UD04.Estructuras de almacenamento Prácticas

DAM1-Programación 2024-25

Agenda **2**
Exportar e Importar Agenda 2
Buscaminas **4**
Wordle **6**
Ejercicios de examen **7**
Arrays y estadísticas 7
Conceptos de programación 8
Buscando letras 9
Registro de tiempos 10
Matrices triangulares 10
Abadías pirenaicas 13
Matrículas 14
Montaña Rusa 16
Conjugar verbos 17
Recorrido Robot 19

Agenda

Implementa una agenda que permita almacenar los datos de contacto (nombre, apellidos, teléfono, correo electrónico) de personas conocidas.

El programa presentará repetidamente un menú con las siguientes opciones:

1. **Añadir contacto:** solicitará por teclado la información de una persona y lo añadirá a la agenda.
2. **Listar contactos:** mostrará un listado resumido de los contactos en la agenda, cada uno en una línea, indicando un índice de contacto.
3. **Eliminar contacto:** Solicitará un número de contacto y, si es válido, lo eliminará de la agenda.
4. **Vaciar agenda:** Eliminará todos los contactos de la agenda.
5. **Buscar contacto:** Solicitará por teclado el nombre del contacto y, si se encuentra, mostrará el resto de la información. Si existe más de un contacto con el mismo nombre?
6. **SALIR [0]**

Exportar e Importar Agenda

- Librería Gson
    a. Descargar .jar

Implementa estas funciones para poder guardar y cargar el contenido de la agenda en un fichero de texto en el disco duro.

DAM1 - Programación con JSON

- Descarga y utiliza la librería Gson para:
    a. exportar el contenido de la agenda a un String JSON
    b. importar en el array de objetos los datos de una agenda en formato String/JSON

```
import com.google.gson.Gson;

// Exportar Persona[] a JSON
        Gson gson = new Gson();
        Persona[] p = new Persona[];
        //…
        String json = gson.toJson(p);



// Importar JSON en Persona[]
        Gson gson = new Gson();
        Persona[] p;

        String json = ""; // contenido JSON;
        p = gson.fromJson(json, Persona[].class);
```

- Guardar y leer JSON a y desde un archivo de texto en disco

```java
/**
 * Crea un fichero de texto con el contenido de un String
 * @param str
 * @param filePath
 */
public static void writeStringToFile(String str, String filePath) {
    try {
      // Creamos un objeto FileWriter que nos permitirá escribir en el fichero
      FileWriter writer = new FileWriter(filePath);

      // Escribimos el String en el fichero
      writer.write(str);

      // Cerramos el fichero
      writer.close();
    } catch (IOException e) {
      e.printStackTrace();
    }
  }

/**
 * Lee y carga el contenido de un fichero de texto a un String
 * @param filePath
 * @return
 */
public static String readFileToString(String filePath) {
    StringBuilder fileContent = new StringBuilder();
    try {
        // Creamos un objeto FileReader que nos permitirá leer el fichero
        FileReader reader = new FileReader(filePath);

        // Creamos un buffer para leer el fichero de forma más eficiente
        BufferedReader buffer = new BufferedReader(reader);

        // Leemos el fichero línea a línea
        String line;
        while ((line = buffer.readLine()) != null) {
            // Vamos añadiendo cada línea al StringBuilder
            fileContent.append(line);
            // Añadimos un salto de línea al final de cada línea
            fileContent.append("\n");
        }

        // Cerramos el buffer y el fichero
        buffer.close();
        reader.close();
    } catch (IOException e) {
        System.out.println("No existe el fichero.");
        //e.printStackTrace();
    }

    // Devolvemos el contenido del fichero como un String
    return fileContent.toString();
}
```

Buscaminas

Buscaminas.java
En el juego del buscaminas se ocultan un número fijo de minas en un tablero bidimensional. Durante el juego el jugador podrá destapar casillas del tablero y marcar aquellas en las que cree que se encuentran minas ocultas.
El jugador pierde si destapa una casilla en la que se oculta una mina. En caso contrario el programa mostrará la casilla destapada y todas aquellas adyacentes en las que no se encuentren minas, indicando para cada una de estas casillas cuántas minas están tocando (en horizontal, vertical o diagonal).
El jugador gana cuando marca correctamente aquellas casillas en las que se ocultan las minas o cuando destapa todas las casillas del tablero excepto las minas.

Para hacerlo más complicado/competitivo se puede añadir un temporizador, o un registro del tiempo invertido en resolver el tablero.

Análisis del problema

1. Utilizaremos constantes para definir las dimensiones del tablero y el número de minas ocultas.

2. Cómo almacenaremos la ubicación de las minas? En una matriz bidimensional de datos booleanos, tableroMinas, que contenga true en las casillas que ocultan minas y false en el resto.

3. Cómo almacenaremos el tablero que se muestra al jugador? Otra matriz bidimensional, tableroJuego, de tipo caracter, que contenga guiones '-' en las casillas sin destapar, la letra eme 'M' en las casillas marcadas como minas y, en las casillas destapadas, el número de minas adyacentes a esa casilla.

Ejemplo de posible programa principal:

```
public class Buscaminas {
    // Declaración de variables y constantes de clase
    static final int FILAS = 5;
    static final int COLUMNAS = 5;
    static final int NUM_MINAS = 4;
    static boolean[][] tableroMinas = new boolean[FILAS][COLUMNAS];
    static char[][] tableroJuego = new char[FILAS][COLUMNAS];
    static int numMarcas = 0; // número de minas marcadas por el jugador
    // Condiciones de salida del juego
    static boolean opcionSalir = false;    // El jugador introdujo la opción de Salir?
    static boolean minaDestapada = false;  // El jugador destapó una mina?
    static boolean minasMarcadas = false;  // El jugador marcó correctamente todas las minas?

    public static void main(String[] args) {
        // Declaración de variables y constantes
        int opcion, fila, columna;

        iniciarTableroMinas();
        iniciarTableroJuego();
```

```
        do {
            imprimirTableroJuego();
            //imprimirTableroMinas(); // Trampa => BORRAR
            imprimirMenu();
            switch (opcion = leerOpcion()){
                case 0:
                    opcionSalir = true;
                    break;
                case 1: // Destapar Casilla
                    fila = leerFila();
                    columna = leerColumna();
                    destaparCasilla(fila, columna);
                    break;
                case 2: // Marcar Casilla
                    fila = leerFila();
                    columna = leerColumna();
                    marcarCasilla(fila, columna);
                    break;
                case 3: // Desmarcar Casilla
                    fila = leerFila();
                    columna = leerColumna();
                    desmarcarCasilla(fila, columna);
                    break;
            }
        }while (!opcionSalir && !minaDestapada && !minasMarcadas);

        imprimirFinJuego();
    }
// ...
}
```

Ejemplos de salida por consola:

```
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0: - - - - - - - - - - - - - - - - - - - -
1: - - - - - - - - - - - - - - - - - - - -
2: - - - - - - - - - - - - - - - - - - - -
3: - - - - - - - - - - - - - - - - - - - -
4: - - - - - - - - - - - - - - - - - - - -
5: - - - - - - - - - - - - - - - - - - - -
6: - - - - - - - - - - - - - - - - - - - -
7: - - - - - - - - - - - - - - - - - - - -
8: - - - - - - - - - - - - - - - - - - - -
9: - - - - - - - - - - - - - - - - - - - -
0: - - - - - - - - - - - - - - - - - - - -
1: - - - - - - - - - - - - - - - - - - - -
2: - - - - - - - - - - - - - - - - - - - -
3: - - - - - - - - - - - - - - - - - - - -
4: - - - - - - - - - - - - - - - - - - - -
5: - - - - - - - - - - - - - - - - - - - -
6: - - - - - - - - - - - - - - - - - - - -
7: - - - - - - - - - - - - - - - - - - - -
8: - - - - - - - - - - - - - - - - - - - -
9: - - - - - - - - - - - - - - - - - - - -

Menú
1. Destapar casilla
2. Marcar mina
3. Desmarcar mina
0. SALIR
```

```
Opción: 1
Fila: 5
Columna: 5
   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2: 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
3: 0 0 0 0 1 - 1 0 0 0 0 0 0 0 0 0 0 0 0 0
4: 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 1 1 0 0 0
5: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 - 1 0 0 0
6: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
7: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8: 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
9: 0 0 0 0 0 0 0 0 0 1 - 1 0 0 0 0 0 0 0 0
0: 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
5: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
6: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
7: 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
8: - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
9: - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
Opción: 2
Fila: 18
Columna: 10

   0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
0: - - - - - - - - - - - - - - - - - - - -
1: - - - - - - - - - - - - - - - - - - - -
2: - - - - - - - - - - - - - - - - - - - -
3: - - - - - - - - - - - - - - - - * - - - -
4: - - - - - - - - - - * - - - - - - - - - -
5: - - - - - - - - - - - - - - - - - - - -
6: - - - - - - - - - - - - - - - - - - - -
7: - - - - - - - - - - - - - - - - - - - -
8: - - - - - - - - - - - - - - - - - - - -
9: - - - - - - - - - - - - - - - - - - - -
0: - - - - - - - - - - * - - - - - - - - - -
1: - - - - - - - - - - - - - - - - - - - -
2: - - - - - - - - - - - - - - - - - - - -
3: - - - - - - - - - - - - - - - - - - - -
4: - - - - - - - - - - - - - - - - - - - -
5: - - - - - - - - - - - - - - - - - - - -
6: - - - - - - - - - - - - - - - - - - - -
7: - - - - - - - - - - - - - - - - - - - -
8: - - - - - - - - - - * - - - - - - - - - -
9: - - - - - - - - - - - - - - - - - - - -

ENHORABUENA!!! HAS DESCUBIERTO TODAS LAS MINAS!!!
```

Wordle

Wordle.java
A finais do ano 2021 fíxose moi popular o xogo Wordle, que consistía en adiviñar una palabra oculta de 5 letras. Investiga o funcionamento do xogo.

- Wordle en galego: https://wordlegalego.appspot.com/

Realiza a túa propia adaptación do Wordle para xogar en consola coas seguintes especificacións adicionais:

1. O programa disporá dun array/colección de palabras.
2. *Opcional:* Ao arrancar, o programa buscará na carpeta do programa/proxecto un ficheiro de nome “palabras.json” coa colección inicial de palabras. Se non existe tal ficheiro, o programa iniciará a colección cunha única palabra, por exemplo “XOGAR” (ou “JUGAR”).
3. En cada execución o programa proporá unha palabra oculta de 5 letras e o xogador terá 6 intentos para adiviñala.
4. Tras cada intento o programa amosará as letras que está na palabra oculta e na posición correcta, as que está pero en posición incorrecta e as que non están.
5. Adicionalmente, se a palabra utilizada polo xogador non está na colección de palabras do programa ofrecerase a posibilidade de engadila á colección para xogar no futuro.
6. O xogo remata cando o xogador acerta a palabra ou agota os 6 intentos, amosando unha mensaxe apropiada.
7. *Opcional:* Antes de rematar a execución do programa gardarase en disco a colección actual de palabras no ficheiro “palabras.json” sobreescribíndoo se é necesario.
8. Para almacenar as palabras define unha clase Palabra que teña os seguintes atributos e métodos:
    a. a palabra en sí como un String.
    b. o número de veces que a palabra apareceu no xogo como palabra oculta.
    c. o número de veces que se acertou a palabra como palabra oculta.
    d. o número de veces que a utilizou un xogador para tentar adiviñar a palabra oculta.
    e. estatisticas(): amosa a palabra e os números anteriores.
    f. toString(): amosa a palabra.

Constructores, setters e getters axeitados.

Ejercicios de examen

Arrays y estadísticas

ArraysEstadisticas.java
3/12/2021
Crea las siguientes funciones que operan con un array de números enteros positivos que reciben como parámetro de entrada:

`static int suma(int[] numeros)`
Devuelve la suma de los elementos del array.

`static int minimo(int[] numeros)`
Devuelve el valor mínimo contenido en el array.

`static int maximo(int[] numeros)`
Devuelve el valor máximo contenido en el array.

`static double media(int[] numeros)`
Devuelve la media aritmética de los elementos del array, es decir, el promedio del conjunto de valores que contiene.

`static double mediana(int[] numeros)`
Devuelve la mediana de los elementos del array. La mediana en un conjunto de datos ordenados es aquel valor que ocupa la posición central. El array de entrada no se presupone ordenado, por lo que para realizar el cálculo de la mediana será necesario ordenarlo previamente. En caso de que el número de elementos del array sea par y no hay un único elemento central sino dos, la mediana será la media aritmética de los dos valores centrales.

`static int moda(int[] numeros)`
Devuelve la moda de los elementos del array. La moda es el valor que aparece con mayor frecuencia en un conjunto de datos, es decir, el que más veces se repite. En caso de que el array de entrada contenga dos o más modas, es decir, dos o más valores que se repitan el mismo número de veces, la función devolverá el valor -1.

Prueba las funciones utilizando el siguiente programa principal (`main`) junto con la función `imprimeEstadisticas()`:

```
public static void main(String[] args) {
    // Declaración de variables y constantes
    int[] numeros1 = {2, 4, 6, 7, 4, 7, 9, 2, 5, 6, 7};
    int[] numeros2 = {2, 4, 6, 7, 4, 9, 2, 5, 6, 7};
```

```
    // Proceso y salida
    imprimeEstadisticas(numeros1);
    imprimeEstadisticas(numeros2);
}

static void imprimeEstadisticas(int[] numeros) {
    System.out.println("ARRAY ORIGINAL: " + Arrays.toString(numeros));
    System.out.println("-------------------------------------------");
    System.out.println("Longitud: " + numeros.length);
    //System.out.println("Suma: " + suma(numeros));
    //System.out.println("Máximo: " + maximo(numeros));
    //System.out.println("Mínimo: " + minimo(numeros));
    //System.out.println("Media: " + media(numeros));
    //System.out.println("Mediana: " + mediana(numeros));
    //System.out.println("Moda: " + moda(numeros));
    Arrays.sort(numeros);
    System.out.println("ARRAY ORDENADO: " + Arrays.toString(numeros));

System.out.println("-------------------------------------------\n\n");
}
```

La salida por pantalla del programa completo deberá coincidir con la siguiente:

```
ARRAY ORIGINAL: [2, 4, 6, 7, 4, 7, 9, 2, 5, 6, 7]
-------------------------------------------
Longitud: 11
Suma: 59
Máximo: 9
Mínimo: 2
Media: 5.363636363636363
Mediana: 6.0
Moda: 7
ARRAY ORDENADO: [2, 2, 4, 4, 5, 6, 6, 7, 7, 7, 9]
-------------------------------------------


ARRAY ORIGINAL: [2, 4, 6, 7, 4, 9, 2, 5, 6, 7]
-------------------------------------------
Longitud: 10
Suma: 52
Máximo: 9
Mínimo: 2
Media: 5.2
Mediana: 5.5
Moda: -1
ARRAY ORDENADO: [2, 2, 4, 4, 5, 6, 6, 7, 7, 9]
-------------------------------------------
```

Conceptos de programación

ConceptosProgramacion.java
Los siguientes términos corresponden a conceptos que el alumnado de programación en Java debe conocer:

```
    "Algoritmo",
```

```
"Lenguaje de programación",
"Entorno de desarrollo",
"Compilación",
"Ejecución",
"Código fuente",
"Bytecode",
"Código objeto"
```

Realiza un programa que almacene los términos anteriores en un array de datos de tipo String y que muestre uno de ellos por pantalla elegido al azar.

Para ello implementa una función que devuelva uno de los términos del array elegido al azar, con el siguiente prototipo:

```
static String fraseAleatoria(String[] t)
```

Buscando letras

BuscandoLetras.java
La siguiente función (método de la clase String) permite convertir un String en un array de caracteres (elementos de tipo char):

```
public char[] toCharArray()
```

Ejemplo de uso:

```
String cadena = "Hola Mundo";
char[] tablaChar = cadena.toCharArray();
```

Crea una función que reciba una cadena de caracteres y una letra como parámetros de entrada y que devuelva un array con las posiciones en que aparece dicha letra en la cadena. La numeración de posiciones comenzará en el 0, y si no se encuentra la letra la función devolverá un array vacío. Utiliza el siguiente prototipo:

```
static int[] buscarLetra(String cadena, char letra)
```

Por ejemplo,

- `buscarLetra("Hola Mundo", 'o')` devolverá el array `[1, 9]`.
- `buscarLetra("Hola Mundo", 'e')` devolverá el array `[]`.

Amplía el programa “Conceptos de programación” para solicitar repetidamente una letra por teclado al usuario e indicar si existe o no en el término elegido al azar, cuantas veces aparece y en que posiciones.
La clase Scanner de Java no dispone de un método **nextChar()** para leer caracteres. En su lugar, para leer una letra de teclado puedes usar el método **next()** de la clase Scanner para leer un String y acceder al carácter de la primera posición de la siguiente forma:

```
char letra = sc.next().charAt(0);
```

El bucle de entrada de letras terminará al introducir una letra que no se encuentre.

Registro de tiempos

RegistroTiempos.java
Crea un programa que registre y almacene los segundos transcurridos desde el inicio del programa hasta cada vez que se pulsa la tecla Enter. La entrada de tiempos finalizará cuando se introduzca la letra “F” o “f”, contabilizando también esta entrada en el registro.

Imagina que se trata del cronómetro utilizado por un profesor de Educación física para cronometrar los segundos que tarda el alumnado en dar 5 vueltas al campo.

Al terminar la introducción de datos, el programa imprimirá los tiempos recogidos (en segundos), calculará la media y contabilizará el número de tiempos registrados por encima de la media, imprimiendo los resultados.

Ejemplos:

1. Se inicia el programa y el usuario pulsa Enter al cabo de 3, 7, y 12 segundos, e introduce una “F” 20 segundos después de iniciado el programa.
   El programa imprimirá lo siguiente:

   [3, 7, 12, 20]
   Media = 10.5
   Número de registros por encima de la media: 2

2. Se inicia el programa y el usuario introduce una “F” al cabo de 5 segundos.
   El programa imprimirá lo siguiente:

   [5]
   Media = 5
   Número de registros por encima de la media: 0

Notas:

- Usa el método now() de la clase LocalTime para obtener la hora del sistema.
- Puedes usar el método equalsIgnoreCase() de la clase String.

Matrices triangulares

Inspirado en Matrices triangulares - ¡Acepta el reto!

MatricesTriangulares.java
Una matriz cuadrada, es decir que tiene el mismo número de filas que de columnas, es *triangular* cuando *todos* los valores que están por encima o por debajo de la diagonal principal

son cero. También son triangulares aquellas matrices que cumplen estas dos condiciones a la vez.

$$T_3 = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 6 & 4 \\ 0 & 0 & 5 \end{bmatrix}$$

Implementa la siguiente función que indique si una matriz es triangular:

```
public static Boolean esTriangular(int[][] t);
```

Consideraciones adicionales:

- el método debe tener un control adecuado sobre matrices nulas o vacías, devolviendo false en esos casos.
- Las matrices no cuadradas no son triangulares, por lo que deberían devolver false.

Puedes usar la siguiente clase de pruebas Junit5

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class MatricesTriangulares {

    // Método estático que estamos probando
    public static Boolean esTriangular(int[][] t) {
        // TODO: Implementa la función
    }

    // 1. Matriz triangular superior
    @Test
    public void testMatrizTriangularSuperior() {
        int[][] matrizTriangularSuperior = {
            {1, 2, 3},
            {0, 4, 5},
            {0, 0, 6}
        };
        assertTrue(esTriangular(matrizTriangularSuperior));
    }

    // 2. Matriz triangular inferior
    @Test
    public void testMatrizTriangularInferior() {
        int[][] matrizTriangularInferior = {
            {1, 0, 0},
            {4, 5, 0},
            {7, 8, 9}
        };
        assertTrue(esTriangular(matrizTriangularInferior));
    }

    // 3. Matriz no triangular (con elementos fuera de la diagonal)
    @Test
    public void testMatrizNoTriangular() {
        int[][] matrizNoTriangular = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };
        assertFalse(esTriangular(matrizNoTriangular));
    }

    // 4. Matriz de tamaño 1x1 (triangular trivial)
    @Test
    public void testMatriz1x1() {
        int[][] matriz1x1 = {
            {1}
        };
```

```
        assertTrue(esTriangular(matriz1x1));
    }

    // 5. Matriz vacía => false
    @Test
    public void testMatrizVacia() {
        int[][] matrizVacia = {};
        assertFalse(esTriangular(matrizVacia));
    }

    // 6. Matriz con valores nulos (null) = false
    @Test
    public void testMatrizNula() {
        int[][] matrizNula = null;
        assertFalse(esTriangular(matrizNula));
    }

    // 7. Matriz rectangular (no cuadrada)
    @Test
    public void testMatrizRectangular() {
        int[][] matrizRectangular = {
            {1, 2},
            {3, 4},
            {5, 6}
        };
        assertFalse(esTriangular(matrizRectangular));
    }

    // 8. Matriz triangular superior con elementos cero en la diagonal
    @Test
    public void testMatrizTriangularSuperiorCero() {
        int[][] matrizTriangularSuperiorCero = {
            {0, 2, 3},
            {0, 0, 5},
            {0, 0, 0}
        };
        assertTrue(esTriangular(matrizTriangularSuperiorCero));
    }

    // 9. Matriz triangular inferior con elementos cero en la diagonal
    @Test
    public void testMatrizTriangularInferiorCero() {
        int[][] matrizTriangularInferiorCero = {
            {0, 0, 0},
            {4, 0, 0},
            {7, 8, 0}
        };
        assertTrue(esTriangular(matrizTriangularInferiorCero));
    }

    // 10. Matriz triangular superior con valores fuera de la diagonal principal
    @Test
    public void testMatrizIncorrecta() {
        int[][] matrizIncorrecta = {
            {1, 0, 3},
            {0, 2, 0},
            {0, 0, 4}
        };
        assertFalse(esTriangular(matrizIncorrecta));
    }
}
```

Abadías pirenaicas

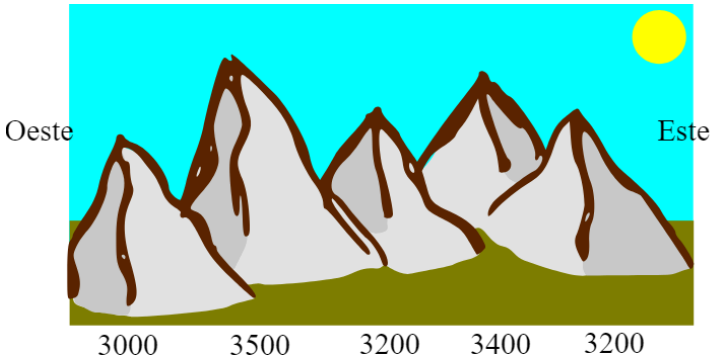
Inspirado en Abadías pirenáicas - ¡Acepta el reto!

AbadiasPirenaicas.java

Es bien sabido que los monjes, en las abadías, se levantan muy temprano para rezar durante *maitines*, observando la salida del sol.

La localización de las abadías y monasterios es por tanto de vital importancia para que se pueda observar el sol aparecer en el horizonte. Los monjes prefieren las localizaciones que no tengan montañas altas hacia el Este, que les impedirían ver salir el sol.

Esto es un problema especial en los Pirineos, la cordillera montañosa que va de Oeste a Este y separa la Península Ibérica del resto del continente europeo. Los Pirineos están llenos de lugares tranquilos y mágicos, por lo que son emblemáticos para la meditación y el rezo. Pero al estar orientados de Oeste a Este, tienen una limitación en el número de localizaciones candidatas a construir abadías, es decir las que reciben el sol en los primeros momentos de la mañana. Este problema lo sufren también otras cordilleras en todo el mundo. Es por eso que los monjes sólo construyen abadías en aquellas montañas que non tengan montañas más altas al este que les puedan hacer sombra durante la mañana.



Realiza un programa que permita introducir por teclado las alturas (en metros, sin decimales) de las montañas que componen una determinada cordillera montañosa, de Oeste a Este. La entrada terminará cuando se introduzca una altura negativa o se llegue al límite de 1000 montañas.

Sólo se pueden construir abadías en las cimas que *sean más altas* que todas las montañas que tengan al Este.

El programa debe indicar el número máximo de abadías que se pueden construir en esa cordillera montañosa y adicionalmente el listado de índices de montañas en las que se pueden construir (comenzando en 1 para la montaña situada más al Oeste).

Por ejemplo, para una entrada que represente la cordillera de la imagen anterior:

> Entrada: 3000 3500 3200 3400 3200 -1

Se obtendrían los siguientes resultados:

> Salida: Se pueden construir 3 abadías en las cordilleras 2, 4 y 5.

Implementa el método `static int[] getMontanasAbadias(int[] MontanasCordillera);`

Matrículas

Matriculas.java
Las matrículas de los automóviles en España están compuestas por dos partes. La primera consiste en un número decimal de 4 dígitos que se va incrementando de uno en uno. Cuando se llega al último valor (9999), se reinicia la cuenta (en 0000) y se incrementa la segunda parte, que contiene letras.

Esta segunda parte consiste en tres letras consecutivas, excluyendo las vocales y las letras Ñ y Q. Cuando, tras recorrer todos los números, se incrementa la letra, se pasa a la siguiente en el abecedario en la letra situada más a la derecha. Si para ella se acaba el abecedario, se vuelve al principio (B) y se incrementa la anterior con este mismo procedimiento.

Implementa y prueba las siguientes dos funciones:

`boolean esMatriculaValida(String matricula);`
Devuelve true si el String que se le pasa como parámetro es una matrícula válida o false en caso contrario. La función no distinguirá entre mayúsculas y minúsculas.

Ideas:

- Podemos utilizar una variable de tipo bandera/interruptor/boolean que inicialice a true, suponiendo que la matrícula es correcta y que cambie a false si detectamos alguna condición para que no lo sea.
- Validar longitud.
- Para los 4 primeros caracteres validamos si es un dígito. `Character.isDigit()`?
- Para los 3 últimos caracteres validamos que sea una letra válida (indiferente minúscula o mayúscula). Módulo: `boolean esLetraValida(char ch);`?

`String siguienteMatricula(String matricula);`
Devuelve el valor de la matrícula siguiente a la que se le pasa como parámetro.

Ideas:

- Invocamos el método anterior para validar la matrícula de entrada.
- Podemos implementar un módulo para incrementar los dígitos: `String incrementarDigitos(String matricula);`
- Si los dígitos han dado la vuelta: `matricula.substring(0, 4).equals("0000")` podemos implementar otro módulo para incrementar las letras: `String incrementarLetras(String matricula);`

Ejemplos de casos de prueba

Matrícula	**esMatriculaValida()**	**siguienteMatricula()**

`123`
`1234BBBB`
`12A4BBB`
	false
	false
	false	

12.4BBB 1234BAB 1234BQB 1234BB. 1234B5B 1234BúB 1234BÚB 1234BñB	false false false false false false false false	
1234BbB 1234BBB 9999BBZ 9999BBD 9999BBP 9999ZZZ	true true true true true true	1235BBB 1235BBB 0000BCB 0000BBF 0000BBR 0000BBB

Comparar matrículas por antigüedad:

```
int comparaMatriculas(String m1, String m2);
```

Devuelve 0 si son iguales, un valor menor que 0 (por ejemplo -1) si m1 es más antigua que m2, y un valor mayor que 0 (por ejemplo 1) si m2 es más antigua que m1).

Ejemplos de casos de prueba

m1	m2	return
1234BBB 1234BBB 1234BBC 1234BBB 1234BBB	1234BBB 1234BBC 1234BBB 1233BBB 1235BBB	0 -1 1 1 -1

Montaña Rusa

MontanaRusa.java
En una montaña rusa, llamamos "pico" a un punto del recorrido que está más alto que el inmediatamente anterior y el inmediatamente siguiente.

Implementa la siguiente función que acepta como parámetro de entrada un array con el registro de las alturas en metros del recorrido de una montaña rusa y devuelve el número de picos.

```
int contarPicos(int[] alturas);
```

Ten en cuenta que las montañas rusas son circulares, y el punto de inicio de la entrada ¡podría ser un pico!

Ejemplos:

- `[4, 10, 3, 2] => 1 pico`
- `[10, 3, 2, 4] => 1 pico`
- `[4, 10, 10, 3, 2] => 0 picos`
- `[0, 1, 2, 3, 2, 3, 4, 4, 4, 4, 3, 4, 5, 2, 0] => 2 picos`

Tests JUnit5

```
@Test
void testContarPicos_CasoBase() {
    int[] alturas = { 1, 2, 3, 2, 1 };
    assertEquals(1, contarPicos(alturas));
}

@Test
void testContarPicos_MultiplesPicos() {
    int[] alturas = { 1, 2, 3, 2, 1, 0, 1, 2, 3, 2, 1 };
    assertEquals(2, contarPicos(alturas));
}

@Test
void testContarPicos_PicoAlFinal() {
    int[] alturas = { 1, 2, 3, 4, 5 };
    assertEquals(1, contarPicos(alturas));
}

@Test
void testContarPicos_PicoEnInicio() {
    int[] alturas = { 5, 4, 3, 2, 3, 4, 4 };
    assertEquals(1, contarPicos(alturas));
}

@Test
void testContarPicos_MontañaCircularSinPicos() {
    int[] alturas = { 3, 2, 1, 2, 3 };
    assertEquals(0, contarPicos(alturas));
}

@Test
void testContarPicos_MontañaCircularConVariosPicos() {
    int[] alturas = { 3, 2, 1, 2, 1, 2, 1, 2, 3 };
    assertEquals(2, contarPicos(alturas));
}
```

Inspirado en [Siete picos - ¡Acepta el reto!]

Conjugar verbos

ConjugarVerbos.java

En español, los verbos regulares se conjugan siguiendo patrones predecibles en cada tiempo verbal. Hay tres conjugaciones principales según la terminación del verbo en infinitivo: -ar, -er, y -ir.

A continuación tienes un ejemplo de cómo se conjugan estos verbos en presente para cada una de las personas verbales:

Personas verbales / Pronombres / Presente	1ª conjugación (-ar) hablar	2ª conjugación (-er) comer	3ª conjugación (-ir) vivir
Yo	hablo	como	vivo
Tú	hablas	comes	vives
Él/Ella/Usted	habla	come	vive
Nosotros/Nosotras	hablamos	comemos	vivimos
Vosotros/Vosotras	habláis	coméis	vivís
Ellos/Ellas/Ustedes	hablan	comen	viven

1. (2) Implementa una función que dado un verbo en infinitivo lo conjugue en presente y devuelva un array de String con los resultados de la conjugación para cada una de las 6 personas verbales.

```
String[] conjugarPresente(String verbo);
```

Si el parámetro `verbo` no tiene una terminación válida la función devolverá `null`.

2. (1) Crea un programa principal que acepte por teclado verbos en infinitivo y muestre por pantalla su conjugación en tiempo presente, añadiendo al inicio los pronombres adecuados. Si el verbo introducido no corresponde a un infinitivo con una terminación adecuada el programa mostrará el mensaje de error correspondiente.

El programa finalizará cuando se introduzca la palabra “fin”.

3. (1) Implementa una nueva función que permita conjugar un verbo en distintos tiempos verbales aceptando como parámetros el verbo en infinitivo y el tiempo verbal:

```
String[] conjugarTiempoVerbal(String verbo, String tiempoVerbal);
```

Los valores válidos de tiempos verbales válidos para la función son:

1. "Presente simple"
2. "Pretérito perfecto simple"
3. "Futuro simple"

Cualquier otro valor de tiempo verbal hará que la función devuelva `null`, del mismo modo que si el parámetro `verbo` no tiene una terminación válida.

A continuación tienes un ejemplo de cómo se conjugan los verbos anteriores en "Pretérito perfecto simple" y "Futuro simple":

Tiempos verbales	1ª conjugación (-ar) hablar	2ª conjugación (-er) comer	3ª conjugación (-ir) vivir
Pretérito perfecto simple	hablé hablaste habló hablamos hablásteis hablaron	comí comiste comió comimos comísteis comieron	viví viviste vivió vivimos vivísteis vivieron
Futuro	hablaré hablarás hablará hablaremos hablaréis hablarán	comeré comerás comerá comeremos comeréis comerán	viviré vivirás vivirá viviremos viviréis vivirán

Inspirado en Conjugar verbos - ¡Acepta el reto!

Recorrido Robot

RecorridoRobot.java
El siguiente array codifica el mapa de un juego:

```
String[] mapa = {
    "  Z       ",
    " *        ",
    "  *  *    ",
    "          ",
    " A        "
};
```

- la “A” representa la casilla de salida
- la “Z” representa la casilla de llegada
- los asteriscos “*” representan minas que no se deben pisar.

El juego consiste en mostrar el mapa y a continuación solicitar al usuario que introduzca por teclado una única cadena de caracteres con las instrucciones que debe seguir un robot para llegar desde la casilla de salida hasta la de llegada sin pisar ninguna mina ni salirse del tablero.

Al inicio del juego el robot está situado en la casilla de salida y orientado hacia *arriba*.
La cadena de instrucciones que se le da al robot puede tener estos 3 caracteres distintos:

- una “A” hará que el robot avance una casilla en la dirección en la que está orientado.
- una “R” hará que la orientación del robot gire a la derecha 90º sin moverse del sitio.
- una “L” hará que la orientación del robot gire a la izquierda 90º sin moverse del sitio.

Implementa una función que reciba como parámetros de entrada un mapa en el formato anterior y una cadena de caracteres con las instrucciones para el robot y que devuelva *true* si tras ejecutar las instrucciones el robot termina en la casilla de llegada y *false* si el robot se pasa de largo, o si pisa alguna mina o si sale del tablero, o si las instrucciones contienen caracteres no válidos, o en cualquier otro caso.

Utiliza el siguiente prototipo:

```
static boolean recorridoRobot(String[] mapa, String instrucciones)
```

Los siguientes son algunos ejemplos de cadenas de instrucciones y sus resultados para el mapa proporcionado al inicio del ejercicio:

1. “AALARAARAA” => True: el robot alcanza la casilla de llegada.
2. “RAALAAAALA” => True: el robot alcanza la casilla de llegada.
3. “ARALA” => False: el robot pisa una mina.
4. “LAA” => False: el robot se sale del mapa.

El mapa que se pasa a la función podrá tener cualquier tamaño pero se supondrá que es correcto, es decir, que tiene una única “A”, una única “Z” y cero, una o varias minas.