Ejercicios propuestos

Verás que esta relación de ejercicios es sustancialmente más larga que las anteriores. Vamos a estar por aquí un buen rato, así que ponte cómodo/a. Ahora tenemos los elementos suficientes en la mano como para empezar a escribir programas más complejos que empiecen a hacer cosas de verdad.

Los ejercicios están colocados en orden de dificultad creciente. Más o menos, porque hay excepciones. Esto quiere decir que, cuanto más avances, más difíciles serán... y mayor será la satisfacción de hacerlos funcionar.

En cada ejercicio, tendrás que desarrollar un programa Java completo. Eso incluye escribir al menos una clase con el método o métodos necesarios para resolver el problema, y al menos otra clase donde se encuentre el método main(). En algunos casos, puede que necesites desarrollar varias clases y crear varios objetos.

Ejercicio 3.1: Positivo y negativo (*)

Leer un número por teclado mediante un método, y decir si es positivo o negativo con otro método. La salida por consola puede ser algo así como: "el número X es positivo"

Ejercicio 3.2: Raíz cuadrada (*)

Calcular la raíz cuadrada de un número introducido por teclado. Hay que

tener la precaución de comprobar que el número sea positivo.

Ejercicio 3.3: Restar (*)

Leídos dos números por teclado, llamémosles A y B, calcular y mostrar la resta del mayor menos el menor. Por ejemplo, si A = 8 y B = 3, el resultado debe ser A – B, es decir, 5. Pero si A = 4 y B = 7, el resultado debe ser B – A, es decir, 3.

Ejercicio 3.4: Año bisiesto (*)

Determinar si un año es bisiesto o no. Los años bisiestos son múltiplos de 4; utilícese el operador módulo. ¡Pero hay más excepciones! Los múltiplos de 100 no son bisiestos, aunque sean múltiplos de 4. Pero los múltiplos de 400 sí, aunque sean múltiplos de 100. Qué follón. La Tierra es muy maleducada al no ajustarse a los patrones de tiempo humanos.

Resumiendo: un año es bisiesto si es divisible entre 4, a menos que sea divisible entre 100. Sin embargo, si un año es divisible entre 100 y además es divisible entre 400, también resulta bisiesto.

Ahora, prográmalo, a ver qué sale.

Ejercicio 3.5: Parte decimal (*)

Averiguar si un número real introducido por teclado tiene o no parte fraccionaria (utilícese el método Math.round() que aparece descrito en el capítulo 1, o si no, búscalo en Internet)

Ejercicio 3.6: Números ordenados (*)

Leer tres números por teclado, X, Y y Z, y decidir si están ordenados de menor a mayor. Complétalo con otro método que nos diga si los números, además de estar ordenados, son consecutivos.

Ejercicio 3.7: Contar cifras (*)

Determinar el número de cifras de un número entero. El algoritmo debe funcionar para números de hasta 5 cifras, considerando los negativos. Por ejemplo, si se introduce el número 5342, la respuesta del programa debe ser 4. Si se introduce –250, la respuesta debe ser 3.

Para los quisquillosos: no, el 0 a la izquierda no cuenta.

Ejercicio 3.8: Mayor, menor, mediano (*)

Dados tres números enteros, A, B, C, determinar cuál es el mayor, cuál el menor y cuál el mediano. Y da gracias a que no lo hemos hecho con 4 variables. Prohibido usar arrays, suponiendo que sepas lo que son. Es un ejercicio de lógica, no de bucles.

Ejercicio 3.9: Pares (*)

Ahora sí empiezan los bucles. Escribe un programa que muestre todos los números pares entre A y B, siendo estos dos valores dos números introducidos por teclado. A debe ser menor que B, claro. En caso contrario, el programa debe avisarnos, pero con cariño.

Ejercicio 3.10: Impares (*)

Escribir todos los números impares entre dos números A y B introducidos por teclado. En esta ocasión, cualquier de ellos puede ser el mayor. Habrá que comprobar, entonces, cuál de los dos números, A o B es mayor, para empezar a escribir los impares desde uno o desde otro.

Ejercicio 3.11: Pares o nones (*)

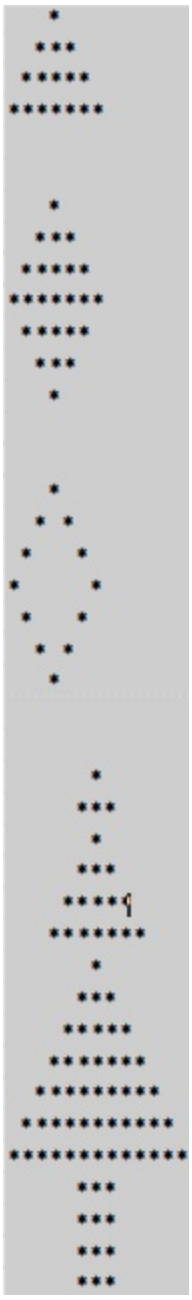
Escribe un programa que pregunte al usuario si desea ver los números pares o impares y que, dependiendo de la respuesta, muestre en la pantalla los números pares o impares entre A y B. Cualquiera de ellos puede ser el mayor. Y sí, es un batiburrillo de los dos anteriores, así que intenta reutilizar todo el código que puedas. En programación eso no se considera plagio, salvo que te pillen.

Ejercicio 3.12: Dibujando con asteriscos (*)

Este ejercicio es un poco más complicado que los anteriores, pero también mucho más divertido.

Escribe una clase capaz de generar en la pantalla, mediante bucles, estos bonitos diseños. Añade alguno de tu propia cosecha, por favor.

(Por si queda alguna duda: el último se supone que es un árbol de navidad)



Ejercicio 3.13: Tabla de multiplicar (*)

Vamos con un clásico de los cursos de introducción a la programación. El usuario teclea un número y el programa muestra la tabla de multiplicar de ese número. Pero que quede bonito, por favor, algo así como:

```
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
etc.
```

Ejercicio 3.14: Acumulador simple (*)

Calcular la suma de todos los números pares entre 1 y 1000. Es decir, 2 + 4 + 6 + ... + 998 + 1000. No preguntes en los foros de programación, seguro que puedes hacerlo por ti mismo.

Ejercicio 3.15: Acumulador interactivo (*)

Calcular el valor medio de una serie de valores enteros positivos introducidos por teclado. Para terminar de introducir valores, el usuario debe teclear un número negativo.

Ejercicio 3.16: Estadística básica (*)

Calcular el valor máximo de una serie de 10 números introducidos por teclado. Completa luego la clase para que también averigüe el valor mínimo, el medio, la desviación típica y la mediana. Si no sabes lo que es alguna de estas cosas, háztelo mirar. También puedes probar en internet.

(Importante: calcular cosas como la desviación típica sin utilizar arrays es una tarea propia de criaturas mitológicas, así que usaremos la solución de este ejercicio como excusa para introducir los arrays en un contexto con significado. Pero, antes, deberías haber pensado un rato en las posibles soluciones)

Ejercicio 3.17: Notas de clase (*)

El usuario de este programa será un profesor, que introducirá las notas de sus 30 alumnos de una en una. El algoritmo debe decirle cuántos suspensos y cuántos aprobados hay. Las notas pueden valer entre 1 y 10, y el programa no debe aceptar valores fuera de ese rango.

Ejercicio 3.18: Factorial (*)

Calcular el factorial de un número entero N. Recuerda que el factorial de un número es el producto de ese número por todos los enteros menores que él. Por ejemplo, el factorial de 5 (simbolizado 5!) se calcula como: 5! = 5 x 4 x 3 x 2 x 1.

Cuando funcione, prueba a calcular el factorial de un número muy grande, como 288399849! o algo parecido, y verás qué risa.

Ejercicio 3.19: Sucesión de Fibonacci (*)

La famosa sucesión de Fibonacci es una sucesión no convergente de números enteros que comienza así:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Cada número de la sucesión se calcula sumando los dos números anteriores (excepto los dos primeros, que son, por definición, 0 y 1).

Se da la curiosa circunstancia de que los números de la sucesión de Fibonacci aparecen con sorprendente precisión en muchas estructuras naturales, como los ángulos de crecimiento de las ramas de árboles cuando son iluminados verticalmente, la disposición de los pétalos de los girasoles o de las piñas en los pinos, la forma de las cochas de los caracoles, y cosas así. Si lo piensas, es un poco inquietante que un producto de la imaginación humana como son las matemáticas tenga una relación tan estrecha con la naturaleza. ¿O era al revés? Bueno, al diablo.

A lo que íbamos: escribe un programa que muestre en la pantalla los N primeros términos de la sucesión de Fibonacci, siendo N un número entero introducido por el usuario.

Ejercicio 3.20: Número de la suerte (*)

El número de la suerte o lucky number, por si hay alguien de Cuenca, es una tontuna de los numerólogos y otros charlatanes que se obtiene sumando todas las cifras de la fecha de nacimiento de un modo azaroso. Por ejemplo, como yo nací el 15 de octubre de 1974 (15-10-1974), se supone que mi número de la suerte es 15+10+1974 = 1999. Ahora sumo todas las cifras de 1999 así: 1+1+1+9 = 12. Como aún tengo dos dígitos, vuelvo a sumarlos. 1 + 2 = 3.

Por lo tanto, 3 es mi número de la suerte. Si alguna vez me toca la lotería y llevo un número acabado en 3, os aviso.

Escribe un programa que, dada una fecha de nacimiento, calcule el número de la suerte de esa persona.

Ejercicio 3.21. Primo (*)

Determinar si un número N introducido por teclado es o no primo. Recuerda que un número primo no es el hijo de mi tío, sino aquél que sólo es divisible

por sí mismo y por la unidad.

Ejercicio 3.22: Eratóstenes (*)

Generalizar el algoritmo anterior para averiguar todos los números primos que existen entre 2 y 1000 (a este proceso se le conoce como criba de Eratóstenes, que no es que tenga mayor importancia, pero sirve para ponerle un título interesante a este ejercicio)

Ejercicio 3.23: Omirps (*)

Un omirp es una de esas cosas que nos hace dudar sobre la estabilidad mental de los matemáticos. Se trata de un número primo que, al invertirlo, también da como resultado un número primo. Por ejemplo, el número 7951 es primo y, si le damos la vuelta, obtenemos 1597, que también es primo. Por lo tanto, 7951 es un omirp.

Se trataría, pues, de introducir un número y que el programa determine si es un omirp o no. O más difícil todavía (redoble de tambores): hacer un programa que muestre la lista de omirps entre 0 y N, siendo N un número introducido por el usuario.

(Este último programa, me consta, es el tipo de cosas que te pueden pedir hacer en una entrevista de trabajo)

Ejercicio 3.24: Lotería primitiva (*)

Generar combinaciones al azar para la lotería primitiva (6 números entre 1 y 49). Debes utilizar el método Math.random() que vimos en el capítulo 1. Por ahora, no te preocupes porque los números puedan repetirse.

No hace falta que corras a la administración de loterías a jugar la primera combinación que te salga. Lo han probado muchas promociones de alumnos antes que tú y no nos consta que nadie haya conseguido salir de pobre.

Ejercicio 3.25: Quiniela

Generar combinaciones al azar para la quiniela (14 valores dentro del conjunto 1, X o 2, por si hay alguien de otro planeta que no sepa cómo se rellena una quiniela). El resultado debe ser algo así, pero generado al azar:

1 - X - X - 2 - 1 - 1 - 1 - 2 - 2 - X - 1 - X - X - 2

Ejercicio 3.26: Calculadora (*)

Diseñar un algoritmo que lea dos números, A y B, y un operador (mediante una variable de tipo carácter), y calcule el resultado de operar A y B con esa operación. Por ejemplo, si A = 5 y B = 2, y operación = "+", el resultado debe ser 7. El algoritmo debe seguir pidiendo números y operaciones indefinidamente, hasta que el usuario decida terminar (utiliza un valor centinela para ello)

Ejercicio 3.27: Juego del número secreto (*)

El ordenador elegirá un número al azar entre 1 y 100. El usuario irá introduciendo números por teclado, y el ordenador le irá dando pistas: "mi número es mayor" o "mi número es menor", hasta que el usuario acierte. Entonces el ordenador le felicitará y le comunicará el número de intentos que necesitó para acertar el número secreto, sin humillarlo en caso de que hayan sido más de cinco.

Ejercicio 3.28: Nóminas (*)

Escribe un método que calcule el dinero que debe cobrar un trabajador a la semana, pasándole como parámetros el número de horas semanales que ha trabajado y el precio que se le paga por cada hora. Si ha trabajado más de 40 horas, el salario de cada hora adicional es 1,5 veces el de las horas convencionales.

(Por si alguien no lo sabe, 40 horas es el número máximo de horas que se pueden trabajar semanalmente en España; el resto a partir de ahí se consideran "horas extraordinarias" y se deben pagar de forma generosa)

Escribe otros métodos que calculen además:

- El salario bruto mensual, sabiendo que todas las horas que excedan de 40 semanales se consideran horas extra. Las primeras 5 horas extra se cobran a 1,5 veces la tarifa normal, y las demás al doble de la tarifa normal.
- Los descuentos por impuestos: se le descontará un 10% si gana menos de 1000 € al mes, y un 15% si gana más de esa cantidad.

El salario neto, es decir, el dinero que cobrará después de descontar los impuestos

Ejercicio 3.29: Polígonos

Cuando se trabaja con polígonos regulares, conociendo su número de lados, la longitud de cada lado y la apotema, se puede calcular el área y el perímetro según estas expresiones:

- Área = nº lados x longitud del lado x apotema / 2
- Perímetro = nº de lados x longitud del lado

Escribe un programa que pregunte esos tres valores y calcule el área y el perímetro de cualquier polígono regular, y además escriba su denominación (triángulo, rectángulo, pentágono, hexágono, etc, hasta polígonos de 12 lados)

Ejercicio 3.30: Máquina expendedora (*)

Escribe un programa que simule el mecanismo de devolución de monedas de una máquina expendedora. El programa preguntará una cantidad el euros y luego calculará qué monedas debería devolver la máquina para completar esa cantidad. Por ejemplo, si la cantidad es 1,45 €, la respuesta del programa debe ser: una moneda de 1 €, dos de 20 céntimos y una de 5 céntimos.

Por favor, que no funcione como los cajeros de los párkings, que son capaces de devolverte cinco euros en modenas de céntimo.

Puedes elegir entre estos dos enfoques:

- Escribir un único método que se encargue del calcular todas las monedas que hacen falta.
- Escribir varios métodos, uno para cada tipo de moneda, y que cada uno se encargue de determinar cuántas monedas de ese tipo hacen falta para realizar la devolución.

Ejercicio 3.31: Predicción meteorológica

Escribe un programa para predecir el tiempo que va a hacer mañana a partir de varios datos atmosféricos suministrados por el usuario. Estos datos son:

- La presión atmosférica: puede ser alta, media o baja.
- La humedad relativa: también puede ser alta, media o baja

El programa se encargará de calcular la probabilidad de lluvia, la

probabilidad de que haga sol y la probabilidad de que haga frio. Tiembla, Roberto Brasero. Nuestro cálculos superprecisos serán estos:

A) Para calcular la probabilidad de lluvia:

Presión	Humedad	Probabilidad de lluvia
Baja	Alta	Muy alta
Baja	Media	Alta
Baja	Baja	Media
Media	Media	Media
En cualquier otro caso		Baja

B) Para calcular la probabilidad de que haga sol:

Presión	Humedad	Probabilidad de que haga sol
Baja	Alta	Baja
Baja	Media	Media
Baja	Alta	Media
Media	Media	Media
En cualquier otro caso		Alta

C) Para calcular la probabilidad que haga frío:

Presión	Humedad	Probabilidad de que haga frío
Baja	Alta	Alta
Baja	Media	Alta
Media	Alta	Alta
Media	Media	Media
En cualquier otro caso		Baja

Ejercicio 3.32: Calculadora de edad (*)

Escribe un programa que pregunte al usuario su fecha de nacimiento y la fecha del día de hoy, y calcule la edad del usuario en años.

(No es tan fácil como parece)

Este programa se puede mejorar haciendo que calcule la edad en años, meses y días (¡incluso en horas, minutos y segundos!), pero es una labor por ahora solo apta para los/las más valientes.

Ejercicio 3.33: Contar días

Escribe un programa que calcule cuántos días han transcurrido desde el 1 de enero del año en curso hasta el día de hoy (incluido), y cuántos días faltan desde el día de mañana hasta el 31 de diciembre. Es necesario tener en cuenta la posibilidad de que el año sea bisiesto. Recuerda que ya hiciste una clase que determinaba si un año era o no bisiesto.

Ejercicio 3.34: TPV ultramegasimplificada

Escribe un programa que puedan utilizar en una tienda para calcular el descuento de los artículos. En esta tienda aplican un descuento del 15% en todos los artículos vendidos a los mayores de 65 años, y de un 10% a los menores de 25.

El programa debe preguntar, para hacer cada venta, el precio del artículo que se vende y la fecha de nacimiento del cliente. Entonces calculará la edad del mismo y, a partir de ella, determinará el descuento que se debe aplicar al artículo.

El proceso se repetirá hasta que se introduzca un precio de artículo negativo.

Para calcular la edad de los clientes puedes retuilizar la clase que escribiste en el ejercicio anterior (he aquí una de las ventajas de la OOP: el código se puede reutilizar fácilmente).

Ejercicio 3.35: Número a letra

Escribe un programa que lea un número de hasta 5 cifras por teclado y lo escriba en forma de letra. Por ejemplo, si se introduce el número 1980, la salida del programa debe ser "mil novecientos ochenta". Utiliza una función para cada posición del número (unidades, decenas, centenas, etc)

Diseña una batería de casos de prueba basada en clases de equivalencia,

como se describe en el capítulo, y luego flipa al descubrir la cantidad de errores que tiene tu código.

Ejercicio 3.36: Atracción gravitatoria

La atracción gravitatoria de la Tierra imprime una aceleración a todos los cuerpos de 9,8 m/s2 (aproximadamente) a nivel del mar, como recordarás de tus clases de física (risas). Sin embargo, ese valor decrece con la altitud, de manera que, por ejemplo, en lo alto del Everest la aceleración gravitatoria es un poco menor (y, por lo tanto, los cuerpos pesan un poco menos)

La gravedad a una altura h puede calcularse según la expresión:

...donde R es el radio de la Tierra (aproximadamente, 6.370.000 metros)

Escribe un programa que muestre por pantalla una tabla en la que aparezca la disminución de la gravedad con la altura en intervalos de 100 kilómetros, hasta alcanzar una altura especificada por el usuario. Por ejemplo, si la altura que introduce el usuario es 400 kilómetros, la salida del programa debe ser más o menos así:

Altitud (km)	Gravedad (m/s2)
0	9,80
100	9,50
200	9,21
300	8,94
400	8,68

Ejercicio 3.37: Contar cifras

Escribe un programa que pida por teclado un número entero y determine cuántas de sus cifras son pares y cuántas impares. Por ejemplo, si el número tecleado es 78 532, la salida del programa debe ser:

Ese número tiene 3 cifras impares y 2 cifras pares

Ejercicio 3.38: Conjetura de Goldbach

La Conjetura de Goldbach, originada durante la correspondencia entre los matemáticos Christian Goldbach y Leonhard Euler en el siglo XVIII, afirma lo siguiente:

"Todo número par mayor que 4 puede escribirse como la suma de dos

números primos impares no necesariamente distintos"

Por ejemplo, el número 20 puede escribirse como la suma de dos primos: 13 + 7. Otro ejemplo: el número 8 puede escribirse como 5 + 3. Y otro ejemplo más: el número 14 puede escribirse como 7 + 7

Este hecho es sólo una conjetura, es decir, no está demostrado que se cumpla para todos los números pares mayores que 4, aunque hasta ahora no se ha encontrado ninguno que no lo cumpla. Es uno de los problemas abiertos más antiguos de la historia de las matemáticas.

Escribe un programa que pida un número N por teclado (N debe ser par y mayor que 4) y compruebe si cumple la conjetura de Goldbach, mostrando los dos números primos que, sumados, dan N.

Ejercicio 3.39: Ruleta

Escribe un programa que permita jugar a la ruleta con el ordenador. Supondremos que la ruleta tiene 20 números rojos y 20 negros. El jugador, que tendrá una suma de dinero inicial, apostará una cantidad (siempre menor que el dinero que le quede) a un número y un color. La ruleta, que puedes simular con el método Math.random(), la manejará el ordenador y comunicará al jugador el resultado. Si acierta, multiplicará por 10 el dinero apostado. Si falla, lo perderá. El proceso se repetirá hasta que el jugador decida abandonar el juego, o bien se quede sin dinero.

Abstenerse ludópatas.

Ejercicio 3.40: Dados

Escribe un programa para jugar a los dados con el ordenador. Las reglas del juego son las siguientes:

- El jugador humano dispondrá de una cantidad inicial de dinero que se introducirá por teclado.
- El jugador apostará una cantidad de dinero (siempre igual o menor del que le queda)
- Después, se tirarán tres dados (lo cual se puede simular con el método Math.random())
- Si en los tres dados sale la misma cantidad, el dinero apostado por el

jugador: a) se multiplica por 5 si en los dados ha salido un 6; b) se multiplica por 3 si sale cualquier otra cantidad; c) si solo en dos dados de los tres sale la misma cantidad, el dinero apostado se multiplica por 2

- En cualquier otro caso, el dinero apostado se pierde
- El proceso se repite hasta que el jugador se queda sin dinero o hasta que decide dejar de jugar.

Ejercicio 3.41: Juego de memoria

Reglas del juego:

El juego comenzará preguntando el nivel de dificultad, que puede ser fácil, medio o difícil.

El ordenador elegirá al azar una serie de números. La serie consistirá al principio en un solo número. Luego serán dos números, luego tres, luego cuatro... y así hasta diez. Los números de la serie solo pueden ser tres: 1, 2 y 3.

El ordenador mostrará su serie de números durante un tiempo en la pantalla. Ese tiempo será tanto menor cuanto más alto sea el nivel de dificultad.

Después, la serie se borrará y el jugador debe demostrar su memoria y sus reflejos repitiéndola. Más tarde esto no te hará ninguna gracia, cuando tengas que ponerte a averiguar como se hace eso de borrar la consola. Qué fácil es decirlo.

Si el jugador acierta en todos los números de la serie, el ordenador pasará a su siguiente serie (que tendrá un número más). Si el jugador falla, el juego termina.

Si el jugador es capaz de repetir con éxito todas las series (desde la que solo tiene un número hasta la que tiene 10), el jugador gana.

Ejercicio 3.42: Las Tres en Raya

Vamos a hacer una versión del popular juego de las Tres en Raya para jugar contra el ordenador. No será un juego con inteligencia artificial como el de la WOPR (si no sabes qué es la WOPR, bueno, ¿para qué sirve wikipedia?), pero te permitirá pasar un buen rato programando, que es de lo que se trata.

El juego se desarrolla en un tablero de 3 x 3 casillas en el que los jugadores

van disponiendo sus fichas tratando de formar una línea vertical, horizontal o diagonal.

Las fichas del jugador humano tendrán forma de círculos (O), y las del ordenador, forma de aspa (X)

Al principio, el tablero está en blanco. Comenzará jugando uno de los dos jugadores, elegido aleatoriamente. El jugador que comienza colocará una ficha en el tablero. Después, será el turno del otro jugador. El proceso se repite hasta que uno de los dos consigue colocar tres fichas formando una línea, o hasta que ya no es posible colocar más fichas (situación de “tablas”)

Ejercicio 3.43: Tragaperras

Vamos a escribir ahora un programa que simule el funcionamiento de una máquina tragaperras. No es que yo tenga ningún problema con los juegos de azar, ¿eh?, es solo que son simples y adecuados para simular con un ordenador en el momento en el que nos encontramos. Además, puedo dejarlos cuando quiera.

El programa debe tener el siguiente comportamiento:

a) Preguntará al usuario con cuánto dinero inicial desea jugar (en euros). Esta cantidad no puede ser menor que 1 euro ni mayor que 50.

b) Cada jugada costará 0,50 euros, que se descontarán automáticamente del saldo que el jugador tenga en cada momento.

c) Cada jugada consiste en una combinación de tres frutas elegidas al azar entre estas seis: Manzana, Naranja, Fresa, Cereza, Limón, Sandía.

d) El jugador no gana nada si las tres frutas que salen son distintas.

e) En cambio, si varias frutas coinciden, el jugador gana un premio, que pasa a incrementar su saldo. El premio será:

- Si dos frutas cualesquiera son iguales: dos cerezas, 3 euros; dos sandías, 2 euros; cualquier otra fruta, 1 euro.
- Si las tres frutas son iguales: tres cerezas, 30 euros; tres sandías, 20 euros; tres fresas, 10 euros; cualquier otra fruta, 5 euros.

f) Después de cada jugada, la máquina comunicará al jugador la combinación que ha salido y le dirá si ha ganado algún premio.

g) Después de eso, la máquina le dirá al jugador cuál es su saldo actual y le preguntará si desea seguir jugando. Si el jugador se queda sin dinero, el juego terminará automáticamente sin preguntar nada.

Este es un ejemplo de ejecución del programa:

```
*** BIENVENIDO AL JUEGO DE LA TRAGAPERRAS ***
¿Con cuánto dinero desea empezar (de 1 a 50 euros) ? 60
Cantidad incorrecta
¿Con cuánto dinero desea empezar (de 1 a 50 euros) ? 20
COMIENZA EL JUEGO...
La combinación de esta jugada es: NARANJA – CEREZA – SANDÍA
Lo siento, no ha obtenido ningún premio
Su saldo actual es de 19,5 euros.
¿Desea jugar otra vez (S/N) ? S
La combinación de esta jugada es: SANDÍA – SANDÍA - LIMÓN
¡Enhorabuena! Ha ganado 20 euros.
Su saldo actual es de 39,5 euros.
¿Desea jugar otra vez (S/N) ? N
¡Hasta la próxima!
```

Ejercicio 3.44: Juego de Nim (simplificado)

El Nim es un juego clásico de estrategia que se supone originario de Oriente. Sus reglas, en nuestra versión ligeramente modificada, son las siguientes:

a) Se tienen tres montones de palillos, cada uno de los cuales contiene, al principio del juego, entre 3 y 6 palillos. El número inicial de palillos en cada montón lo determinará el ordenador al azar y puede variar de una partida a otra, pero siempre habrá un mínimo de 3 y un máximo de 6.

b) El jugador humano elige un montón y quita de él 1 ó 2 palillos.

c) Después, el ordenador hace lo mismo: elige un montón y quita 1 ó 2 palillos.

d) Los pasos b) y c) se repiten hasta que sólo queda un palillo en total. El jugador que deba retirar ese último palillo, pierde.

Para que quede más claro, mostramos un ejemplo de funcionamiento del programa. Las líneas precedidas de ">" se supone que son introducidas por el usuario. El resto son la salida producida por el programa. Asegúrate de entender bien lo que tiene que hacer el programa antes de empezar a pensar

en cómo vas a programarlo.

Bienvenido al Juego de Nim.
El contenido actual de los montones es:
Montón 1: 5 palillos
Montón 2: 3 palillos
Montón 3: 4 palillos
¿De qué montón quieres quitar palillos (1, 2 ó 3) ?
> 1
¿Cuántos palillos quieres quitar del montón 1 ?
> 2
Es mi turno...
Elijo el montón 2
Quito 2 palillos
El contenido de los montones es:
Montón 1: 3 palillos
Montón 2: 1 palillo
Montón 3: 4 palillos
¿De qué montón quieres quitar palillos (1, 2 ó 3) ?
> 1
¿Cuántos palillos quieres quitar del montón 1 ?
> 3
Error: el número de palillos que puedes quitar es 1 ó 2
¿Cuántos palillos quieres quitar del montón 1?
> 2
Es mi turno...
Elijo el montón 2
Quito 1 palillo
El contenido de los montones es:
Montón 1: 1 palillo
Montón 2: 0 palillos
Montón 3: 4 palillos
¿De qué montón quieres quitar palillos (1, 2 ó 3) ?
> 2
Error: ese montón ya no tiene palillos
¿De qué montón quieres quitar palillos (1, 2 ó 3) ?
> 1
¿Cuántos palillos quieres quitar del montón 1 ?
> 2
Error: en ese montón sólo queda 1 palillo
¿Cuántos palillos quieres quitar del montón 1?

> 1
Es mi turno...
...etc...

Este es, con diferencia, el programa más complejo que hemos creado hasta ahora, así que te vamos a dar algunas pistas de cómo podrías estructurarlo. Pero, ¡ojo!, solo son sugerencias. Puedes hacerlo de otro modo si lo prefieres.

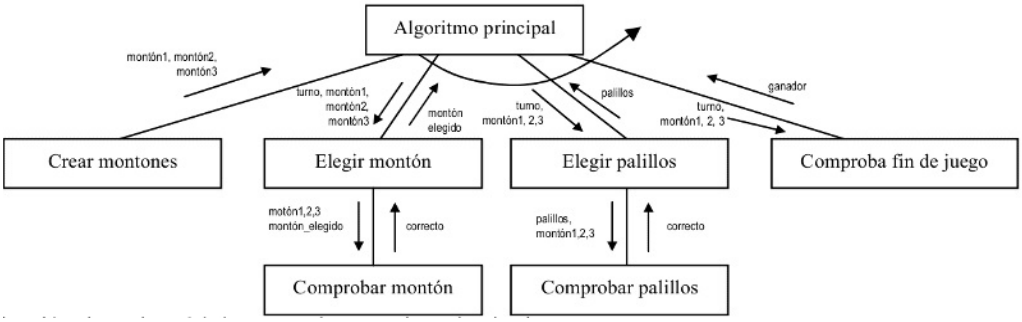
Seguramente sea buena idea disponer de estos métodos:

- Método crearMontones(): se encarga de crear los 3 montones, asignándole a cada uno de ellos una cantidad aleatoria de palillos (entre 3 y 6).
- Método elegirMontón(): dependiendo del parámetro "turno", se encargará de pedir al usuario que elija un montón, o bien de lograr que el ordenador elija un motón al azar. Después, llama al método comprobarMontón() para ver si el montón elegido es correcto. Si es así, devuelve el montón elegido. Si no, mostrará un mensaje de error ("Error: ese montón ya no tiene palillos") y volverá a pedir que se elija un montón.
- Método comprobaMontón(): mira si el motón elegido tiene algún palillo; si es así, devuelve correcto = verdadero; si no, devuelve correcto = falso
- Método elegirPalillos(): dependiendo del valor de "turno", le pide al usuario que elija el número de palillos que quiere retirar, o bien hace que el ordenador lo elija al azar. Ese número debe ser 1 ó 2. Luego llama a comprobarPalillos(), que decide si el número de palillos elegido es correcto. Si es así, se devuelve el número de palillos elegidos. Si no, se muestra un mensaje de error ("Error: ese montón no tiene tantos palillos") y se vuelve a pedir que se introduzca un número de palillos.
- Método comprobarPalillos(): si el número de palillos elegido es igual o menor que los que quedán en el montón, devuelve correcto = verdadero; si no, devuelve correcto = falso.
- Método comprobarFinJuego(): mira si, entre todos los montones, sólo queda por retirar un palillo. Si es así, el juego debe acabar y el ganador es el jugador que posee el turno actualmente.

Además, necesitarás un método que se encargue, como es lógico, de invocar

a todos los demás en el orden adecuado, de restar de cada montón los palillos que se hayan retirado, y de controlar cuándo debe finalizar el juego.

Te lo mostramos en un diagrama:



Escribe primero los métodos de más bajo nivel (comprobar_montón y comprobar_palillos). Luego continúa por los de nivel superior y, por último, escribe el algoritmo principal.