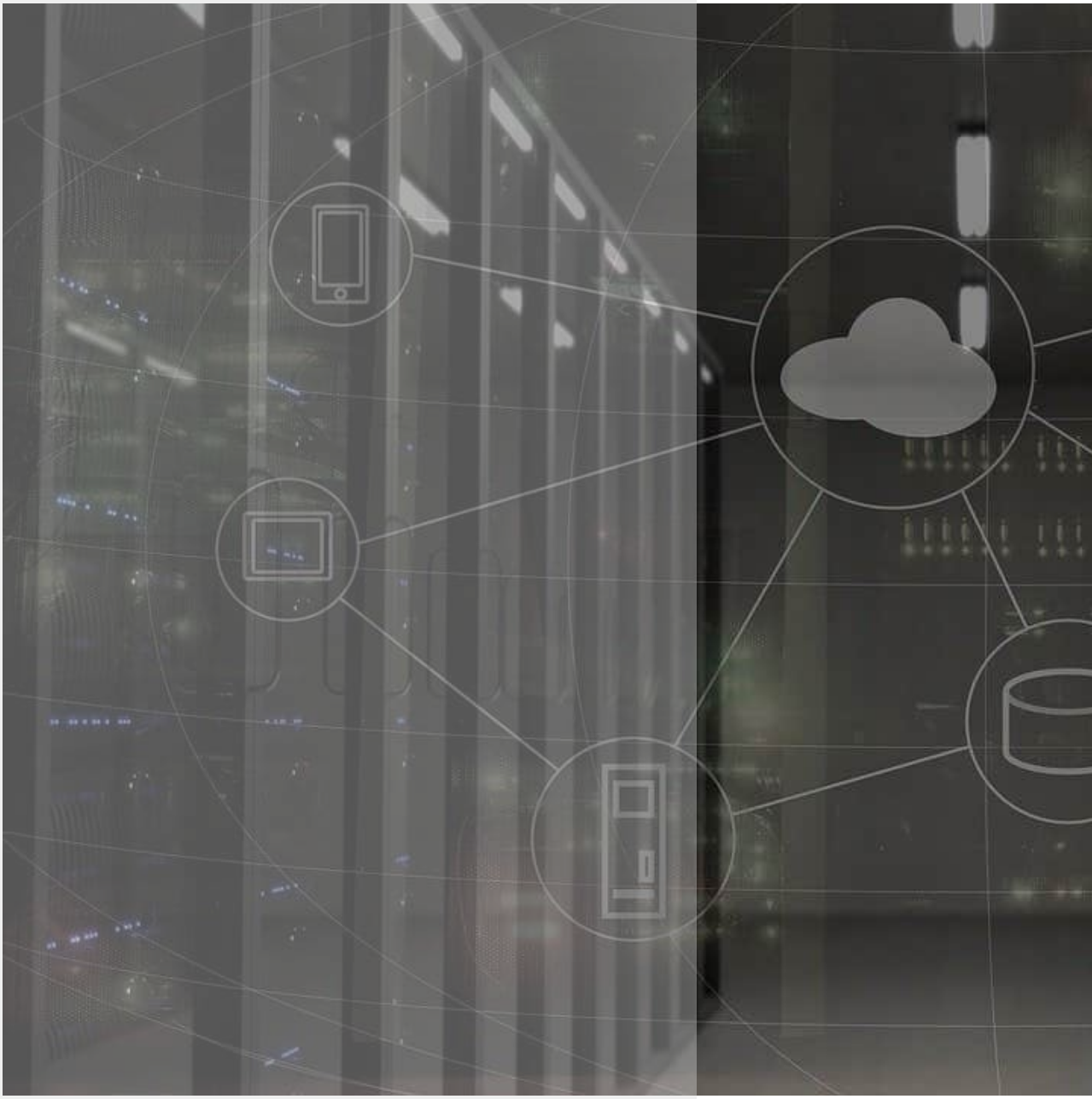
PROGRAMACIÓN

EJERCICIOS DE REPASO -FEBRERO-



INDICE

1 Búsqueda del buey

Búsqueda del buey (OX en inglés).

Crea un programa que rellene un array de 20 caracteres (no cadenas), rellenado con caracteres aleatorios. Los caracteres a elegir son ‘O’, ‘X’, ‘L’ y ‘B’. Con estos cuatro caracteres se deberá rellenar el array, conformando un array distinto para cada ejecución. Un ejemplo de array generado podría ser:

[B, B, B, X, L, X, L, L, O, X, O, L, O, O, B, O, B, X, O, L]

Después se deberá buscar si dentro de dicho array de 20 caracteres aparecen, en posiciones consecutivas, los símbolos O y X. Solo hay que buscar la primera aparición (si la hubiese). En el caso del ejemplo anterior estarían en la posición 8:

[B, B, B, X, L, X, L, L, O, X, O, L, O, O, B, O, B, X, O, L]

Para la búsqueda no se puede invocar a ningún método de ninguna clase (ni convertir el array a un String). Como resultado de la ejecución, el programa debería mostrar algo así:

[B, B, B, X, L, X, L, L, O, X, O, L, O, O, B, O, B, X, O, L]

OX encontrado en posición 8

O bien, si no aparece la secuencia OX:

[L, L, B, X, B, B, X, L, L, X, B, X, O, L, X, L, X, X, X, L]

OX no encontrado

Ampliación del ejercicio:

Una vez resuelto, intentar ampliar el ejercicio para que en lugar de indicar si se ha encontrado o no, se rellene un array de int donde se indique las posiciones en las cuales se ha encontrado algún "OX". Una vez finalice el programa se indicará cuántas veces ha sido encontrado y en qué posiciones (usando el array de posiciones que se habrá rellenado previamente). Por ejemplo, para este caso:

[X, X, O, X, X, O, L, X, O, B, O, O, X, X, O, O, O, X, X, B]

El resultado debería ser algo así:

OX ha sido encontrado. 3 veces.
En posición 2.
En posición 11.
En posición 16.

2 Conversor de nombres de números

Conversor de nombres de números.

Dado el siguiente programa que ya tiene implementada la entrada de datos y la salida de resultados, escribir el código necesario en Java para que se procese el array de entrada con números del 0 al 99 y se rellene un array de salida con los nombres en español de esos números.

```
public class NumeroAPalabra99 {
    /**
     * Programa principal
     *
     * @param argsthecommand line arguments
     */
publicstatic void main(String[] args) {

        //------------------------------------------------
        //          Declaración de variables
        //------------------------------------------------
        // Variables de entrada
int[] arrayEntrada = {0, 99, 10, 20, 15, 25, 66, 11, 7, 90, 72};

        // Variables de salida
        String[] arrayResultado;

        // Variables auxiliares
        String[][] arrayNombresNumeros = {
            {"cero", "uno", "dos", "tres", "cuatro", "cinco", "seis", "siete",
"ocho", "nueve"},
            {"diez", "once", "doce", "trece", "catorce", "quince", "dieciséis",
"diecisiete", "dieciocho", "diecinueve"},
            {"veinte", "veintiuno", "veintidós", "veintitrés", "veinticuatro",
"veinticinco", "veintiséis", "veintisiete", "veintiocho", "veintinueve"},
            {"treinta"},
            {"cuarenta"},
            {"cincuenta"},
            {"sesenta"},
            {"setenta"},
            {"ochenta"},
```

```
            {"noventa"}
        };
int contador;

        //-----------------------------------------------
        //                Entrada de datos
        //-----------------------------------------------
System.out.println("CONVERSOR DE NOMBRES DE NÚMEROS");
System.out.println("------------------------------");
System.out.println("Lista de números de prueba:");
System.out.println(Arrays.toString(arrayEntrada));

        //-----------------------------------------------
        //  Procesamiento (a implementar por el alumnado)
        //-----------------------------------------------



        //-----------------------------------------------
        //              Salida de resultados
        //-----------------------------------------------
System.out.println();
System.out.println("RESULTADO");
System.out.println("---------");
System.out.printf("Los nombres de esos números son:\n%s\n",
Arrays.toString(arrayResultado));

    }

}
```

El resultado que debe proporcionar el programa es el siguiente:

CONVERSOR DE NOMBRES DE NÚMEROS

Lista de números de prueba:
[0, 99, 10, 20, 15, 25, 66, 11, 7, 90, 72]
RESULTADO
RESULTADO

Los nombres de esos números son:
[cero, noventa y nueve, diez, veinte, quince, veinticinco, sesenta y seis, once, siete, noventa, setenta y dos]

Tan solo hay que implementar la parte del procesamiento así como declarar y utilizar todas las variables auxiliares que consideres oportunas. La entrada de datos y la salida de resultados están ya hechas y no hay que tocar nada.

En el procesamiento que implementes podrás hacer uso del array bidimensional arrayNombresNumeros que contiene los textos de los números necesarios para componer cada uno de los nombres de los números en castellano entre 0 y 99.

Debes aprovechar ese array para componer los cien posibles números:

1) Si el número es menor que 30, dispondrás directamente del nombre del número en el array arrayNombresNumeros.
2) A partir de 30, tendrás que componer el nombre usando decena y unidad. Recuerda que puedes obtener la decena de un número mediante la división entera y su unidad mediante el módulo o resto (el texto "y xxx", donde "xxx" sería la unidad, por ejemplo "setenta y dos" para 72). Y no olvides que si el número es una decena "pura" (por ejemplo 90), entonces no debes añadir la unidad (tan solo sería "*noventa*").

3 Clase compraVideojuego

Clase CompraVideojuego. Clase que modela una acción de compra en una tienda de videojuegos a partir de ticket. Se analizará una cadena de caracteres que contiene todos los datos de compra según un determinado formato y a partir de ahí se obtendrá cuántos juegos se han comprado para cada tipo de plataforma (PS5, SWITCH, XBOX) y cuánto se ha gastado para cada una de esas plataformas.

Te pedimos que implementes esta clase utilizando programación orientada a objetos. Es decir, creando la clase que modele el comportamiento del análisis y empleo del ticket con el siguiente diseño:

Nombre de la clase: CompraVideojuego

Atributos de objeto: (todos privados)

cantidadPS5	Cantidad de juegos comprados para la plataforma PS5. De objeto, privado, variable.
cantidadXbox	Cantidad de juegos comprados para la plataforma XBOX. De objeto, privado, variable.
cantidadSwitch	Cantidad de juegos comprados para la plataforma SWITCH. De objeto, privado, variable.
gastoPS5	Cantidad de dinero gastado en juegos para la plataforma PS5. De objeto, privado, variable.
gastoXbox	Cantidad de dinero gastado en juegos para la plataforma XBOX. De objeto, privado, variable.
gastoSwitch	Cantidad de dinero gastado en juegos para la plataforma SWITCH. De objeto, privado, variable.

Tendrás que implementar los siguientes métodos get:

1. **Método** intgetCantidad(String plataforma), que recibe como parámetro un String con el nombre de una plataforma (posibles valores “PS5”, “SWITCH” o “XBOX”) y devuelve la **cantidad de juegos comprados para esa plataforma**. Puedes hacerlo, por ejemplo, con una sentencia switch. Si no se proporciona ninguno de esos tres valores, la salida será cero.
2. **Método** doublegetGasto (String plataforma), que recibe como parámetro un String con el nombre de una plataforma (posibles valores “PS5”, “SWITCH” o “XBOX”) y devuelve la **cantidad de dinero gastado para esa plataforma**. No tiene en cuenta el IVA. Si no se proporciona ninguno de esos tres valores, la salida será cero.
3. **Método** publicdoublegetGastoTotal(), que devuelve la **cantidad total de dinero gastado en la compra**. No tiene en cuenta el IVA.

El único constructor que debes implementar tendrá un único parámetro:

publicCompraVideojuegos(String ticket)

En este caso se recibe un String con el ticket de compra que seguirá el siguiente formato:

- Cada **artículo** estará **separado** del siguiente por el **carácter almohadilla** (‘#’).
- Cada **artículo** estará compuesto por los siguientes **elementos**:
  - **Nombre** del vídeojuego.
  - **Plataforma** (posibles valores “PS5”, “SWITCH” o “XBOX”).
  - **Precio** (número real).
  - Cada **elemento** estará **separado** por un **carácter coma** (‘,’).

El constructor deberá analizar la cadena que se pasa como parámetro y extraer cada uno de los artículos, contabilizando cuántos artículos se han comprado para cada plataforma (actualización de los atributos de objeto cantidadPS5, cantidadXbox, cantidadSwitch) así como cuánto dinero se ha gastado para cada plataforma (actualización de los atributos de objeto gastoPS5, gastoXbox, gastoSwitch).

Para analizar y extraer toda esta información puedes utilizar las herramientas y utilidades que consideres oportunas y que se hayan visto hasta el momento (StringTokenizer, expresiones regulares, búsqueda y extracción “manual” con métodos de la clase String, etc.).

Este constructor **no lanzará ninguna excepción**. Si no contiene artículos válidos, simplemente los atributos del objeto tendrán valor cero.

Programa de pruebas (main)

Finalmente, habrá que implementar un programa principal (main) de pruebas que lleve a cabo las siguientes acciones:

1) Crear un objeto CompraVideojuegos usando el **único constructor disponible** con el siguiente ticket como parámetro:

   "GodofWar Ragnarok,PS5,66.10#FIFA 23,XBOX,57.84#NBA 2k23,XBOX,37.18#Mario Kart 8 Deluxe,Switch,41.31#Need ForSpeed Unbound,PS5,56.19"

2) Mostrar por pantalla **una línea para cada plataforma de videojuegos** (PS5, SWITCH, XBOX) indicando lo siguiente:
   - **cantidad de artículos** adquiridos para esa plataforma;
   - **nombre de la plataforma** (PS5, SWITCH, XBOX);
   - **gasto total** en artículos para esa **plataforma** (dos decimales);
   - **gasto total** en artículos para esa **plataforma** (dos decimales), con **IVA incluido.**
3) Finalmente, se mostrará una última línea donde se indique el **total de dinero gastado** en la **compra** (dos decimales) sin tener en cuenta el IVA primero, y después con el IVA incluido. **Recuerda que el precio con IVA incluido se calcula incrementando el valor en un 21%**.

Aquí tienes un ejemplo como debería quedar la salida de ejecución del programa de prueba una vez implementada la clase solicitada:

```
EJERCICIO 3. COMPRA DE VIDEOJUEGOS
----------------------------------
Creando objeto de compra con el ticket:
"GodofWar Ragnarok,PS5,66.10#FIFA 23,XBOX,57.84#NBA 2k23,XBOX,37.18#Mario Kart 8
Deluxe,Switch,41.31#Need ForSpeed Unbound,PS5,56.19"
2 Videojuego/s PS5 122,29€ (147,97€ IVA incluido)
1 Videojuego/s SWITCH 41,31€ ( 49,99€ IVA incluido)
2 Videojuego/s XBOX 95,02€ (114,97€ IVA incluido)
--------------------------------------------------
Total: 258,62€ (312,93€ IVA incluido)
```

Recuerda también que tanto la clase propiamente dicha como el programa de prueba (método main) están implementados ambos dentro de la clase *CompraVideojuegos*.

4 Implementación de la clase JuegoAdivinar

Implementa una clase JuegoAdivinar que modela un juego de adivinación de números, que generará un número secreto aleatorio y a la que se le podrá consultar si un número determinado es ese número secreto o no.

Nombre de la clase: JuegoAdivinar

Atributos de clase: (todos públicos y constantes)

LIMITE_MAX_VAL_POSIBLE = 40	Máximo valor posible para el número secreto al generarlo. No se pueden crear juegos con valor máximo superior a éste. De clase, público, constante.
LIMITE_MAX_INTENTOS = 10	Máximo número de intentos configurable para un juego. No se pueden crear juegos con un límite de intentos superior a este valor. De clase, público, constante.
DEFAULT_MAX_VAL_POSIBLE = 10	Valor por omisión (default) para el máximo valor posible para el número secreto al generarlo. Se utilizará si no se proporciona un valor para ello en un constructor. De clase, público, constante.
DEFAULT_LIMITE_MAX_INTENTOS = 5	Valor por omisión (default) para el máximo número de intentos configurable para un juego. Se utilizará si no se proporciona un valor para ello en un constructor. De clase, público, constante.

Atributos de objeto: (todos privados)

`numeroSecreto`	Número a averiguar. De objeto, privado, variable.
`intentosActuales`	Cantidad de intentos consumidos hasta el momento. De objeto, privado, variable.
`maxValorPosible`	Máximo valor posible para el número secreto al generarlo. De objeto, privado, constante.
`maxIntentos`	Cuántos intentos posibles hay para intentar averiguar el número. De objeto, privado, constante.

Tendrás que implementar únicamente los siguientes métodos get:

1) Método getIntentosRestantes, que devuelve la cantidad de intentos aún disponibles (es fácil de calcular a partir de la cantidad de intentos consumidos: se trataría de hacer una simple resta).
2) Método getMaxValPosible, que devuelve el máximo valor posible para el número secreto.

Los constructores que debes implementar son:

1) **Constructor con dos parámetros**, que creará un objeto juego de adivinar con un valor limite para el número aleatorio y con el máximo número de intentos pasados como parámetros:
   JuegoAdivinar (intmaxValorPosible, intmaxIntentos)
   Si algún parámetro no es válido, se lanzará una ***IllegalArgumentException*** con un mensaje apropiado: "*Límite de número secreto fuera de rango*" o bien "*Límite de intentos fuera de rango*". Para llevar a cabo esa comprobación, podrás utilizar los atributos constantes de clase que has tenido que definir. **El máximo valor para el número aleatorio no puede ser inferior a cero ni superior a la constante de clase LIMITE_MAX_VAL_POSIBLE**. En el caso del **máximo número de intentos debe ser superior a cero y no debe superar a la constante LIMITE_MAX_INTENTOS**. Si los parámetros son válidos, se generará un **número aleatorio entre 0 y el máximo valor posible** indicado.

2) **Constructor con un parámetro** que creará un objeto juego de adivinar con un máximo de intentos basado en el parámetro que se le pasa. Si el parámetro no es válido también habrá que lanzar una **excepción** como en el caso anterior.
   JuegoAdivinar (intmaxIntentos)
   El **máximo valor posible** será el definido en el **atributo público de clase** que indica el valor por omisión (DEFAULT_MAX_VAL_POSIBLE). Este constructor debe evitar la redundancia de código e invocar al constructor anterior.

3) **Constructor sin parámetros** que creará un objeto juego de adivinar donde el valor límite para el número aleatorio y el máximo número de intentos serán los valores por omisión declarados en los atributos públicos de clase de tipo DEFAULT. Nuevamente, para evitar la redundancia, este constructor debería emplear el constructor anterior. Cuanto menos código se repita, mejor. Este constructor no lanzará ninguna excepción, pues no se pueden producir errores con él.

Otros métodos de la clase:

boolean int adivinarNumero(int numero)

Recibe como parámetro el número que deseamos comprobar si es el número secreto generado. Se devolverán tres posibles valores:

valor **1**, si el número pasado como parámetro es **menor** que el número secreto;

valor **-1**, si el número pasado como parámetro es **mayor** que el número secreto;

valor **0**, si el número pasado como parámetro es **igual** al número secreto.

Obviamente, cada vez que se ejecute este método y se compruebe si un número es o no el secreto, se **incrementará el número de intentos consumidos** hasta el momento.

Programa de pruebas (main)

Finalmente, habrá que implementar un programa principal (main) de pruebas que lleve a cabo las siguientes acciones:

1) Intento de crear un objeto JuegoAdivinar inválido con los valores 10 y 41 (constructor de dos parámetros). Se debe capturar la excepción que se lance y mostrar un mensaje por pantalla basado en el método getMessage del objeto excepción recogido.
2) Intento de crear un objeto JuegoAdivinar inválido con el valor 11 (constructor de un parámetro). Se debe capturar la excepción que se lance y mostrar un mensaje por pantalla basado en el método getMessage del objeto excepción recogido.
3) Intento de crear un objeto JuegoAdivinar inválido con el valor 0 (constructor de un parámetro). Se debe capturar la excepción que se lance y mostrar un mensaje por pantalla basado en el método getMessage del objeto excepción recogido.
4) Crear un objeto JuegoAdivinar válido usando el constructor sin parámetros. Se debería crear con los valores 5 y 10.
5) Informar al usuario de los intentos que tiene así como del rango para adivinar.
6) Implementar un bucle en el que mientras queden intentos, se solicite al usuario un número para intentar adivinar el número secreto. Con ese número y utilizando el método adivinarNumero, se indicará al usuario uno de los siguientes mensajes: "¡Correcto!", "El número que buscas es menor que xxx", "El número que buscas es mayor que xxx", donde xxx sería el número introducido. El bucle finalizará cuando el usuario consiga adivinar el número o bien se consuman todos los intentos disponibles.

Aquí tienes un ejemplo como podría quedar la salida de ejecución del programa de prueba una vez implementada la clase solicitada:

EJERCICIO 3. ADIVINA EL NÚMERO

Intentando crear juego con valores erróneos: 10 y 41.

Error. Límite de número secreto fuera de rango: 41.

Intentando crear juego con valor erróneo: 11.

Error. Límite de intentos fuera de rango: 11.

Intentando crear juego con valor erróneo: 0.

Error. Límite de intentos fuera de rango: 0.

Comienza el juego

Tienes 5 intentos para averiguar el número entre 0 y 10.

Introduce número:

5

El número que buscas es mayor que 5

Introduce número:

7

El número que buscas es menor que 7

Introduce número:

6

¡Correcto!

Recuerda que tanto la clase propiamente dicha como el programa de prueba (método main) están implementados ambos dentro de la clase JuegoAdivinar.

5 Exploración numérica en arrays

Dados dos arrays de 10 elementos que se os proporcionan, escribir un programa en Java que cree un tercer array de tamaño el doble de los anteriores y con el siguiente contenido para su primera mitad:

- En cada posición impar se almacenará la suma de todos los elementos del primer array hasta esa posición.
- En cada posición par se almacenará la suma de todos los elementos del segundo array hasta esa posición.

Para la segunda mitad:

- En cada posición impar se almacenará el doble de la posición impar anterior de ese mismo array.
- En cada posición par se almacenará la suma de las dos posiciones pares anteriores de ese mismo array

NOTA: consideramos la posición 0 como par .

Por último, debe proporcionarse la suma de todos los números impares del array resultado.

Dado que los arrays se proporcionan y, por tanto, no hay entrada de datos, el resultado debería ser similar al siguiente:

```
EXPLORACIÓN NUMÉRICA EN ARRAYS
------------------
Array 1:        [1, 8, 3, 1, 3, 7, 5, 2, 4, 6]
Array 2:        [3, 4, 5, 6, 2, 5, 1, 7, 5, 5]

RESULTADO
---------
Array resultado: [3, 9, 12, 13, 20, 23, 26, 30, 38, 40, 64, 80, 102, 160, 166, 320, 268, 640, 434, 1280]
Suma de todos los números impares del array resultado: 48
```

6 Conversor de número convencional a número romano

Implementar un programa que reciba un conjunto de números entre 0 y 99 a partir de un array de *int* (podemos usar como base el ejercicio 2) y genere un **array de *String*** con esos números en su equivalente en "romano". El cero podríamos representarlo, por ejemplo, con un guion ("-").

Podemos usar como entrada la **misma batería de pruebas que en el ejercicio 2**:

int[] arrayEntrada = {0, 99, 10, 20, 15, 25, 66, 11, 7, 90, 72};

1) Primero lo podemos hacer usando la notación romana "fácil" o "simplificada". Se trata de una notación "aditiva" donde el 4 en lugar de ser "IV" es "IIII" (como en los relojes), el 9 sería "VIIII" en lugar de "IX", el 40 "XXXX" en lugar de "XL", el 90 "LXXXX" en lugar de "XC", etc. Esto simplificará bastante el algoritmo de generación del número romano.
   En tal caso la salida sería así:
   [-, LXXXXVIIII, X, XX, XV, XXV, LXVI, XI, VII, LXXXX, LXXII]

2) **Una vez logrado esto, podríamos saltar al siguiente paso y generar la notación romana convencional**. Aquí sí se utiliza la notación "sustractiva" para **evitar escribir más de tres veces seguidas el mismo símbolo** (IV para 4, IX para 9, XL para 40, XC para 90, etc.).
   En este caso la salida quedaría así:
   [-, XCIX, X, XX, XV, XXV, LXVI, XI, VII, XC, LXXII]

3) **Una vez conseguido esto, intentar hacerlo para cualquier número romano entre 0 y 3000**.

7 Validación de códigos (I)

Escribir un programa en Java para que se procese un array de entrada con números de serie y se rellene un array de salida indicando para cada número de serie si es válido o no (cadena “válido” o “inválido”).

Los requisitos que debe cumplir un número de serie para ser válido son:

1) Debe comenzar por dos letras mayúsculas. La primera obligatoriamente tiene que ser X o Y. La segunda puede ser cualquiera (sin contar la eñe).
2) A continuación habrá un guión (‘-’).
3) Ha de terminar por un año que debe encontrarse entre 1970 y el año actual.
4) Si el año es par, la primera letra del número de serie debe ser ‘Y’. Si el año es impar, la primera letra debe ser ‘X’.

La batería de números de serie de pruebas es:

[ZA-2000, XAZ2000, XA2000, XA-1969, YH-1969, XQ-1970, XJ-2002, YV-2021, XB-2022, YV-2042, YA-1970, YH-2002, XB-2021]

El resultado que debe proporcionar el programa es el siguiente:

VALIDADOR DE NÚMEROS DE SERIE

Lista de números de serie de prueba:

[ZA-2000, XAZ2000, XA2000, XA-1969, YH-1969, XQ-1970, XJ-2002, YV-2021, XB-2022, YV-2042, **YA-1970**, **YH-2002**, **XB-2021**]

RESULTADO

El resultado de la comprobación de la validez de los números de serie es:

[inválido, inválido, inválido, inválido, inválido, inválido, inválido, inválido, inválido, inválido, **válido**, **válido**, **válido**]

8 Códigos de guardia

En la sucursal de una empresa, los empleados han de hacer guardia un día de la semana, asignada según su código de identificación personal. Este código está compuesto por cuatro posiciones alfanuméricas, donde la primera indica el código de sucursal, seguidas de dos dígitos numéricos, que indican la edad del trabajador. En total seis posiciones.

Para la sucursal se ha decidido llevar a cabo la asignación de guardias a los empleados que cumplan las siguientes condiciones:

- La primera posición de su código de empleado ha de ser 1 (código de sucursal);
- La segunda posición ha de ser una vocal mayúscula, o una S o una D (también mayúsculas);
- El resto de posiciones alfanuméricas han de ser mayúsculas;
- El empleado ha de tener 20 años o más.

El día de guardia está determinado por la segunda posición del código (vocal), correspondiendo:
A : Lunes, E: Martes, I:Miércoles, O:Jueves, U:Viernes,
S o D : Fin de Semana completo
Si no se cumplen estas condiciones, el empleado no hará guardias.

Escribe un programa en Java que determine el día de guardia / fin de semana que le toca o, en caso de no hacer guardia, que indique que no debe hacerla, para la siguiente relación de códigos de empleado: "3ABC36", "1EDE27", "1UWX19", "1DEF32", "1AB45", "1STU45", "1aBC28", "1ABC31"

El resultado de la asignación debería ser:

```
ASIGNACIÓN DE GUARDIAS

======================

El empleado  3ABC36          No trabaja

El empleado  1EDE27Trabaja el MARTES

El empleado  1UWX19          No trabaja

El empleado  1DEF32Fin de Semana

El empleado  1AB45   No trabaja

El empleado  1STU45Fin de Semana

El empleado  1aBC28 No trabaja

El empleado  1ABC31          Trabaja el LUNES
```

9 Implementación de la clase número

Debemos implementar en Java una clase llamada *Numero*, que "envuelve" o "encapsula" un número entero de dos cifras como máximo (entre 0 y 99):

- Los objetos instancias de esta clase serán inmutables.
- Atributo de objeto numérico (*int*) para el número que "envolvemos", con su correspondiente *getNumero()*, que devolverá ese *int*.
- Constructor que comprueba que el número está en el rango. Si no, se genera una excepción.
- Atributos estáticos públicos constantes para los rangos mínimo y máximo (0 y 99).
- Método *toString*() que genera la representación textual del número en castellano.
- Varios métodos "*toString*" que generan representaciones textuales en otros idiomas:

1) toStringSpanish();
2) toStringEnglish();
3) toStringSwahili();
4) toStringFrench();
5) toStringGerman();
6) Y lo generalizamos con un método parametrizado donde se indique el idioma: toStringLanguage (enumLanguagelang). De esa manera se puede hacer un bucle desde el programa principal para ir llamando a cada idioma disponible. Habrá que definir un enum público llamado Language, fuera de la clase Numero (mejor incluso en otro archivo), que defina todos los posibles idiomas que se pueden enviar al método toStringLanguage .

- Añadimos atributos estáticos con la configuración auxiliar:
    - Un array por cada idioma (o bien un array con una dimensión más) con la tabla de los números en forma de palabra en cada idioma, tal y como ya se hizo en el ejercicio 2 y su ampliación. Privados.
- Métodos para obtener el número en romano (usando lo que ya se ha hecho en algún ejercicio anterior):

    1) *toStringRomanoFacil()* (método "aditivo fácil": IIII en lugar de IV, VIIII en lugar de IX, etc.);
    2) *toStringRomano()*.

- Método toStringBinary(), que devuelve una cadena con la representación en binario del número. Utilizar las herramientas de la clase "envoltorio" *Integer*.

Prueba de la clase Numero

Una vez hayamos implementado la clase, implementar un método *main* de prueba que recorra todos los números desde 0 hasta 99, cree un objeto *Numero* con ese valor y muestre por pantalla la siguiente información en columnas, usando los métodos de la clase:

Número (int) - Número en binario - Número romano "fácil" - Número romano - Número español - Número inglés - Número francés - Número - swahili - Número alemán

10 Clase JuegoAhorcado

Implementar una clase llamada *JuegoAhorcado* que implemente una versión simplificada del conocido juego del ahorcado. La clase deberá contener:

1) Un array privado de *String* con unas 20 palabras con tamaños entre 4 y 10 letras.
2) Atributos constantes, públicos y de clase que indiquen el mínimo (4) y máximo (10) tamaño de las palabras posibles.
3) Un atributo de objeto con la palabra secreta elegida cada vez que se inicie el juego (se cree un objeto instancia de la clase).
4) Algún mecanismo estático que permita que cada vez que se cree un objeto instancia de la clase, la palabra secreta sea diferente. Cuando ya se haya "gastado" más del 80% de las palabras, entonces podrán volverse a usar todas de nuevo, comenzando otra vez a "eliminar" palabras.
5) Un constructor que recibirá como parámetro el tamaño máximo de la palabra aleatoria que se va a elegir, así como el máximo número de intentos que se van a permitir para el juego. Si ese tamaño máximo ya no es posible (porque se hayan gastado las palabras), entonces se elegirá cualquier otra palabra disponible. Si se proporciona un valor fuera de los rangos posibles, se generará una excepción.
6) Un método llamado "*probar*", que enviará una palabra y se indicará si la letra está en la palabra o no (*true* o *false*). Cada vez que se llame a este método, quedará un intento menos. Cuando no queden intentos, este método lanzará una excepción *IllegalStateException*.
7) Los siguientes métodos gettter:
    a. *getIntentosRestantes()*, que devolverá cuantos intentos nos quedan aún;
    b. *getLetrasRestantes()*, que devolverá cuántas letras quedan por adivinar.
8) Un método *toString* que devolverá la palabra "enmascarada" donde solo se observarán las letras que ya se han encontrado. Las que aún no se hayan encontrado serán caracteres del tipo '_'.

    Por ejemplo "_ A B _ C _ _ A" para "CABECERA", si ya se han encontrado las letras A, B, C. También se indicarán cuántos intentos quedan disponibles.

Una vez dispongamos de la clase, implementar un método *main()* de prueba que genere un número aleatorio entre 4 y 10 y cree un objeto del tipo *JuegoAhorcado* con ese número aleatorio como tamaño máximo de palabra y con 12 intentos. El programa tendrá que intentar "adivinar" la palabra a base de ir probando con letras aleatorias (sin repetirse) hasta que consiga adivinar la palabra o bien se le acaben los intentos disponibles. Para cada paso (cada letra aleatoria), se irá indicando por pantalla en qué estado se encuentra el juego (mediante el método *toString*). El programa no debe tener ningún tipo de intervención humana. Debe "jugar" solo.

11 Clase Molécula

Implementar una clase llamada *Molecula*con las siguientes propiedades y comportamientos:

1) Un array con los símbolos químicos de cada átomo que compone la molécula (*String[]*);
2) Un array con la cantidad de átomos que hay de cada tipo (*int[]*).
3) Un constructor que recibirá un *String* con la fórmula química de la molécula. Por ejemplo, "H2O", "NaCl" o "H2SO4". La estructura de la fórmula de una molécula estará siempre compuesta por uno o varios de los siguientes elementos:
    a. Un símbolo químico de un átomo (una o dos letras, la primera siempre en mayúsculas).
    b. Un número, que puede o no aparecer (el número de veces que aparece ese átomo en la molécula). Si no hay número, es que ese átomo aparece una única vez. El número puede ser como máximo 99 y no puede ser 0.
    c. Si el símbolo químico de un átomo aparece más de una vez en la fórmula, o se detecta alguna otra incongruencia en la fórmula (empieza por un número o hay algún número superior a 99) se lanzará una excepción y no se creará el objeto.
4) Un método *String[] getListaAtomos()*, que devolverá un array con los símbolos de los átomos que componen esa molécula. El tamaño de ese array será exactamente de la cantidad de átomos diferentes queforman la molécula.
5) Un método *intgetAtomo (String atomo)*, que recibirá como parámetro un *String* indicando un átomo (su símbolo químico: "H", "Na", "S", "O", "Cl", etc.) y devolverá la cantidad de átomos que hay de su especie en la molécula. Por ejemplo, para "H2SO4", *getAtomo("H")* debería devolver 2, *getAtomo("S")* debería devolver 1 y *getAtomo("O")* debería devolver 4. Para cualquier otra entrada debería devolver 0.
6) Un método *intgetNumAtomosDistintos()*, sin parámetros, que devolverá la cantidad de átomos diferentes que participan en la formación de la molécula. Por ejemplo, para "H2SO4" debería devolver 3 (participan tres átomos: hidrógeno, azufre y oxígeno).
7) Un método *intgetNumAtomosTotales()*, sin parámetros, que devolverá la cantidad de átomos en total que contiene la molécula. Por ejemplo, para "H2SO4" debería devolverse 7 (dos hidrógenos, un azufre y cuatro oxígenos).
8) Un método *toString()* que, a partir de la información contenida en el objeto (no se almacena un *String* con la fórmula), vuelva a generar un texto con la fórmula de la molécula (con los átomos en el mismo orden en el que fue proporcionada en el constructor).

Una vez implementada la clase, escribir un método *main()* de prueba que solicite al usuario una fórmula de una molécula, se genere un objeto de tipo *Molecula*y se muestre por pantalla toda la información posible sobre esa molécula a partir de los métodos que ofrece la clase.

12 Tabla de días de la semana por mes

Implementa un programa en Java que solicite por teclado un año y a continuación rellene un **array bidimensional de enteros de tamaño 12x7** que contenga, para cada mes (fila), cuántos días de la semana contiene de cada tipo (cuántos lunes, cuántos martes, cuántos miércoles, etc.).

Una vez relleno ese array, mostrarlo por pantalla en forma de tabla.

Por ejemplo, si se introdujera por teclado el año 2022, el resultado que debería mostrarse por pantalla sería el siguiente:

```
AÑO 2022

--------

      L M X J V S D

Mes  1  5 4 4 4 4 5 5

Mes  2  4 4 4 4 4 4 4

Mes  3  4 5 5 5 4 4 4

Mes  4  4 4 4 4 5 5 4

Mes  5  5 5 4 4 4 4 5

Mes  6  4 4 5 5 4 4 4

Mes  7  4 4 4 4 5 5 5

Mes  8  5 5 5 4 4 4 4

Mes  9  4 4 4 5 5 4 4

Mes 10  5 4 4 4 4 5 5

Mes 11  4 5 5 4 4 4 4

Mes 12  4 4 4 5 5 5 4
```

Eso significa que en enero hay cinco lunes, cuatro martes, cuatro miércoles, etc.