

1ºDAM

PROJECTO PRESENTATION PROGRAMME

KEVIN MUÑOZ CORCOLES



INDICE

1. INTRODUCCIÓN

**2. ESTRUCTURA
GENERAL DEL CÓDIGO**

**4. LÓGICA DEL
ALGORITMO**

**5. INTERACCIÓN CON
EL USUARIO**

**6. PRUEBAS
UNITARIAS (JUNIT)**

**7. DOCUMENTACION
JAVADOC**

8. POSIBLES MEJORAS / JUEZ

9. CONCLUSION



1. INTRODUCCIÓN: NÚMEROS AFORTUNADOS (146)

- Objetivo del programa: calcular los llamados números afortunados a partir de un número N, aplicando un proceso de eliminación.
- Idea principal: se genera una lista de 1 a X numero y se van eliminando posiciones siguiendo reglas sucesivas (primero cada 2, luego cada 3, etc.) hasta que no se puede continuar.
- Resultado: los números que quedan son los afortunados, que se muestran en orden descendente.



2. ESTRUCTURA GENERAL DEL CÓDIGO

El programa está organizado en una clase principal Números Afortunados.

Contiene varios métodos estáticos que dividen el problema en pasos claros:

- Crear el vector inicial. (CrearVector)
- Elimina elementos en posiciones múltiples del paso (incluyendo el primero) (EliminarCadaPaso)
- Compactar resultados.(compactar)
- Controlamos el numero de veces que eliminamos según reglas (quitarNúmero)
- Validar la entrada del usuario.(filtro)

```
public static int Filtro(int numero) { 1 usage
    Scanner entrada = new Scanner(System.in);
    do {
        System.out.print("Introduce un número entre 1 y 10000 (0 para salir): ");
        String linea = entrada.nextLine();

        try {
            numero = Integer.parseInt(linea);
            if (numero == 0) { //si es cero nos da igual porque el 0 es cerrar el programa
                break;
            } else if (numero < 1 || numero > 10000) { //queremos que sea el numero de este rango de numeros
                System.out.println("Número fuera de rango. Debe estar entre 1 y 10000.");
                numero = -1;
            }
        } catch (NumberFormatException e) { //si el que mete el numero de casualidad o saber porque mete una letra se lo invalidamos
            System.out.println("Entrada inválida. Solo se permiten números enteros.");
            numero = -1;
        }
    } while (numero == -1); //hacemos que el bucle siempre se repita haciendo que en caso de que meta algo erroneo el numero sea -1

    return numero;
}
```

```
Rename usages
public class NumerosAfortunados {
    public static int[] crearVector(int numero) { 1 usage
        int[] vector = new int[numero]; //Crea la anchura del vector
        for (int i = 0; i < numero; i++) { // mete los numeros en orden
            vector[i] = i + 1;
        }
        return vector;
    }

    public static int[] eliminarCadaPaso(int[] vector, int paso) { 1 usage
        int longitud = vector.length; //tomamos la longitud
        int[] nuevo = new int[longitud]; //creamos el nuevo vector que utilizaremos
        int contador = 0; //contamos cuantos espacios tendra que tener el nuevo vector

        for (int i = 0; i < longitud; i++) {
            if (i % paso != 0) { // eliminamos el primero y luego cada 'paso' (indices múltiplos de
                nuevo[contador] = vector[i]; //colocamos solo los numeros que queramos en posicion
                contador++; //contamos el espacio
            }
        }
        return compactar(nuevo, contador); //antes de entregarlo lo compactamos llamando a la funci
    }

    public static int[] compactar(int[] vector, int contador) { 1 usage
        int[] resultado = new int[contador]; //Creamos el vector final
        for (int i = 0; i < contador; i++) {
            resultado[i] = vector[i]; //metemos los numeros hasta llenar el resultado, controlado por el contador
        }
        return resultado;
    }

    public static int[] quitarNumeros(int[] vector) { 1 usage
        int longitud = vector.length;
        for (int paso = 2; paso <= longitud; paso++) {
            vector = eliminarCadaPaso(vector, paso); // usamos la funcion eliminarCadaPaso para eliminar los numeros de los indices multiples de 'paso'
            longitud = vector.length; //guardamos la nueva longitud del vector
        }
        return vector;
    }
}
```



3. ESTRUCTURA GENERAL DEL CÓDIGO

```
public static void main(String[] args) {
    while (true) {
        int numero = Filtro(numero: -1); //Filtramos el numero usando la funcion filtro
        if (numero == 0) {
            System.out.println("Programa terminado.");
            break;
        }
        int[] vector = crearVector(numero); //usamo la funcion crear vector para crar el vector que necesitamos

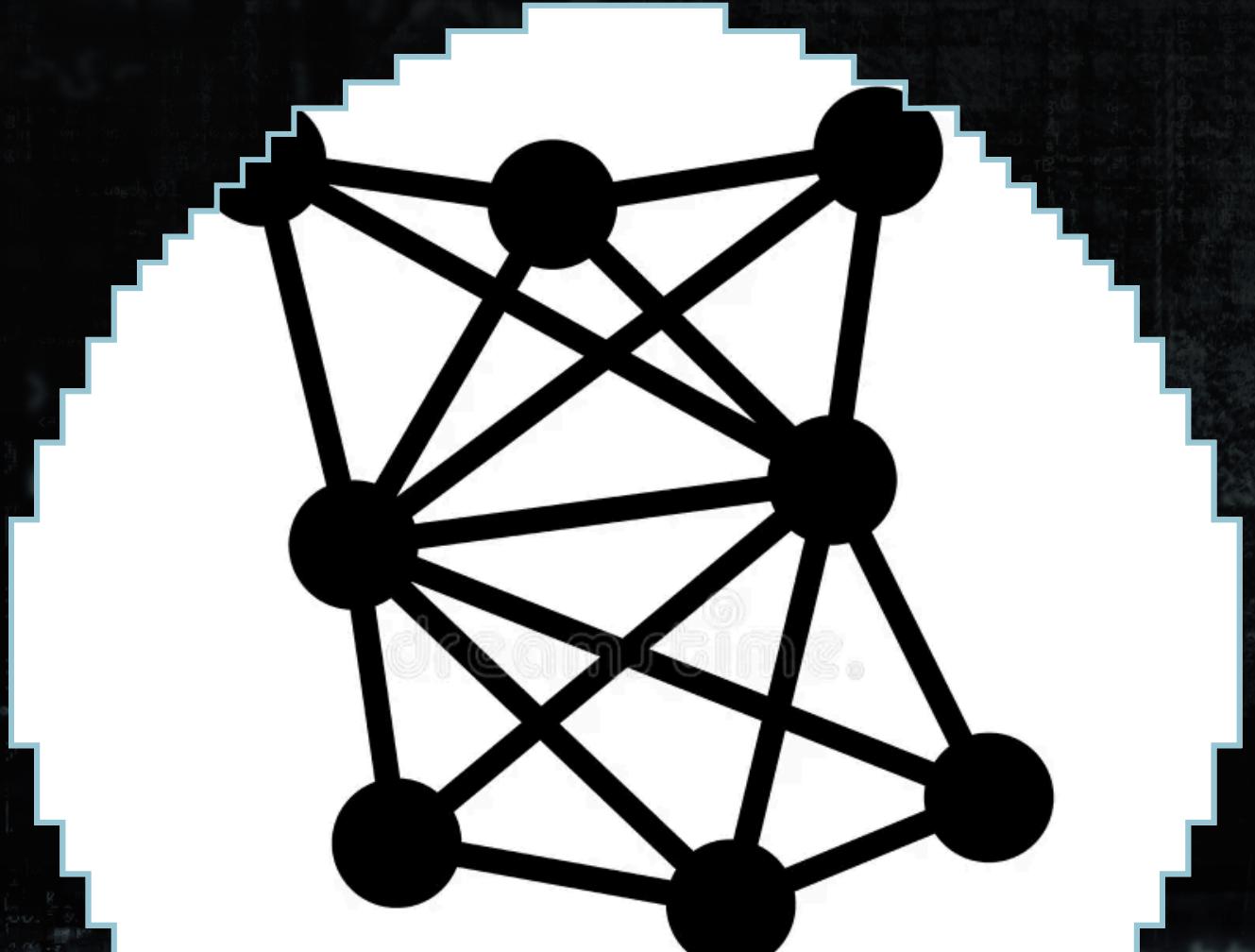
        // enseñar el vector que hemos creado :3
        System.out.print("Vector inicial: ");
        for (int num : vector) {
            System.out.print(num + " ");
        }
        System.out.println();

        // eliminar los numeros que tengamos que eliminar :^/ (modo termineitor)
        int[] resultado = quitarNumeros(vector); //Usamos la funcion quitar numeros

        // Mostrar resultado final °W°
        System.out.print("Números afortunados: ");
        for (int i = resultado.length - 1; i >= 0; i--) {
            System.out.print(resultado[i] + " ");
        }
        System.out.println();
    }
}
```



4. LÓGICA DEL ALGORITMO



Algorithm

1. Se parte de un vector con todos los números.
2. En cada iteración:
 - Se elimina el primero y luego cada número de paso.
 - Se compacta el vector para eliminar huecos.
 - Se incrementa el paso.
3. El proceso termina cuando el número de elementos restantes es menor que el paso actual.
4. Los números que quedan son los afortunados.



5. INTERACCIÓN CON EL USUARIO



- El programa pide números por consola.
- El usuario puede introducir varios casos de prueba.
- El número 0 termina la ejecución.
- Se muestran mensajes de validación si la entrada es incorrecta.
- La salida se presenta en formato: X: lista_de_afortunados_en_descendente



6. PRUEBAS UNITARIAS (JUNIT)

TEST CREAR VECTOR

comprobamos que al meter un vector de 5 numeros salga el vector relleno con los 5 numeros

```
@Test
void testCrearVector() {
    int[] vector = bonito.crearVector( numero: 5); // crear vector de 5
    assertArrayEquals(new int[]{1, 2, 3, 4, 5}, vector); // debe salir [1,2,3,4,5]
}
```

TEST ELIMINAR CADA PASO

valida que se eliminan las posiciones correctas.

```
@Test
void testEliminarCadaPaso() {
    int[] vector = {1, 2, 3, 4, 5, 6}; // vector inicial
    int[] resultado = bonito.eliminarCadaPaso(vector, paso: 2); // eliminamos cada 2 empezando por el primero
    assertArrayEquals(new int[]{2, 4, 6}, resultado); // deben quedar los pares
}
```



6. PRUEBAS UNITARIAS (JUNIT)

TEST COMPACTAR

asegura que los ceros se eliminan y el array queda limpio.

```
@Test
void testCrearVector() {
    int[] vector = bonito.crearVector( numero: 5); // crear vector de 5
    assertArrayEquals(new int[]{1, 2, 3, 4, 5}, vector); // debe salir [1,2,3,4,5]
}
```

```
@Test
void testEliminarCadaPaso() {
    int[] vector = {1, 2, 3, 4, 5, 6}; // vector inicial
    int[] resultado = bonito.eliminarCadaPaso(vector, paso: 2); // eliminamos cada 2 empezando por el primero
    assertArrayEquals(new int[]{2, 4, 6}, resultado); // deben quedar los pares
}
```



7. DOCUMENTACION JAVADOC

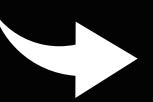
CADA MÉTODO ESTÁ DOCUMENTADO CON:

- @param: descripción de parámetros.
- @return: descripción del valor devuelto.
- Comentarios explicativos del propósito y funcionamiento.
- Esto permite generar documentación HTML automática con javadoc.

```
Rename usages
/**
 * @Author Kevin Muñoz Corcoles
 * @deprecated implementa la generación y filtrado de números afortunados.
 * Contiene métodos para crear vectores, eliminar elementos según reglas,
 * compactar resultados y aplicar la salida de números afortunados (mayor a menor).
 *
 */
public class NumerosAfortunados { new *

    /**
     * @param numero - Pasamos el numero para crear un vector con el tamaño de ese numero
     * @return vector - devolvemos el vector creado
     */

}
```



7. DOCUMENTACION JAVADOC

The screenshot shows a JavaDoc interface with the following elements:

- Header:** PACKAGE (highlighted in orange), CLASS, USE, TREE, DEPRECATED, INDEX, HELP.
- Search Bar:** SEARCH: Search X
- Breadcrumb:** PACKAGE: DESCRIPTION | RELATED PACKAGES | CLASSES AND INTERFACES
- Title:** Package org.example
- Content:** package org.example
- Class List:** Classes (highlighted in orange), Class, Description
- Table:** A table showing one class:

Class	Description
NumerosAfortunados	Deprecated. implementa la generaciÃ³n y filtrado de nÃºmeros afortunados.



8. POSIBLES MEJORAS / JUEZ

JUEZ

El juez no acepta el programa porque tarda demasiado tiempo en hacerse, dando como resultado Run-time error.

SOLUCIONES O MEJORAS ENCONTRADAS

Usar ArrayList por las siguientes mejoras:

- Tamaño dinámico: se ajusta automáticamente al añadir o eliminar elementos.
- Eliminación directa: puedes usar `remove(index)` sin necesidad de crear un nuevo array.
- Menos código: no necesitas métodos como compactar, porque la lista ya gestiona huecos.
- Mayor legibilidad: el código se simplifica y se centra en la lógica del algoritmo.

```
QUE EL JUEZ ACEPTE

Copiar código

public class Main {
    // Precomputar valores hasta MAXN = 10000
    static final int MAXN = 10000;
    static int[] respuesta = new int[MAXN + 1];

    public static void precomputar() {
        // Generamos lista inicial de 1..MAXN
        ArrayList<Integer> lista = new ArrayList<>();
        for (int i = 1; i <= MAXN; i++) lista.add(i);

        int pasoIndex = 1; // Empezamos en paso=2 (índice 1)

        // criba de números afortunados
        while (pasoIndex < lista.size()) {
            int paso = lista.get(pasoIndex);
            if (paso > lista.size()) break;

            // Eliminar cada 'paso'
            for (int i = lista.size() - 1; i >= 0; i--) {
                if ((i + 1) % paso == 0) lista.remove(i);
            }

            pasoIndex++;
        }

        // Para cada N, buscamos el mayor afortunado <= N
        int idx = 0;
        for (int n = 1; n <= MAXN; n++) {
            while (idx + 1 < lista.size() && lista.get(idx + 1) <= n)
                idx++;

            // El mayor afortunado menor o igual a n
            lista.get(idx) <= n ? respuesta[n] = lista.get(idx);
        }
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        System.out.println(respondida(n));
    }
}
```

Fecha 12/12/2025, 17:48:02 (UTC)

Lenguaje del envío Java

Veredicto Run-time error (RTE)

Comentario
(0/1000)

Comentario de este envío para diferenciarlo de otros. Al igual que el código, sólo será visible por ti.

Guardar

9.CONCLUSION

El programa implementa correctamente el algoritmo de números afortunados y demuestra cómo una estructura modular facilita la comprensión y el mantenimiento del código. Además, la documentación y las pruebas unitarias aportan fiabilidad y aseguran que cada parte funcione como se espera. En conjunto, constituye un buen ejemplo de cómo dividir un problema en pasos claros y verificables, logrando una solución ordenada, eficiente y fácil de seguir.

El único problema encontrado fue la aceptación del juez de la página acepta el reto, por el tiempo de ejecución, El cual se arregla usando arraylist y se vuelve un problema mas sencillo.



THANKYOU

