora sería:

Dime nº Filas

> (Se introduce un número, por ejemplo 2)

Dime nº Columnas

> (Se introduce un número, por ejemplo 3)

tu cartón es:

12 20 15

6 1 97

Comenzamos a jugar:

Sale el 38

No lo tienes

Sale el 1

Lo tienes. Aciertos restantes: 5

....

Sale el 20

Lo tienes. Aciertos restantes:0

Fin de partida.

**Ejercicio 6:** **OrdenarMatriz (30xp)**

En tu main, crea una matriz de String de 3x3:

| “muchos” ”ahora” “vez” |

| “la” “y capaces” “a” |

|”de trabajar mucho” “sois” “alumnos” |

Haz una funcion ordenarMatriz, que reciba por parámetro esta matriz de String, y devuelva una con los

mismos elementos, pero ordenados por longitud del String, de forma que la posición [0][0] tenga el

string más corto: “a”, la posición [0][1] el siguiente más corto: “la” … y la [2][2] tenga el string más largo:

“de trabajar mucho”. Imprime la matriz resultante de ejecutar la función en el main.

**Ejercicio 7:** **IntercambiarPosiciones (10xp)**

Crea una función intercambiarPosiciones que reciba como argumentos una matriz de string (puedes

usar la del ejercicio anterior), y los datos necesarios para indicar dos posiciones (como [0][1] y [1][2]).

Dentro de la función, intercambiar los valores de las dos posiciones. Probarlas en el main del ejercicio

anterior: Pedir por teclado la fila y columna de las dos posiciones que se quieren intercambiar, ejecutar

la función, y e imprimir por pantalla la matriz con las posiciones intercambiadas.

**Ejercicio 8: Descifrar matriz (20xp)**

Crear un main donde creeis esta matriz de String:

| “yo quird!” “descifero” “passworar” |

| “techo” “mueño” “sungo” |

Vamos a hacer un principio de cifrado de contraseñas, donde una de las primeras operaciones que se hace es mezclar elementos. En esta matriz, se han mezclado las tres últimas letras de cada palabra de la siguiente forma: En la fila cero, se han puesto en la columna n, las tres últimas letras de la columna n-1, de forma circular. Es decir: la columna 1 “descif**ero**”, tiene las tres últimas letras de la columna 0, “yo qui**rd!**”, con lo que la columna cero debería ser “yo qui**ero**”. De la misma forma, esta columna cero tiene las tres últimas letras de la columna 2 “passwo**rar**”, con lo que la columna 2 debería ser “passw**ord!**”. Por último, la columna 2 tiene las tres últimas letras de la columna 1, que debería ser “descif**rar**”.

En la segunda fila, ocurre lo mismo pero a la inversa, se han intercambiado de forma circular, en la columna n las tres últimas letras de la columna n+1. “techo” tiene las tres últimas letras de “mueño”, “mueño” las de sungo”,y “sungo” las de “techo”.

Tu tarea es crear una función descifrarMatriz que reciba por argumento una matriz de 2x3 cifrada (puedes usar esta) y **devuelva otra matriz distinta** con las palabras descifradas. Es decir, la matriz anterior debería devolver una que pusiese:

| “yo quiero” “descifrar” “password!” |

| “tengo” “mucho” “sueño” |

La función debe descifrar cualquier matriz que venga cifrada igual. Por ejemplo:

| “vondo” “aproy a” “arrasabar” |

| “qui un” “teneriez” “dero” |

Debe devolver:

| “voy a” “aprobar” “arrasando” |

| “quiero” “tener un” “diez” |

**Ejercicio 9: Cifrar matriz (20xp)**

Crea la función inversa a la anterior, que te cifre una matriz de 3x2 string con el algoritmo del ejercicio anterior. Es decir, pasarle la matriz:

| “voy a” “aprobar” “arrasando” |

| “quiero” “tener un” “diez” |

Debe devolver:

| “vondo” “aproy a” “arrasabar” |

| “qui un” “teneriez” “dero” |

**Ejercicio 10: subMatriz extendida (20xp)**

A partir de esta función subMatriz:

/\*\*

\* Coge una submatriz a partir de la posición 0,0 de la

\* matriz original y la devuelve

\* Prerequisito: la matriz orig tiene que ser cuadrada o rectangular

\*

\* @param filas cuántas filas queremos que se copien

\* @param columnas cuántas columnas queremos que se copien

\* @param orig matriz original a partir de la que se construye la submatriz

\* @return matriz subconjunto de la original, de tamaño indicado por filas y columnas. Devuelve null si el tamaño pedido por filas o columnas es más grande que el tamaño de orig

\*/

public static int[][] subMatriz(int filas,int columnas,int[][] orig){

if(filas>orig.length||columnas>orig[0].length){

return null;

}

//Declaro matriz a devolver del tamaño introducido

int[][] ret=new int[filas][columnas];

for (int i = 0; i < filas; i++) {

for (int j = 0; j < columnas; j++) {

ret[i][j]=orig[i][j];

}

}

return ret;

}

Modifícala para que el origen de la submatriz no sea siempre la posición 0,0, asegurándose de que la submatriz se puede seguir creado. Para ello:

* Incluye dos argumentos más en la cabecera, filaOrigen y columnaOrigen.
* Asegúrate de que la distancia entre filaOrigen+filas no sea mayor a orig.length, y que columnaOrigen+columnas no sea mayor a orig[0].lenght. Devuelve null si esta comprobación falla.
* Si no se ha devuelto null antes, crear una submatriz que comience en filaOrigen y columnaOrigen, y devuélvela.   
  Ejemplo:  
  Con la función original si a la matriz:  
  1 2 3 4 5 6 7

8 9 10 11 12 13 14

15 16 17 18 19 20 21

La llamamos con:   
 int[][] m2=subMatriz(2,2,m);  
 Devuelve en m2:

* 1 2

8 9

Si con nuestra modificación la llamamos así:  
 int[][] m2=subMatriz(2,2,m,1,4);   
 Donde 1 es el valor de filaOrigen, y 4 el valor de columnaOrigen, devuelva:  
 12 13

19 20  
 Porque empieza a capturar la submatriz a partir de la posición (1,4)  
  
Prueba esta función en main con una matriz de enteros que tu elijas. Imprime esa matriz original, pide por teclado filas,columnas,filaOrigen y columnaOrigen, y que imprima la matriz resultado.

**Ejercicio 11: Matrices irregulares: Pirámide alimenticia (30xp)**

Vamos a usar una matriz irregular para crear una pirámide alimenticia, con los valores:

Dulces

Carne Embutidos

Pescado Lácteos Huevos

Pasta Pan Patatas Harinas

Imprímela por pantalla usando alguna de las funciones que ya has hecho, y crea otra función que la imprima centrada (Aproximadamente). Ej:

Dulces

Carne Embutidos

Pescado Lácteos Huevos

Pasta Pan Patatas Harinas

**Ejercicio 12: Bolas que rebotan en una matriz (45xp)**

Vamos a crear una matriz cuadrada de un número de filas y columnas impar. Pide por teclado un número. Asegurate que es mayor que 1 e impar. A partir de ella crearemos una matriz de char llena del caracter - , que en la columna central de todas las filas tendrá un o. Ejemplo con tamaño 5:

- - o - -

- - o - -

- - o - -

- - o - -

- - o - -

Haz en un bucle, que durante 50 veces, las bolitas se desplacen a izquierda o derecha en la misma fila en la que están, cuando una bolita llega a un extremo, “rebota”, y pasa desplazarse en el sentido contrario. Las bolitas de las filas impares empiezan desplazandose hacia la izda, y las bolitas de las filas impares hacia la derecha. Que en cada iteración del bucle se imprima la matriz. Ej:

Iteración 1  
-----------------

- - o - -

- - o - -

- - o - -

- - o - -

- - o - -

Iteración 2  
-----------------

- - - o -

- o - - -

- - - o -

- o - - -

- - - o -

Iteración 3  
-----------------

- - - - o

o - - - -

- - - - o

o - - - -

- - - - o

Iteración 4  
-----------------

- - - o -

- o - - -

- - - o -

- o - - -

- - - o -

Iteración 5  
-----------------

- - o - -

- - o - -

- - o - -

- - o - -

- - o - -

Iteración 6  
-----------------

- - - o -

- o - - -

- - - o -

- o - - -

- - - o -

Etcétera.

**Ejercicio 13: Sacar media recursivamente (30xp)**

Vamos a hacer un par de funciones que obtengan la media de los valores de una matriz de short recursivamente. Una de ellas:  
public float mediaMatriz( short [][] matriz, byte nfila){ … }

otra:

public float mediaFilaMatriz(short [] fila, byte ncolumna){ … }

El main debe llamar a mediaMatriz(matriz,0). Recursivamente recorrer las filas, y en cada fila llamar a la función mediaFilaMatriz, que es también una función recursiva, que recorre todas las columnas de una fila. Usar esas funciones para imprimir por pantalla la media de una matriz. Por ejemplo:

1 2 3

4 5 6

7 8 9

Debe dar: 5.0

**Bloque 3 : Programación Orientada a Objetos:**

**Entrega 8 MARZO 23:00**

**Para todos los ejercicios de este bloque se usarán varios proyectos distintos, uno por cada parte , que se entregará en una sola entrega de Edmodo. Tu puntuación dependerá de qué ejercicios logres hacer que funcionen correctamente. Hay ejercicios, marcados en naranja, que son más difíciles, pero el no hacerlos no bloquea la realización de los ejercicios siguientes.**

**PARTE DE FECHAS**

**1 - Creación clase Fecha (15XP)**

Crea una clase llamada Fecha. Sus variables internas deberían ser: dia, mes, año. El mes debería estar almacenado en un número. **Crea su constructor, getters y setters.** Haz el método toString, para que imprima las fechas en formato español: dia/mes/año (ej: 03/01/2020). Crea en un main una variable de tipo fecha, inicializa sus valores e imprímelos.

**2 - Función esValida (15XP)**

Crea un método de instancia en la clase Fecha, llamada esValida. Debe devolver un booleano que indique si el objeto fecha que la llama es correcta o no. Por ejemplo, no valen fechas como 34/05/2020 o 30/02/2020. Ignora la existencia de los años bisiestos: El 29 de Febrero nunca puede existir.

**3 - Método mayorQue(15XP)**

Crea un método de instancia llamado mayorQue en la clase Fecha. Debe devolver true si la fecha que llama a la función es mayor que una fecha pasada por parámetro, o false en caso contrario.

**4 - Crear el main(5XP)**

En el main, pide ingresar dos fechas, pidiendo para cada una, primero año, luego mes y luego día. Usa la función mayorQue para imprimir “La primera fecha es mayor”, o “La segunda fecha es mayor o igual”.

**PARTE DE TAMAGOTCHI**

Vamos a implementar un tamagotchi en modo texto, que pueda hacer las cosas básicas de un tamagotchi: Comer, dormir, etcétera. Tendremos tres tipos de tamagotchi: Perro, Pez y Pájaro. Cada uno tendrán una serie de valores de tipo byte que indicarán sus necesidades, a los que les daremos valores entre 0 (totalmente insatisfecha) y 10 (totalmente satisfecha). Para completarlo, haz las siguientes actividades:

**5 - Crear clases (15XP)**

Primero, crea la clase abstract Tamagotchi. Va a ser la clase madre del resto. Tendrá como variables internas: Un String nombre, un byte hambre, un byte sueño, un byte higiene y un byte diversion. Créale un constructor, los getter y los setters.

Crea una clase Perro que herede de Tamagotchi, con una única variable interna propia: byte paseo. Indicará entre 0 y 100 cómo de satisfecha está su necesidad de pasearse. Haz su constructor, getters y setters.

Crea una clase Pájaro que herede de Tamagotchi, con una única variable interna propia: byte vuelo. Indicará entre 0 y 100 cómo de satisfecha está su necesidad de volar. Haz constructor, getters y setters.

Crea una clase Pez que herede de Tamagotchi,sin variables internas propias. Haz su constructor.

**6 - Constructores alternativos (10XP)**

Crea un constructor en Tamagotchi que solo reciba el nombre, le ponga ese nombre a tamagotchi, e inicialice todos los byte internos a 100. Crea un constructor que haga lo mismo en las tres clases hija.

**7 - ¡Nace, Tamagotchi! (15XP)**

Crea una función public final static Tamagotchi nacer(). En su interior, que pida por teclado una raza (asegurarse que es una de las tres), y un nombre. Después, declara una variable de tipo tamagotchi, e inicialízala como una nueva instancia de una de las tres clases hijas, según la raza que se haya introducido por teclado. Por último, devuelve esa variable.

**8 - ¡Imprime el Tamagotchi! (5XP)**

Crea una función toString dentro de tamagotchi. Haz que te imprima todas las variables internas menos nombre, con este formato:

Hambre - (nivel de hambre)

Sueño - (nivel de hambre)

Higiene - (nivel de hambre)

Diversión - (nivel de hambre)

Ejemplo:

Hambre - 100

Sueño - 75

Higiene - 20

Diversión- 50

**9 - ¡Imprime todos los Tamagotchi! (10XP)**

En cada clase hija, sobreescribe la función toString, y haz que concatene todo esto en un String para devolver:

1- llame a getNombre() y lo imprima junto a su raza así: Nombre - Raza

2- Imprime una cantidad que estimes oportuna de guiones medios en otra línea, como separador: ---------------

3- Llama al toString de tu superclase, tamagotchi

4- Imprime, con el mismo formato que en el toString de tamagotchi, las variables internas propias de esta clase hija, si las tiene. Ejemplo de cómo quedaría:

Sputnik - Perro

---------------------------

Hambre - 100

Sueño - 75

Higiene - 20

Diversión- 50

Paseo - 66

**10 - ¡Está vivo! (20XP)**

Crea un main para este proyecto (recuerda que es tu tamagotchi por pantalla. Crea un menú, otro distinto al de fechas), en el que declares una variable de tipo Tamagotchi, y la inicialices al valor que devuelve la función nacer(). Después, imprime que se ejecute hasta que se seleccione la opción salir, que ofrezca las siguientes opciones: 0 - Salir, 1 - No hacer nada, 2 - Dar de comer, 3 - Dormir , 4 - Bañar, 5 - Jugar. Además, si es un perro, ofrecerá la opción 6 - Pasear, y si es un pájaro la opción 6 - Sacar a volar. Para eso, puedes usar la función getClass().getName() que vienen heredadas de Object. Mira un poco de documentación para averiguar cómo ;). Usa un switch para imprimir en cada opción lo que has elegido. Por ejemplo, si se elige la opción 1, que imprima por pantalla algo como: ¡Voy a alimentar al tamagotchi!. Estas opciones por ahora no tendrán ningún efecto en el tamagotchi. Al final de cada iteración, que vuelva a imprimir el tamagotchi por pantalla.

**11 - ¡Está haciendo caca! (15XP)**

Crea una función public void vivir() dentro de Tamagotchi. Haz que disminuya en 5 todas las variables byte de tamagotchi. Sobreescribe la función en las clases Perro y Pájaro. Haz que además de lo anterior, disminuyan en 5 su variable interna. En el main, llama a vivir() de tu variable tamagotchi del main al final de cada iteración del bucle, justo antes de imprimir el tamagotchi.

**12 - ¡Que se te muere! (15XP)**

Haz una función public boolean seHaMuerto() dentro de Tamagotchi, que compruebe si alguna de sus variables internas byte llega a cero, devuelva true. Si ninguna ha llegado a cero, devuelva false. En las subclases Perro y Pájaro, sobreescribe la función, y haz que compruebe también si la variable interna de esa subclase ha llegado a cero (además de las de la clase tamagotchi), para decidir si se ha muerto o no. En el main, con tu variable tamagotchi (pongamos que se llama t), incluye, al final del bucle, después de imprimir el tamagotchi:

if(t.seHaMuerto()){

System.out.println(“Tu tamagotchi se ha muerto.”);

break;

}

**13 - ¡Sabe hacer cosas! (5XP)**

Crea en tamagotchi los siguientes métodos:

* public void comer() -> Aumenta en 15 el nivel de hambre;
* public void dormir() -> Aumenta en 15 el nivel de sueño;
* public void baniarse() -> Aumenta en 15 el nivel de higiene;
* public void jugar() -> Aumenta en 15 el nivel de diversión;

Crea en Perro el método:

* public void pasear() -> Aumenta en 15 el nivel de paseo;

Crea en Pajaro el método:

* public void volar() -> Aumenta en 15 el nivel de vuelo;

**14 - ¡Están pasando cosas! (5XP)**

En el main, en cada una de las opciones que tienes en el Switch, llama a la función del ejercicio anterior que corresponda con cada una de las opciones del switch. La opción no hacer nada no llama a ninguna.

**¡Enhorabuena! Si lo has hecho todo bien, has programado tu primer tamagotchi funcional. Lo puedes jugar de verdad. Para él, el tiempo no pasa en horas, sino en vueltas del bucle, pero el pobretico ha nacido así, hay que quererlo tal y como es. Si algo te revienta o no funciona como funcionaría un tamagotchi real, repasa los ejercicios, algo estás haciendo mal.**