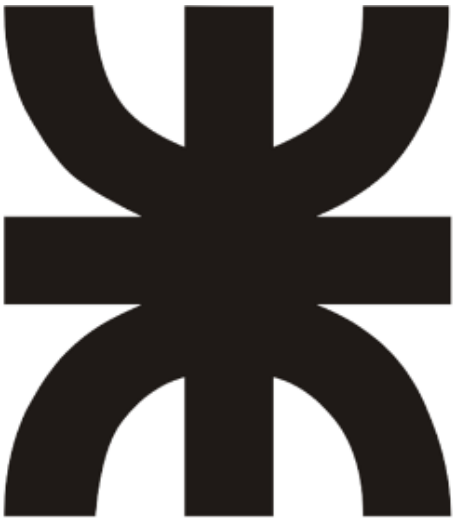


Universidad Tecnológica Nacional – Facultad Regional Villa María

Ingeniería en Sistemas de Información

Paradigmas de programación

Trabajo Práctico Evaluativo

Greenfoot – 24/09/2020

Docente:

Rinaldi, Mario

Integrantes:

Álvarez, Joaquín - Arias, Matías - Ortiz, Lucas

Año 2020

2- (60 pts.) Sobre el escenario del Laboratorio de Newton, se deben realizar las siguientes tareas (**grupal - colaborativo**):

A: Describir las principales características de los 3 Newton Labs (funcionamiento, clases, objetos, métodos, etc.)

B: A partir del Newton Lab3, incorporar una modificación **significativa** a elección.

En la presentación de la solución de este punto deberá incluirse:

- una memoria descriptiva del cambio a realizar.

- imágenes, figuras, fotos, etc. que documenten los cambios realizados.

- un pequeño video comentando el resultado obtenido

A:

Escenario Newton – lab 1:

El funcionamiento que tiene el primer Newton lab a simple vista es prácticamente nulo, solo podemos ver un escenario que representa el espacio mediante una imagen, y podemos agregar un objeto llamado Body mediante características ingresadas por el programador o agregar un nuevo Body con características default. Las clases que se encuentran son:

- **World**: superclase que contiene la clase:
  - Space: La clase space es quien marca el escenario, trata de un fondo que representa al espacio. En la clase space podemos encontrar métodos como super () que indica el tamaño del escenario o métodos como sunAndPlanet (), sunAndTwoPlanets (), sunPlanetMoon () pero están comentados asique no realizan ninguna acción en el escenario.

- **Actor**: superclase que contiene la clase:
  - SmothMover: Es una clase que contiene bastantes métodos, pero no se pueden ver en funcionamiento en el escenario, métodos que nos sirven para que la clase Body pueda desplazarse en nuestro escenario, la clase SmothMover Contiene:

  - Body: Clase que puede crear un nuevo objeto, que tiene parámetros somo el tamaño, el color y el parámetro vector, que posee la funcionalidad para que luego pueda desplazarse, esta clase posee un tamaño de Body, tamaño de vector y color por default, pero pueden ser elegidos por el usuario/programador gracias al método Body (int, size, double, mass, Vector velocity, Color color).

Escenario Newton – lab 2:

Dentro del escenario newton – lab2, podemos observar dentro de lo visual una pantalla con diferentes tonos de azul que dan la idea de espacio. Existen dos superclases llamadas Word y la otra superclase es actor que a su ves contiene a las clases SmothMover en la cual se especifican todos los movimientos que tienen los objetos en cuanto a dirección, velocidad y sentido teniendo en cuenta una clase separada llamada vector en donde se retornan la mayoría de las variables implementadas. La otra clase perteneciente a Actor es Body, que cuenta con diferentes métodos explicados a continuación:

- **Body** (int size, double mass, Vector velocity, Color color): Este método es un constructor que recibe como parámetros el tamaño del objeto, la masa, la velocidad y el color. Y con todos estos datos, aplicando los diferentes métodos utilizados se crea el objeto.

- **applyForces** (): Este método se encarga de aplicarle la gravedad a todos los cuerpos que se encuentran en el universo, llamando a otro método llamado applyGravity explicado en el próximo ítem.

- **applyGravity** (Body other): Este método recibe como parámetro un objeto de tipo Body, y su función es: primero mide la distancia entre dos cuerpos en los ejes “x” y “y” con el método getExactX (); y luego igual con “y”, después almacena esos valores en dos variables, que se utilizan para crear un vector que va a determinar la dirección de los cuerpos a medida que se van acercando entre sí. Se calcula también a la distancia total mediante el teorema de Pitágoras, aplicando una función matemática (Math.sqrt). Esa distancia va a ser utilizada para calcular la fuerza que aplica la gravedad sobre los cuerpos. Esto se da ya que newton dice que la fuerza (en Newton) es igual a la multiplicación de la constante G y las dos masas (en kilogramos), dividida por el cuadrado de la distancia (en metros), y en eso se basa el escenario. Además de todo esto también se calcula la aceleración, que esta dada entre la división de la fuerza sobre la masa de un cuerpo. Con todos estos datos se modifican las características del movimiento de cada cuerpo después de la acción de la gravedad.

- **Setter**: Estos métodos se utilizan para cargarle valores a una variable de instancia (métodos mutadores).

- **Getter**: Son utilizados para acceder a una variable protegida (private).

Con respecto a la ejecución, se puede observar que si se crea un nuevo objeto tiene un leve movimiento hacia la derecha, que es interrumpido al chocar con una pared, pero al crear dos objetos al mismo tiempo ocurre que tienen atracción entre ellos a medida que se van acercando uno al otro y cuando llegan a juntarse, chocan, y se disparan a una mayor velocidad hacia lados contrarios. También pudimos observar que esto se da solamente con dos cuerpos, al incorporar un tercero, sufre la atracción, pero no el choque, ya que este lo sufrirán solo los

cuerpos que se encuentren más cerca entre sí, y el tercero seguirá su camino como venía desde un principio.

Escenario Newton – lab3

En la versión 3 del juego Newton - lab observamos todas las clases y métodos descriptos en las versiones anteriores, y además se incluyen las siguientes clases poniendo en funcionamiento algunos métodos nuevos.

Clases:

- **Obstacle**: Esta es una nueva clase que está a simple vista ubicada en la jerarquía de clases en el costado derecho de la pantalla. Es una subclase de la clase actor, por lo que hereda todos los atributos y métodos de la misma. Al inicio del juego esta clase esta instanciada 14 veces, por lo que en el escenario se muestran 14 obstáculos. El comportamiento que tiene es básicamente que el obstáculo se queda suspendido en una posición fija del escenario, y cuando es alcanzado por una bola, cambia su apariencia y reproduce un sonido específico. Los métodos que utiliza para adquirir el comportamiento que tiene son los siguientes:

    - Obstacle: es un constructor que inicializa una variable sound la cual va a guardar los valores de los sonidos que se van a reproducir cuando una bola toque un obstáculo.

    - Act: Dentro del método act se incluyen todos los métodos o líneas de código que se van a ejecutar una vez que se oprima el botón “run” del juego. En este caso, lo que hay dentro de este método es otro método (getOneIntersectingObject) el cual devuelve una referencia con el objeto que se ha colisionado. Con esta referencia luego se evalúan unos condicionales que comprueban si una bola está tocando una tecla. De ser así se cambia la imagen del obstáculo con el método setImage, y se reproduce un sonido guardado en las carpetas de sonido de Greenfoot, según el obstáculo que se haya tocado.

    - playSound: método que cuando se invoca, reproduce un sonido determinado de la biblioteca de sonidos de Greenfoot.

B:

EXPLICACIÓN DE LO QUE HICIMOS:

Lo que incluimos en el escenario de Newton – Lab3 es lo siguiente:

La principal característica que define el ***cambio significativo*** pedido en la consigna fue incluir un comportamiento en los planetas que permitiese que cada vez que toquen un objeto del tipo obstáculo en el escenario, se dividiese en dos partes iguales. En el momento en que estos dos cuerpos colisionan, además de eliminar el objeto (planeta) original y crear dos objetos del mismo tipo (Body) de

un tamaño menor, también alteramos los valores de la velocidad y la dirección de dichos cuerpos en base a la ubicación y tamaño que tengan en el momento en que colisionan, como así también el color. En cuanto a esto, el color que adquieren los nuevos cuerpos después de haberse puesto en contacto es aleatorio.

Modificamos además la cantidad de planetas que aparecen al iniciar el juego, de modo que la eliminación de obstáculos no sea tan abrupta y llegue a apreciarse. Sin embargo, notamos que a pesar de este cambio la eliminación seguía siendo rápida, por lo que optamos por añadir más obstáculos al inicio del juego (4 filas, 14 columnas).

Imágenes adjuntas:

A continuación, se van a adjuntar una serie de imágenes en las cuales se va a visualizar una sección de código. En dichas secciones es donde se implementaron los cambios que hacen posible el nuevo comportamiento del mundo. Se detallará con una breve explicación que cambio se incluyó en cada porción de código:

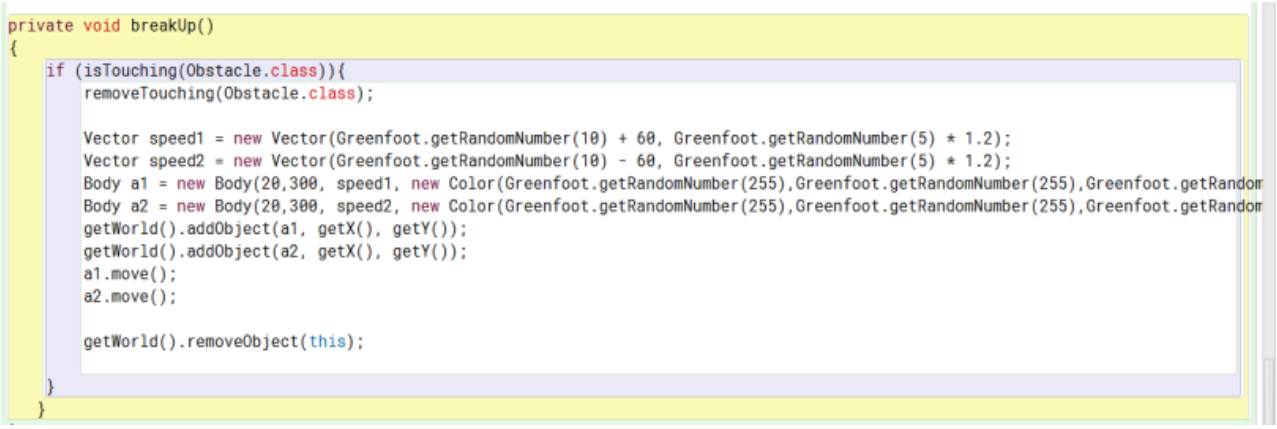
```
public void createObstacles()
{
    int i = 0;
    int j = 0;
    while (i < soundFiles.length)
    {
        addObject (new Obstacle (soundFiles[i] + ".wav"), 80 + j*60, altura);
        i++;
        j++;
        if (j > 13){
            altura = altura - 60;
            j = 0;
        }
    }
```

Descripción del cambio: Modificación en la clase Space, dentro del método createObstacles (), con lo cual logramos añadir una cantidad de 4 filas y 14 columnas de obstáculos. Los contadores “i” y “j” nos ayudaron a crear los objetos en las posiciones deseadas.

```
public void randomBodies(int number)
{
    while (number > 0)
    {
        int size = 20 + Greenfoot.getRandomNumber(30);
        double mass = size * 7.0;
        int direction = Greenfoot.getRandomNumber(360);
        double speed = Greenfoot.getRandomNumber(150) / 100.0;
        int x = Greenfoot.getRandomNumber(getWidth());
        int y = Greenfoot.getRandomNumber(getHeight()+300)+300;
        int r =  Greenfoot.getRandomNumber(255);
        int g =  Greenfoot.getRandomNumber(255);
        int b =  Greenfoot.getRandomNumber(255);
        addObject (new Body (size, mass, new Vector(direction, speed), new Color(r, g, b)), x, y);
        number--;
    }
}
```

Descripción del cambio:

Hicimos otra modificación en la clase Space, pero esta vez dentro del método randomBodies (), con la cuál logramos que los planetas aparezcan en la parte inferior de la mitad del escenario. Esto lo hicimos para que los planetas no aparezcan inicialmente entre medio de los obstáculos y se pueda apreciar su movimiento de atracción gravitatoria.

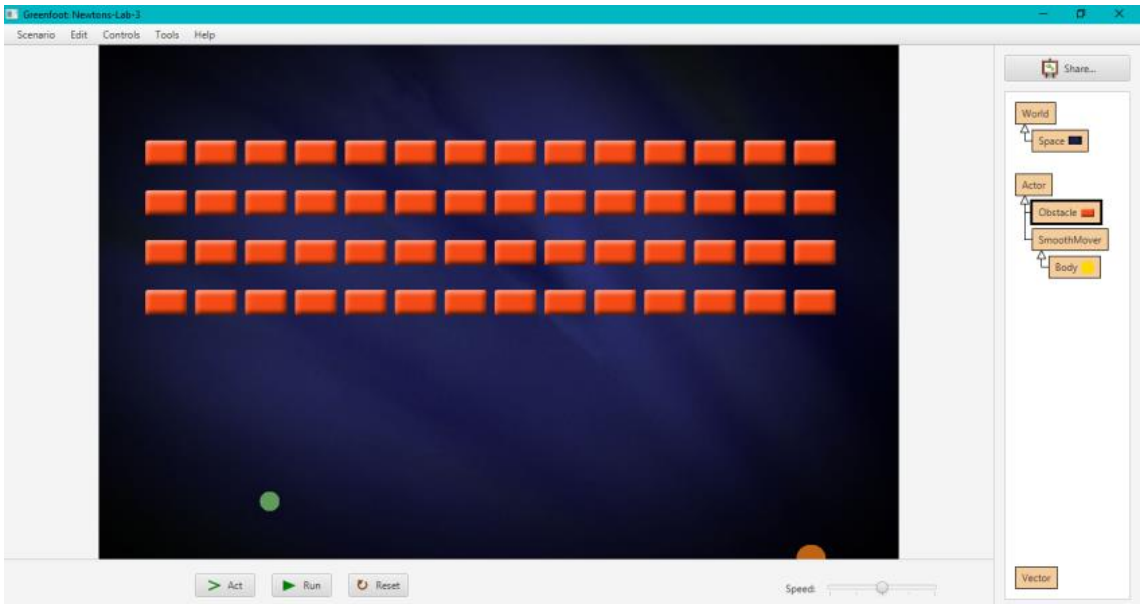


```
private void breakUp()
{
    if (isTouching(Obstacle.class)){
        removeTouching(Obstacle.class);

        Vector speed1 = new Vector(Greenfoot.getRandomNumber(10) + 60, Greenfoot.getRandomNumber(5) * 1.2);
        Vector speed2 = new Vector(Greenfoot.getRandomNumber(10) - 60, Greenfoot.getRandomNumber(5) * 1.2);
        Body a1 = new Body(20,300, speed1, new Color(Greenfoot.getRandomNumber(255),Greenfoot.getRandomNumber(255),Greenfoot.getRandom
        Body a2 = new Body(20,300, speed2, new Color(Greenfoot.getRandomNumber(255),Greenfoot.getRandomNumber(255),Greenfoot.getRandom
        getWorld().addObject(a1, getX(), getY());
        getWorld().addObject(a2, getX(), getY());
        a1.move();
        a2.move();

        getWorld().removeObject(this);

    }
   }
```

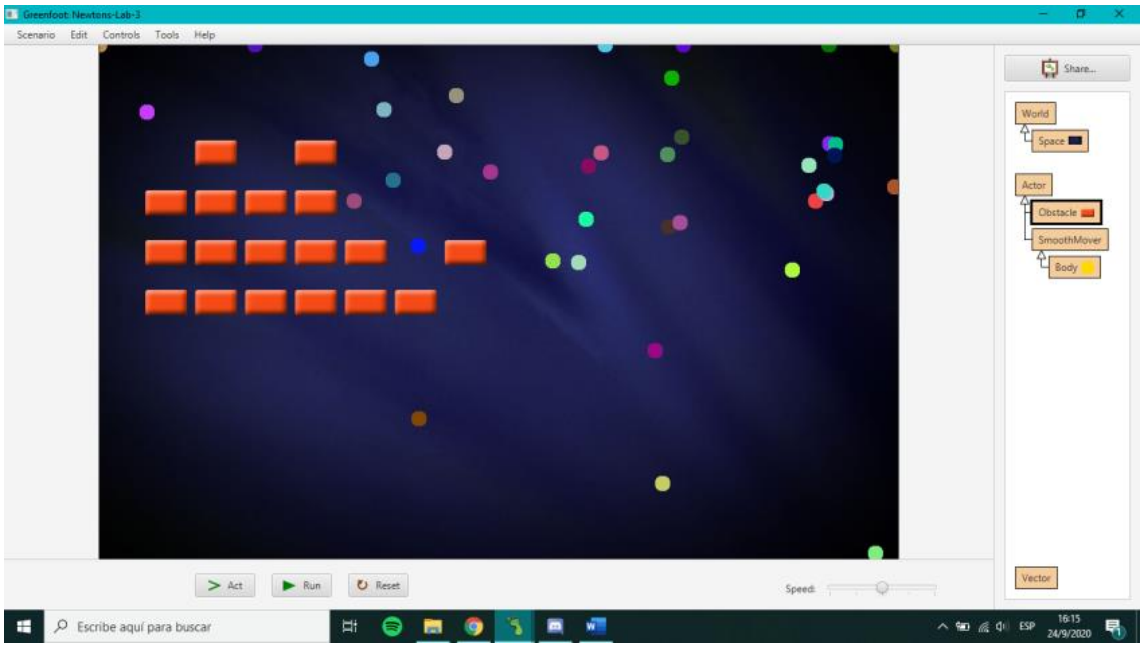
Descripción del cambio:

Implementamos una modificación en la clase Body, específicamente dentro del método breakUp (). Este método fue creado por nosotros, y es el que permite que cuando un planeta choca contra un obstáculo, se divida en dos, y cambie todas las características de dicho cuerpo (velocidad, dirección, color, movimiento).

Aclaración: cómo se puede observar, las líneas de código donde se crean los dos nuevos cuerpos como producto de la colisión salen cortadas. Sim embargo, se repite el mismo parámetro que se venía colocando hasta el momento (Greenfoot.getRandomNumber(255))



Pantalla inicial del juego



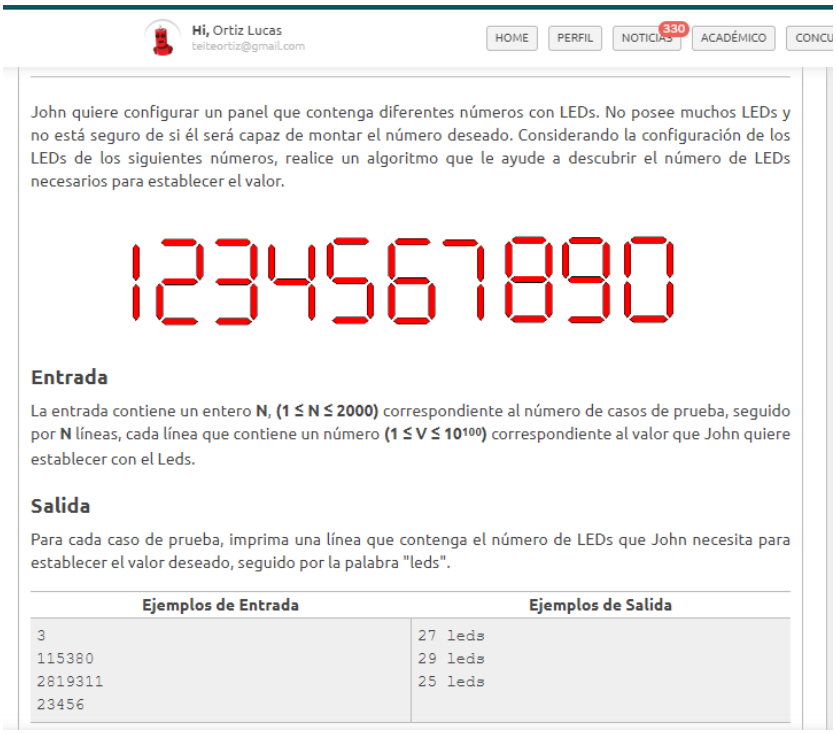
Pantalla del juego corriendo

Link del video realizado:

https://drive.google.com/file/d/1p7GNJWCI5F0TQ9gPCyYh7WfJOoo2pdv6/view?usp=sharing

3- (20 pts) Ingresar a https://www.urionlinejudge.com.br. Buscar el torneo Actividad Paradigmas de Programación 20/9 (4106). En dicho torneo solo deberá inscribirse solo el integrante que fue designado como portavoz. (grupal – cooperativo).

Primer enunciado panel led:



John quiere configurar un panel que contenga diferentes números con LEDs. No posee muchos LEDs y no está seguro de si él será capaz de montar el número deseado. Considerando la configuración de los LEDs de los siguientes números, realice un algoritmo que le ayude a descubrir el número de LEDs necesarios para establecer el valor.

1234567890

Entrada

La entrada contiene un entero **N**, **(1 ≤ N ≤ 2000)** correspondiente al número de casos de prueba, seguido por **N** líneas, cada línea que contiene un número **(1 ≤ V ≤ 10^{100})** correspondiente al valor que John quiere establecer con el Leds.

Salida

Para cada caso de prueba, imprima una línea que contenga el número de LEDs que John necesita para establecer el valor deseado, seguido por la palabra "leds".

Ejemplos de Entrada	Ejemplos de Salida
3 115380 2819311 23456	27 leds 29 leds 25 leds

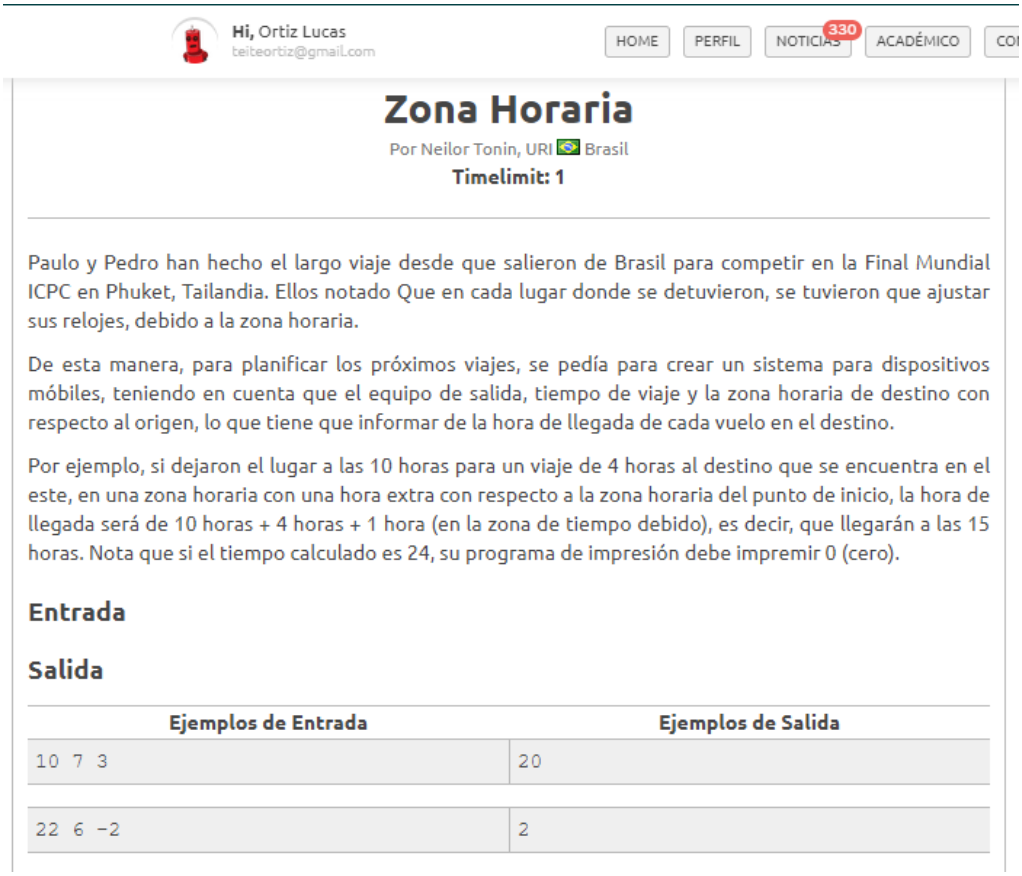
Resolución:

```
package Practicos;

import java.util.Scanner;


public class main{


    public static void main(String[] args) {

        // TODO Auto-generated method stub

        Scanner entrada = new Scanner(System.in);

        int cantidadNumeros;

        int cantLeds;
```

```
        int[] valores = new int[] {6,2,5,5,4,5,6,3,7,6};
        cantidadNumeros=entrada.nextInt();
        entrada.nextLine();
        String[] numeros = new String[cantidadNumeros];
        for (int i=0;i<numeros.length;i++)
        {
            numeros[i]=entrada.nextLine();
        }
        for (int i=0;i<cantidadNumeros;i++)
        {
            cantLeds=0;
        char[] numeroSolo = numeros[i].toCharArray();
        for (int j=0;j<numeros[i].length();j++) {
            String cadena = Character.toString(numeroSolo[j]);
            int entero = Integer.valueOf(cadena);
            cantLeds=cantLeds+valores[entero];
        }
        System.out.println(cantLeds+" leds");
        }
    }
}
```

Link del archivo:
https://drive.google.com/file/d/1SFV8UIOgOtfOC34TERFvaHzvXZ76gQf3/view?usp=sharing

Segundo enunciado zona horaria:



Resolución:

```
package Practicos;

import java.io.IOException;

import java.util.Scanner;


public class Hora {


    public static void main(String[] args) throws IOException {

        Scanner entrada = new Scanner(System.in);

        int TV = entrada.nextInt();

        int ZH = entrada.nextInt();

        int HL = entrada.nextInt();
```

```
        int hora = TV + ZH + HL;

        if (hora >= 24) hora -= 24;

        else if (hora < 0) hora += 24;

        System.out.println(hora);

    }

}
```

Link del archivo: https://drive.google.com/file/d/1zVhTr7PKudi-YPgHHFXijtd_rYA_gU4g/view?usp=sharing