**PRÁCTICA**

**DE SEMÁFOROS:**

**RED DE METRO**

**Índice**

1. Introducción
   1. Antecedentes
   2. Entorno de Desarrollo
2. Enunciado Práctico
   1. Explicación del Problema a resolver
3. Solución
   1. Código fuente de la Aplicación e Interfaces
   2. Descripción y justificación de la solución
4. Explicación
   1. Clases
   2. Estructura de datos
5. Listado de pruebas
   1. Entrada
   2. Ejecución
   3. Salida
6. Conclusiones

**1. Introducción**

**a. Antecedentes**

Hay que realizar una práctica para clase para poder obtener los conocimientos necesarios para sacar la asignatura adelante. La práctica consiste en aprender a programar mediante la utilización de semáforos.

**b. Entorno de desarrollo**

Windows 10 Pro de 64 bits

Con el programa Eclipse Java Oxygen

**2. Introducción**

1. **Explicación del programa a resolver**

Hay que implementar una aplicación que simule una red de metro.

La línea consta de 18 estaciones y con un mínimo de 3 trenes y máximo de 10 trenes

En cada estación existe un semáforo que inicialmente se encuentra en verde y cuando el tren llega a la estación, el semáforo se muestra en rojo. Cuando el tren sale a una nueva estación, pone en verde el semáforo de la estación anterior.

Ejemplo: el tren llega a la estación 3 y pone dicha estación en rojo para que no pueda pasar nadie más. Cuando dicho tren va a la estación 4 pone el semáforo en rojo y pone en verde el semáforo de la estación 3.

Se puede elegir el número de vueltas que dan los trenes entre 1 y 10.

Cuando un tren llega a la última estación y no se encuentra en la última vuelta, vuelve a empezar desde la estación 1.

Existe un tiempo de recorrido entre cada estación y además existe un tiempo aleatorio de estacionamiento en cada estación.

**3. Solución**

Se mostrará el Código Fuente y a continuación una explicación de dicho código

|  |
| --- |
| **Inicialización** |
| package metro;  import java.io.File;  import java.util.ArrayList;  import java.util.Random;  import java.util.concurrent.Semaphore;  class Inicializacion {  public static int numEstaciones;  public static int numTrenes;  public static int vueltas;  public static final int paradasL3 = 18;  // public static final int minNumEstaciones = 12;  public static Semaphore[] smfEstaciones;  public static Semaphore smfTren = new Semaphore(0);  public static Semaphore mutex= new Semaphore (1);  public static final String RutaNombreParadas = "G:"+File.separator+"PROYECTOS"+File.separator+"PSP"+File.separator+"Semaforos-Metro"+File.separator+"src"+File.separator+"L3.txt"; //Ruta del archivo donde se encuentran los nombres de las estaciones  public static ArrayList<String> nombreParadas = new ArrayList<String>(); // Creo un array con los nombres de las paradas  public static String[] nombreEstaciones;  public static int[][] estacionOcupada; // En la primera columna se indicará el nº de estación que es, en la segunda si está ocupada(poner nº tren) o no (poner 99)  public static final int columnas = 2;  public static Random aleatorio = new Random();  } |

En esta primera clase inicializamos todas las variables que vamos a tener.

Declaramos el número de estaciones, el número de trenes y el número de vueltas que vamos a tener que los inicializaremos más adelante en la clase **Main**.

Declaramos e inicializamos una constante con el número de paradas de la Línea 3 de metro, en esta línea hay 18 paradas.

Se inicializó un número mínimo de estaciones (ahora comentado) que explicaremos más adelante el porqué de esta inicialización.

Declaramos un array de semáforos, uno para cada estación, que inicializaremos en el **Main** después de haber inicializado los valores anteriores.

Declaramos e inicializamos un semáforo para los trenes que inicializamos a 0, este semáforo controla el movimiento de los trenes.

Declaramos e inicializamos un semáforo que se llama mutex que inicializamos a 1, este semáforo controla que solo entre un tren a una estación y no pregunten dos trenes a la vez por la misma estación.

Declaramos una constante String con la ruta del archivo que contiene los nombres de las estaciones. Se utiliza File.separator para que la ruta sea válida en todos los sistemas operativos.

Declaramos un arrayList de String para posteriormente guardar el nombre de las estaciones y poderlas utilizar.

Declaramos un array para el nombre de las estaciones, ya que con este tipo de array le ponemos un tamaño exacto y evitamos posibles errores al leer el array.

Declaramos una matriz. En la primera columna se indicará el número de estación en la que nos encontramos y en la segunda columna nos mostrará si está ocupada o no. Si está ocupada nos mostrará el tren que está en dicha estación, si está vacía nos mostrará un 99.

Declaramos e inicializamos una constante para más adelante indicar a la matriz que tiene solo dos columnas.

Declaramos un Random para poderlo utilizar en los sleeps para que el tiempo de espera sea aleatorio, los valores del aleatorio, se decidirán cuando se utilice.

|  |
| --- |
| **Main** |
| package metro;  import java.io.IOException;  import java.util.concurrent.Semaphore;  import javax.swing.JOptionPane;  public class Main {  private static Trenes a\_trenes[];  public static void main(String[] args) {  Metodos m = new Metodos(); // Clase métodos  /\* \*  \* INICIALIZACIÓN DE LAS ESTACIONES \*  \* \*/  /\*do {  try {  Inicializacion.numEstaciones = Integer.parseInt(JOptionPane.showInputDialog("Introduce nº de estaciones (mín 12 " + Inicializacion.minNumEstaciones + " y máx " + Inicializacion.paradasL3 + "): "));  } catch (NumberFormatException e) {  System.err.println("Debes introducir un número entero.");  }  } while(Inicializacion.numEstaciones < Inicializacion.minNumEstaciones || Inicializacion.numEstaciones > Inicializacion.paradasL3);  \*/  Inicializacion.numEstaciones = Inicializacion.paradasL3;  System.out.print("El número de estaciones introducidas es: " + Inicializacion.numEstaciones + "\n\n\n");  Inicializacion.nombreEstaciones = new String[Inicializacion.numEstaciones];  try {  m.inicializarArrayList(Inicializacion.RutaNombreParadas, Inicializacion.nombreParadas);  m.mostrarArrayList(Inicializacion.nombreParadas);  System.out.print("\n\n\n");  m.inicializarArrayNombres();  m.mostrarArrayNombres();  } catch (IOException e) {  System.out.println("El archivo no está disponible.");  }  /\* \*  \* INICIALIZACIÓN DE LOS TRENES \*  \* \*/  do {  try {  Inicializacion.numTrenes = Integer.parseInt(JOptionPane.showInputDialog("Introduce nº de trenes (mín 3 y máx 10): "));  } catch (NumberFormatException e) {  System.err.println("Debes introducir un número entero.");  }  } while(Inicializacion.numTrenes < 3 || Inicializacion.numTrenes > 10);  System.out.println("El número de trenes es de: " + Inicializacion.numTrenes);  /\* \*  \* INICIALIZACIÓN DE LAS VUELTAS QUE DARÁN LOS TRENES \*  \* \*/  do {  try {  Inicializacion.vueltas = Integer.parseInt(JOptionPane.showInputDialog("Introduce nº de vueltas(mín 1 y máx 10): "));  } catch (NumberFormatException e) {  System.err.println("Debes introducir un número entero.");  }  } while(Inicializacion.vueltas < 1 || Inicializacion.vueltas > 10);  System.out.println("El número de vueltas es de: " + Inicializacion.vueltas + "\n\nEmpezando a pasar trenes.\n\n\n");  try {  Inicializacion.estacionOcupada = new int[Inicializacion.numEstaciones][Inicializacion.columnas];  m.inicializarMatrizEstacion();  } catch (IOException e) {  System.out.println("Error en la inicialización de la matriz.");  }  Inicializacion.smfEstaciones = new Semaphore[Inicializacion.numEstaciones];  for(int i = 0; i < (Inicializacion.numEstaciones-1); i++) Inicializacion.smfEstaciones[i] = new Semaphore(1);  // Inicializo a 1 todos menos el ultimo que es a cero  // porque se me va a poner a 1 cuando el primer tren llegue a la primera estacion  Inicializacion.smfEstaciones[Inicializacion.numEstaciones-1] = new Semaphore(0);  System.out.println("Inicializados los semáforos.");  a\_trenes = new Trenes[Inicializacion.numTrenes];  /\* \*  \* INICIANDO RED DE METRO \*  \* \*/  Ventana v = new Ventana("Red de metro"); // Clase ventana  for (int j = 0; j < Inicializacion.numTrenes; j++) {  a\_trenes[j] = new Trenes(j,v,Inicializacion.vueltas);  a\_trenes[j].start();  try {  Thread.sleep(Inicializacion.aleatorio.nextInt(3000)+2500);  } catch (InterruptedException e) {  e.printStackTrace();  }  }  }  } |

Lo primero que hacemos en la clase Main es inicializar el número de estaciones que va a tener nuestra línea. Ahora está puesto para que tenga un número fijo de estaciones (18) pero en un futuro el programa puede pedir el número de estaciones que se quieren y que la ventana se adecúe al número de estaciones que quiera el usuario. De ahí que se haya querido dejar este código comentado, para una futura mejora del programa.

Teniendo el número de estaciones, inicializamos los dos arrays para las estaciones y mostramos por consola los arrays para ver que no ha habido ningún fallo.

Después inicializamos el número de trenes con un mínimo de 3 trenes y un máximo de 10 y mostramos por consola que se haya guardado bien. Hacemos lo mismo con el número de vueltas con un mínimo de 1 y un máximo de 10.

Ahora pasamos a la declaración del array. Con este código Inicializacion.estacionOcupada = new int[Inicializacion.numEstaciones][Inicializacion.columnas]; le indicamos que va a haber dos columnas y tantas filas como estaciones hayamos inicializado. Llamamos al método para inicializar dicha matriz que en la siguiente clase explicaremos como se inicializa.

Una vez hecho todo lo anterior, inicializamos el array del semáforo para las estaciones, le ponemos el mismo tamaño que estaciones hay y con un bucle lo inicializamos a uno para indicar que se puede pasar por cada estación, excepto en la última estación que se inicializará a cero porque cuando el tren pase por la primera estación, va a inicializar la última estación y de esta forma evitamos una posible colisión de los trenes en la misma estación por no haberse cerrado ese semáforo.

Después de esta parte, creamos un array de la clase **Trenes**, aquí lo declararemos para luego crear los hilos con los que trabajaremos.

Llamamos a la ventana pasándole por parámetros el nombre que va a llevar nuestra ventana.

Por último, con un bucle creamos los hilos de los trenes, cada hilo va a llevar pasado por parámetros el número de tren que es, la ventana que utilizamos y la inicialización de vueltas.

Hacemos que el hilo empiece y hacemos que el programa se duerma un número aleatorio de tiempo para retrasar la ejecución de cada tren.

El aleatorio lo creamos llamando al Random que teníamos creado en la clase **Inicializacion**. En este caso le decimos que espere un tiempo entre 3000 y 2500 milisegundos.

|  |
| --- |
| **Métodos** |
| package metro;  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.util.ArrayList;  import java.util.Iterator;  public class Metodos {  public void inicializarArrayList(String ruta,ArrayList<String> arrayList) throws IOException {  BufferedReader buffer = new BufferedReader(new FileReader(ruta)); // Creamos un buffer del contenido de los archivos  String linea = buffer.readLine(); // Leo la primera línea y si no es igual a null entro en el bucle  while(linea != null) { // mientras estoy en el bucle relleno el arrayList hasta que linea sea igual a null  arrayList.add(linea);  linea = buffer.readLine();  }  buffer.close(); // Cierro el buffer del fichero que hemos accedido para evitar consumir más recursos de los deseados  }  public void mostrarArrayList(ArrayList<String> arrayList) {  Iterator<String> it = arrayList.iterator();  while(it.hasNext()) {  System.out.println(it.next());  }  }  public void inicializarArrayNombres() throws IOException {  Iterator<String> it = Inicializacion.nombreParadas.iterator();  for(int i = 0; i < Inicializacion.numEstaciones; i++) {  if(it.hasNext()) {  String nombre = it.next();  Inicializacion.nombreEstaciones[i] = nombre;  }  }  }  public void mostrarArrayNombres() {  for(int i = 0; i < Inicializacion.numEstaciones; i++) {  System.out.println("Estación: " + Inicializacion.nombreEstaciones[i]);  }  }  public void inicializarMatrizEstacion() throws IOException {  for(int i = 0; i < Inicializacion.numEstaciones; i++) {  Inicializacion.estacionOcupada[i][1] = 99; // Iniciamos a 99 porque no hay ninguna estación ocupada  }  for(int i = 0; i < Inicializacion.numEstaciones; i++) {  Inicializacion.estacionOcupada[i][0] = i; // Iniciamos el nº de estación que es  }  }  } |

El *primer método* que tenemos es el de inicializar el array list al que le hemos pasado por parámetros la ruta y el nombre del array. A este método le añadimos la excepción *IOException*.

Dentro de este método, creamos y abrimos un buffer del contenido del archivo que le estamos pasando.

Creamos una variable y en ella leemos la primera línea, si está vacío el archivo contendrá **null** y no nos entrará en el bucle while que tenemos a continuación.

Si no está vacío significa que esta variable no contiene null y entra en el bucle. Dentro del bucle añadimos al arrayList cada línea que vayamos leyendo. Después de guardar la línea, leemos la siguiente y el bucle vuelve a chequear si contiene null o no. Este proceso se repite hasta salir del bucle. Una vez fuera del bucle, cerramos el buffer que habíamos creado anteriormente.

En el *segundo método* vamos a mostrar en la consola el arrayList que acabamos de inicializar. Para ello creamos un iterator y vamos leyendo con un bucle while y cuando acaba el arrayList, sale del bucle y de este método.

En el *tercer método* vamos a inicializar el array de nombres, le añadimos también la excepción *IOException*.

Volvemos a crear un iterator para recorrer el arrayList y con un bucle for, vamos leyendo con el iterator y a la vez vamos añadiendo valores al array. Cuando el bucle acaba es porque no se le pueden poner más estaciones al array.

En el *cuarto método* vamos a mostrar el array que acabamos de crear, con un bucle for recorremos el array y lo vamos mostrando en la consola para saber si hay algún fallo o no.

En el *quinto y último método* vamos a inicializar la matriz, a este método también le tenemos que añadir la excepción.

Creamos dos bucles for.

En el primero hemos inicializado a 99 la segunda columna de la matriz, el 99 indica que está vacío y en la clase **Trenes** vamos a ir cambiando los valores de esta columna según este vacía o no la estación. Si no está vacía, se escribirá el número de tren que la ocupa, si está vacía el 99 como hemos dicho anteriormente. La razón de utilizar el 99 y no el número cero es porque los arrays se inicializan en cero por lo que el primer hilo tren será el tren será el hilo cero y cometeríamos un gran fallo.

En el segundo bucle hemos inicializado el número de estación en la primera columna, empezando en cero y acabando en 17.

Cuando hayan acabado estos dos bucles, tendremos cada estación inicializada en 99 y preparada para recibir trenes.

|  |
| --- |
| **Trenes** |
| package metro;  import java.awt.Dimension;  import java.awt.Rectangle;  import javax.swing.ImageIcon;  class Trenes extends Thread {  int trenes;  private Ventana objVentana;  int vueltas;  final int distancia = 60;  ImageIcon icon;  Trenes(int tren, Ventana ventana, int vueltas){  this.trenes = tren;  this.objVentana = ventana;  this.vueltas = vueltas;  this.setName("Tren " + tren);  } // fin Trenes  void move(int estacion){  objVentana.labelImg[trenes].setVisible(true);  Rectangle r = objVentana.labelImg[trenes].getBounds();  Dimension dimensionR = r.getSize();  int x = objVentana.labelImg[trenes].getX();  while (x < objVentana.imagenesVerde[estacion].getX()-20) {  objVentana.labelImg[trenes].setBounds(x, objVentana.y, dimensionR.width, dimensionR.height);  try {  Thread.sleep(10);  } catch (InterruptedException e) {  e.printStackTrace();  }  x++;  }  }  void moveFinal() {  objVentana.labelImg[trenes].setVisible(true);  Rectangle r = objVentana.labelImg[trenes].getBounds();  Dimension dimensionR = r.getSize();  int x = objVentana.labelImg[trenes].getX();  while (x < objVentana.imagenesVerde[Inicializacion.numEstaciones-1].getX()+200) {  objVentana.labelImg[trenes].setBounds(x, objVentana.y, dimensionR.width, dimensionR.height);  try {  Thread.sleep(10);  } catch (InterruptedException e) {  e.printStackTrace();  }  x++;  }  }  public void run() {  try {  System.out.println("soy el " + this.getName() +" entrando");  do { // while vueltas  // evitamos que dos procesos comprueben si la estación está vacía a la vez, ya que podría darse una falsa lectura  Inicializacion.mutex.acquire(); // Disminuye 1 mutex  System.out.println("Soy el " + this.getName() + " paso mutex");  int empezar = 0;  objVentana.labelImg[trenes].setVisible(false);  Rectangle r = objVentana.labelImg[trenes].getBounds();  Dimension dimensionR = r.getSize();  objVentana.labelImg[trenes].setBounds(empezar, objVentana.y, dimensionR.width, dimensionR.height);  for(int estacion = 0; estacion < Inicializacion.numEstaciones; estacion++) {  boolean continuar = false;  while(!continuar) {  if(Inicializacion.estacionOcupada[estacion][1] == 99) {  Thread.sleep(Inicializacion.aleatorio.nextInt(1800)+500);  objVentana.imagenesRojo[estacion].setVisible(true); objVentana.imagenesVerde[estacion].setVisible(false);  Inicializacion.smfEstaciones[estacion].acquire();  Inicializacion.smfTren.release(); // +1. Activa mov tren  System.out.println(this.getName() + " moviendose");  if(estacion == 0) {  move(estacion); Inicializacion.estacionOcupada[estacion][1] = trenes; // Pongo el valor a la estación con el nº de tren Inicializacion.estacionOcupada[Inicializacion.numEstaciones-1][1] = 99; // La estación anterior la vacío con 99 Inicializacion.smfEstaciones[Inicializacion.numEstaciones-1].release(); // Activo el semáforo de la estación anterior objVentana.imagenesVerde[Inicializacion.numEstaciones-1].setVisible(true); objVentana.imagenesRojo[Inicializacion.numEstaciones-1].setVisible(false);  } else {  move(estacion); Inicializacion.estacionOcupada[estacion][1] = trenes; // Pongo el valor a la estación con el nº de tren  Inicializacion.estacionOcupada[estacion-1][1] = 99; // La estación anterior la vacío con 99 Inicializacion.smfEstaciones[estacion-1].release(); // Activo el semáforo de la estación anterior  objVentana.imagenesVerde[estacion-1].setVisible(true); objVentana.imagenesRojo[estacion-1].setVisible(false);  } // fin del if ver estaciones  System.out.println("Soy " + this.getName() + " en la vuelta " + vueltas + " y estoy en la estación " + Inicializacion.nombreEstaciones[estacion]);  Inicializacion.smfTren.acquire(); // -1. Desactiva mov tren  System.out.println(this.getName() + " parado");    Inicializacion.mutex.release(); // Aumenta 1 mutex  continuar = true;  } else {  objVentana.imagenesRojo[estacion].setVisible(true);  objVentana.imagenesVerde[estacion].setVisible(false);  System.out.println("Esperando a que " + this.getName() + " tenga libre la estación: " + Inicializacion.nombreEstaciones[estacion]);  continuar = false;  Thread.sleep(Inicializacion.aleatorio.nextInt(1500)+1000);  } // fin del if  } // fin while  } // fin for estaciones  Thread.sleep(Inicializacion.aleatorio.nextInt(1800)+500);  moveFinal();  System.out.println("Soy " + this.getName() + " en la vuelta " + vueltas);  vueltas--;  } while(vueltas > 0);  Thread.sleep(1000);  objVentana.labelImg[trenes].setVisible(false);  Inicializacion.estacionOcupada[Inicializacion.numEstaciones-1][1] = 99;  objVentana.imagenesVerde[Inicializacion.numEstaciones-1].setVisible(true);  objVentana.imagenesRojo[Inicializacion.numEstaciones-1].setVisible(false);  System.out.println("\n\nSoy " + this.getName() + " en la vuelta " + vueltas + ". He acabado el recorrido");  JOptionPane.*showMessageDialog*(**null**, "El " + **this**.getName() + " ha terminado de pasar","El programa ha terminado", JOptionPane.***WARNING\_MESSAGE***);  } catch(NullPointerException e) {  e.printStackTrace();  } catch (InterruptedException e) {  e.printStackTrace();  }  }  } |

En esta nueva clase vamos a manejar los hilos y el movimiento de cada tren. Lo primero que hacemos a esta clase es añadirle la extensión Thread.

Declaramos las variables y los objetos que le hemos pasado por parámetros y luego creamos un constructor en el que recogemos estos datos que han sido pasado por parámetros y los guardamos en unas variables que usaremos durante la clase. Además creamos un nombre para el tren que estamos pasando para poderlo llamar posteriormente.

Ahora nos encontramos con el método *move*, este método se va a encargar de que el tren se mueva por las estaciones.

Actualmente se encuentran los trenes ocultos, con este método lo primero que hacemos el que el tren que vayamos a utilizar sea visible.

Utilizamos el método *Rectangle*, con esta función vamos a coger los valores del setBounds del tren que vamos a utilizar y los vamos a guardar. Con el método *Dimension* cogemos el ancho y el alto que hemos guardado del setBounds con el método anterior y lo guardamos.

Declaramos la variable x y la inicializamos sacando el valor del eje X donde se encuentra la imagen a utilizar.

Con un bucle while podemos hacer que el tren se mueva estación a estación, para ello la condición del while es que la variable x que acabamos de averiguar sea menor al valor del eje X de cada estación menos unos pixeles para que quede centrado en tren. Mientras que esta condición se cumpla realizaremos este código: objVentana.labelImg[trenes].setBounds(x, objVentana.y, dimensionR.width, dimensionR.height);

La x del setBounds indica el eje X, objVentana.y indica el eje Y, con dimensionR.width sacamos el ancho de la imagen y con dimensionR.height sacamos el alto.

Después de este código hacemos que el programa se duerma 10 milisegundos para retardar un poco este movimiento y por último hacemos que la x sume 1 por cada vez que se recorre el bucle, al sumar 1 a la x y esta x se encuentra en el setBounds conseguimos que se mueva de 1 en 1 hasta llegar a la estación que queremos ir.

También tenemos un método *moveFinal* que solo se ejecuta cuando el tren está en la última estación. En *move* el tren se movía de estación a estación y se quedaba parado en la última. Con esto logramos que el tren continúe hasta el final. Se ha decidido hacer en otro método diferente para una mejor legibilidad del código. La base de este método es la misma que en el anterior solo que la condición es que se hace cuando el tren llega a la última estación y avanza hacia delante hasta salir de la ventana.

Una vez aclarados estos conceptos, entramos en el hilo que habíamos llamado en la clase **Main**. Lo primero de todo lo que hacemos es crear un try-catch para controlar las excepciones que nos pueden salir.

Creamos un bucle while con las vueltas que van a dar los trenes, recordamos que se había inicializado en el Main y se pasó por parámetros y ese parámetro se guardo en una variable en el constructor. Utilizamos esta variable en el bucle con la condición de que las vueltas sean mayores que cero.

Al principio de este bucle, llamamos al semáforo *mutex* y restamos uno con acquire. De esta forma evitamos que dos procesos comprueben a la vez si la estación está vacía o no, ya que podría darnos una falsa lectura. Después sacamos por consola que el tren ha pasado a mutex para comprobar que está todo correcto. Inicializamos una variable a cero, ponemos el tren invisible e igual que hemos hecho en el método *move* obtenemos el *Rectangle*  y *Dimension* e inicializamos el setBounds en el ejeX como la variable que hemos inicializado a cero. Con esto conseguimos que cuando lleve una o más vueltas y el tren esté al final del recorrido que vuelva a empezar desde el principio recorriendo todas las estaciones de nuevo.

Al final de este bucle creamos un sleep con un tiempo aleatorio entre 1800 y 500 milisegundos que va a ser el tiempo que esté el tren en la última estación. Después llamamos al método *moveFinal*, escribimos en la consola la vuelta en la que nos encontramos y después restar una vuelta al número de vueltas que teníamos. Esta parte del código explicada quedaría de esta forma:

|  |
| --- |
| do{  Inicialización.mutex.acquire(); // Disminuye 1 mutex  System.out.println("Soy el " + this.getName() + " paso a mutex");  int empezar = 0;  objVentana.labelImg[trenes].setVisible(false);  Rectangle r = objVentana.labelImg[trenes].getBounds();  Dimension dimensionR = r.getSize();  objVentana.labelImg[trenes].setBounds(empezar, objVentana.y, dimensionR.width, dimensionR.height);  /\* CONTENIDO QUE SE EXPLICARÁ A CONTINUACIÓN \*/  Thread.sleep(Inicializacion.aleatorio.nextInt(1800)+500);  moveFinal();  System.out.println("Soy " + this.getName() + " en la vuelta " + vueltas);  vueltas--;  } while ( vueltas > 0); |

Después de esta línea de código objVentana.labelImg[trenes].setBounds(empezar, objVentana.y, dimensionR.width, dimensionR.height); creamos un bucle for, con él pasaremos por todas las estaciones que tenemos. Inicializamos un booleano para continuar el proceso siguiente y lo inicializamos en false. Creamos un bucle while con la condición de que continuar sea false. Una vez creado el bucle while empezamos con una condición.

Si la matriz de la estación a la que queremos acceder está vacía (igual a 99) crearemos un sleep de aleatorio que se encuentre entre 1800 y 500 milisegundos. Llamamos a la ventana y ponemos visible el semáforo en rojo de dicha estación e invisible el semáforo en verde.

Llamamos al array de semáforos y accedemos al semáforo de esa estación y con un acquire() bloqueamos la estación para evitar que otro tren pueda entrar en la estación. Llamamos al semáforo de los trenes y con un release() activamos el movimiento del tren. Para ver si hemos llegado hasta aquí y funciona todo correctamente, escribimos en la consola que el tren se está moviendo.

Creamos una nueva condición para controlar correctamente las estaciones.

Si la estación es igual a cero (es la primera) hacemos que se mueva el tren llamando al método *move*, llamamos a la matriz de las estaciones y en la estación en la que nos encontramos cambiamos en la segunda columna su valor, le vamos a poner como nuevo valor el número del tren en el que nos encontramos. Después en esa misma matriz llamamos a la última estación (estación 17 en nuestro caso pero se puede hacer más sencillo poniendo el total de estaciones menos uno) y le ponemos el valor 99 para indicar que el tren ya no se encuentra en esa estación y se ha quedado libre para que otro tren pueda entrar. Llamamos al semáforo de la última estación de la misma manera que a la matriz (total -1) y le hacemos un reléase() para activar el semáforo de dicha estación y permite el paso del siguiente tren. Llamamos a la ventana y hacemos visible el semáforo en verde e invisible el semáforo en rojo de la estación anterior.

\* Anotación: aunque sea la primera vuelta, llamamos a la última estación porque este mismo código nos va a servir para cuando estén en otras vueltas, de esta forma evitamos repetir código y añadir condiciones que pueden retrasar el programa en las siguientes vueltas.

\*Anotación 2: Recordamos que en la clase **Main** inicializamos el último semáforo de las estaciones a cero, el motivo es que aquí en la primera vuelta lo vamos a inicializar por lo que si lo inicializamos antes a 1, en un futuro nos podría haber dado un error y podrían entrar dos trenes en esta última estación.

Si la estación es distinta de cero, realizamos lo mismo de antes exceptuando una pequeña variación. Llamamos a *move* y cambiamos en la matriz de la estación el valor 99 por el tren que se encuentra en dicha estación. En la estación anterior ya no llamamos a la última, ahora podemos coger el número de la estación que nos encontramos y restarle uno, es decir, si nos encontramos en la estación tres, vamos a liberar la estación dos.

Llamamos a la matriz de la estación anterior y le ponemos el valor 99 para indicar que se ha vaciado, el semáforo de la estación anterior con un release() volvemos a activar el semáforo y llamamos a la ventana y activamos o desactivamos el semáforo verde y rojo de la estación anterior.

Esta parte del código quedaría de esta forma:

|  |
| --- |
| /\* CÓDIGO ANTERIOR \*/  if(estacion == 0) {  move(estacion);  Inicializacion.estacionOcupada[estacion][1] = trenes; // Pongo el valor a la estación con el nº de tren  Inicializacion.estacionOcupada[Inicializacion.numEstaciones-1][1] = 99; // La estación anterior la vacío con 99    Inicializacion.smfEstaciones[Inicializacion.numEstaciones-1].release(); // Activo el semáforo de la estación anterior  objVentana.imagenesVerde[Inicializacion.numEstaciones-1].setVisible(true);  objVentana.imagenesRojo[Inicializacion.numEstaciones-1].setVisible(false);  } else {  move(estacion);  Inicializacion.estacionOcupada[estacion][1] = trenes; // Pongo el valor a la estación con el nº de tren  Inicializacion.estacionOcupada[estacion-1][1] = 99; // La estación anterior la vacío con 99  Inicializacion.smfEstaciones[estacion-1].release(); // Activo el semáforo de la estación anterior  objVentana.imagenesVerde[estacion-1].setVisible(true);  objVentana.imagenesRojo[estacion-1].setVisible(false);  } // fin del if ver estaciones  /\* CÓDIGO POSTERIOR \*/ |

Para comprobar si hay algún fallo, en la consola ponemos un mensaje diciendo el número de tren que es, la vuelta en la que se encuentra y la estación por la que va.

El semáforo del tren con un acquire() desactivamos su movimiento y en la consola sacamos un mensaje que nos indica que el tren se ha parado.

Llamamos al semáforo mutex y con un release() volvemos a activarlo.

Por último le cambiamos el valor del booleano continuar a True.

Si la condición de que esté libre la estación no se cumple, llamamos a la ventana y le ponemos el semáforo en rojo en visible y el verde en invisible.

Sacamos en la consola un mensaje diciendo que el tren está esperando a que se quede libre la estación.

El booleano continuar le ponemos que siga siendo false.

Hacemos que el programa se duerma con un aleatorio entre 1500 y 1000 milisegundos. De esta forma conseguimos que el tren no pregunte continuamente y nos llene la consola con el mismo mensaje, además de dar tiempo al tren que tiene delante a avanzar a la siguiente estación.

Aquí acabamos la condición para saber si la estación está vacía o no. Después cerramos el bucle while continuar y por último cerramos el bucle for de las estaciones.

Después de cerrar el bucle for de las estaciones nos encontramos con el código que hemos explicado anteriormente que se encontraba al final del bucle while en el que chequeábamos si se habían acabado las vueltas o no.

Para finalizar el hilo, cuando salimos del bucle while de vueltas, dormimos de nuevo el programa, esta vez con 1000 milisegundos, hacemos que desaparezca la imagen del tren, llamamos a la matriz en la última estación y le damos el valor 99, llamamos a la ventana y ponemos el semáforo en verde como visible y en rojo como invisible y mostramos un mensaje en la consola que nos diga que el tren ya ha finalizado el recorrido. En la ventana hacemos que nos aparezca una alerta indicando que el tren ya ha acabado su recorrido.

Esta última parte la tenemos que hacer porque el tren llegaba a la última estación en la última vuelta, el tren se marchaba pero la estación se quedaba sin vaciar porque se salía de todos los bucles anteriores. Se quedaban todos los trenes bloqueados sin llegar a acabar ningún tren más, de esta manera corregimos este error y ya pueden acabar todos los trenes.

|  |
| --- |
| **Ventana** |
| package metro;  import java.awt.Color;  import java.awt.Container;  import java.awt.Font;  import java.awt.Image;  import javax.swing.Icon;  import javax.swing.ImageIcon;  import javax.swing.JFrame;  import javax.swing.JLabel;  import javax.swing.JPanel;  import javax.swing.JTextField;  @SuppressWarnings("serial")  public class Ventana extends JPanel{  private JFrame ventana = new JFrame();  private JTextField via1;  private Container container;  JLabel[] imagenesEstaciones;  JLabel[] imagenesVerde;  JLabel[] imagenesRojo;  JLabel[] nombres;  JLabel[] labelImg;  ImageIcon[] imgTrenes;  Icon[] iconoTrenes;  private final String imgMetro = "metro5.png";  private final String imgVerde = "verde5.gif";  private final String imgRojo = "rojo5.gif";  // Valores del eje Y  private final int ySemArriba = 170;  private final int yEstArriba = 60;  // Valores del ancho y el alto de las imágenes  private final int AnchoEstArriba = 55;  private final int AltoEstArriba = 31;  private final int AnchoSemArriba = 25;  private final int AltoSemArriba = 25;  int x = 0;  int y = 145;  public Ventana(String nombre) {  ventana.setName(nombre);  ventana.setTitle(nombre);  ventana.setSize(1250, 600);  ventana.setResizable(false);  ventana.setLocationRelativeTo(null);  ventana.setLayout(null);  container=ventana.getContentPane();  container.setBackground(Color.decode("#f8f9f9"));      /\* \*  \* NOMBRES \*  \* \*/  nombres = new JLabel[Inicializacion.numEstaciones];  for(int i = 0; i < Inicializacion.numEstaciones; i++) {  nombres[i] = new JLabel();  nombres[i].setText(Inicializacion.nombreEstaciones[i]);  nombres[i].setFont(new Font("Calibre", Font.PLAIN,9));  // System.out.println(nombres[i].getText());  }  /\* \*  \* IMAGENES \*  \* \*/  imgTrenes = new ImageIcon[]{  new ImageIcon(this.getClass().getResource("tren1\_peque.png")),  new ImageIcon(this.getClass().getResource("tren2\_peque.png")),  new ImageIcon(this.getClass().getResource("tren3\_peque.png")),  new ImageIcon(this.getClass().getResource("tren4\_peque.png")),  new ImageIcon(this.getClass().getResource("tren5\_peque.png")),  new ImageIcon(this.getClass().getResource("tren6\_peque.png")),  new ImageIcon(this.getClass().getResource("tren7\_peque.png")),  new ImageIcon(this.getClass().getResource("tren8\_peque.png")),  new ImageIcon(this.getClass().getResource("tren9\_peque.png")),  new ImageIcon(this.getClass().getResource("tren10\_peque.png")) // 10  };  labelImg = new JLabel[Inicializacion.numTrenes];  iconoTrenes = new Icon[Inicializacion.numTrenes];  for(int i = 0; i < Inicializacion.numTrenes; i++) {  labelImg[i] = new JLabel();  labelImg[i].setVisible(false);  labelImg[i].setBounds(x, y, 50, 30);  iconoTrenes[i] = new ImageIcon(imgTrenes[i].getImage().getScaledInstance(labelImg[i].getWidth(),labelImg[i].getHeight(), Image.SCALE\_DEFAULT));  labelImg[i].setIcon(iconoTrenes[i]);  container.add(labelImg[i]);  }  imagenesEstaciones = new JLabel[]{  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))),  new JLabel(new ImageIcon(this.getClass().getResource(imgMetro))), // 18  };  imagenesVerde = new JLabel[]{  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))),  new JLabel(new ImageIcon(this.getClass().getResource(imgVerde))), // 18  };  imagenesRojo = new JLabel[]{  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))),  new JLabel(new ImageIcon(this.getClass().getResource(imgRojo))), // 18  };  for(int i = 0; i < imagenesVerde.length; i++) {  imagenesVerde[i].setVisible(true);  imagenesRojo[i].setVisible(false);  }  /\* \*  \* VIAS \*  \* \*/  via1 = new JTextField();  via1.setBounds(0, 150, 1250, 20);  for(int i = 0; i < 600; i++) via1.setText(via1.getText() + "|");  via1.setEnabled(false);  via1.setDisabledTextColor(Color.WHITE);  via1.setBackground(Color.GRAY);  container.add(via1);    /\* \*  \* ESTACIONES \*  \* \*/  // Estaciones de arriba, x+60  nombres[0].setBounds(50, 110, 150, 50);  container.add(nombres[0]);  imagenesEstaciones[0].setBounds(60, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[0]);  imagenesVerde[0].setBounds(75, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[0]);  imagenesRojo[0].setBounds(75, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[0]);  nombres[1].setBounds(110, 90, 150, 50);  container.add(nombres[1]);  imagenesEstaciones[1].setBounds(120, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[1]);  imagenesVerde[1].setBounds(135, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[1]);  imagenesRojo[1].setBounds(135, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[1]);  nombres[2].setBounds(160, 110, 150, 50);  container.add(nombres[2]);  imagenesEstaciones[2].setBounds(180, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[2]);  imagenesVerde[2].setBounds(190, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[2]);  imagenesRojo[2].setBounds(190, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[2]);  nombres[3].setBounds(220, 90, 150, 50);  container.add(nombres[3]);  imagenesEstaciones[3].setBounds(240, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[3]);  imagenesVerde[3].setBounds(250, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[3]);  imagenesRojo[3].setBounds(250, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[3]);  nombres[4].setBounds(280, 110, 150, 50);  container.add(nombres[4]);  imagenesEstaciones[4].setBounds(300, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[4]);  imagenesVerde[4].setBounds(310, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[4]);  imagenesRojo[4].setBounds(310, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[4]);  nombres[5].setBounds(340, 90, 150, 50);  container.add(nombres[5]);  imagenesEstaciones[5].setBounds(360, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[5]);  imagenesVerde[5].setBounds(370, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[5]);  imagenesRojo[5].setBounds(370, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[5]);  nombres[6].setBounds(420, 110, 150, 50);  container.add(nombres[6]);  imagenesEstaciones[6].setBounds(420, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[6]);  imagenesVerde[6].setBounds(430, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[6]);  imagenesRojo[6].setBounds(430, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[6]);  nombres[7].setBounds(490, 90, 150, 50);  container.add(nombres[7]);  imagenesEstaciones[7].setBounds(480, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[7]);  imagenesVerde[7].setBounds(490, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[7]);  imagenesRojo[7].setBounds(490, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[7]);  nombres[8].setBounds(550, 110, 150, 50);  container.add(nombres[8]);  imagenesEstaciones[8].setBounds(540, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[8]);  imagenesVerde[8].setBounds(550, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[8]);  imagenesRojo[8].setBounds(550, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[8]);  nombres[9].setBounds(580, 90, 150, 50);  container.add(nombres[9]);  imagenesEstaciones[9].setBounds(600, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[9]);  imagenesVerde[9].setBounds(610, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[9]);  imagenesRojo[9].setBounds(610, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[9]);  nombres[10].setBounds(660, 110, 150, 50);  container.add(nombres[10]);  imagenesEstaciones[10].setBounds(660, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[10]);  imagenesVerde[10].setBounds(670, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[10]);  imagenesRojo[10].setBounds(670, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[10]);  nombres[11].setBounds(730, 90, 150, 50);  container.add(nombres[11]);  imagenesEstaciones[11].setBounds(720, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[11]);  imagenesVerde[11].setBounds(730, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[11]);  imagenesRojo[11].setBounds(730, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[11]);  nombres[12].setBounds(800, 110, 150, 50);  container.add(nombres[12]);  imagenesEstaciones[12].setBounds(780, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[12]);  imagenesVerde[12].setBounds(790, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[12]);  imagenesRojo[12].setBounds(790, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[12]);  nombres[13].setBounds(855, 90, 150, 50);  container.add(nombres[13]);  imagenesEstaciones[13].setBounds(840, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[13]);  imagenesVerde[13].setBounds(850, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[13]);  imagenesRojo[13].setBounds(850, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[13]);  nombres[14].setBounds(895, 110, 150, 50);  container.add(nombres[14]);  imagenesEstaciones[14].setBounds(900, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[14]);  imagenesVerde[14].setBounds(910, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[14]);  imagenesRojo[14].setBounds(910, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[14]);  nombres[15].setBounds(950, 90, 150, 50);  container.add(nombres[15]);  imagenesEstaciones[15].setBounds(960, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[15]);  imagenesVerde[15].setBounds(970, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[15]);  imagenesRojo[15].setBounds(970, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[15]);  nombres[16].setBounds(1030, 110, 150, 50);  container.add(nombres[16]);  imagenesEstaciones[16].setBounds(1020, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[16]);  imagenesVerde[16].setBounds(1030, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[16]);  imagenesRojo[16].setBounds(1030, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[16]);  nombres[17].setBounds(1090, 90, 150, 50);  container.add(nombres[17]);  imagenesEstaciones[17].setBounds(1080, yEstArriba, AnchoEstArriba, AltoEstArriba);  container.add(imagenesEstaciones[17]);  imagenesVerde[17].setBounds(1090, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesVerde[17]);  imagenesRojo[17].setBounds(1090, ySemArriba, AnchoSemArriba, AltoSemArriba);  container.add(imagenesRojo[17]);  ventana.setVisible(true);  ventana.toFront();  }  public static void main(String[] args) {  new Ventana("Red de metro");  }  } |

En esta clase ventana la haremos pública para poder acceder desde el resto de las clases y la pondremos una extensión con JPanel.

Dentro de ella lo primero que haremos será declarar todas las variables y constantes que vamos a utilizar y definir tanto si son públicas o privadas.

Una vez hecho esto haremos un constructor público de la ventana que cogeremos un String que le pasamos desde la clase **Main**. Con este String le daremos el título y el nombre a la ventana.

A la ventana le pondremos un layout con formato null para poder colocar los componentes en la posición que queramos con setBounds, creamos una capa contenedora (recordamos que las declaraciones las hacemos encima del constructor en la parte habilitada para ello) y le decimos que ocupa todo el espacio de la ventana y le damos un color de fondo a la capa contenedora.

Creamos un array de JLabel para los nombres de las estaciones con el tamaño del número de estaciones y lo inicializamos en un bucle for. Con setText y dentro de este método llamamos al array que contiene los nombres en la clase **Inicialización** y así conseguimos asignar cada JLabel con el nombre de cada estación. Le damos una fuente, un tipo de texto y un tamaño al texto del JLabel. Se encuentra comentado que nos muestre por consola los JLabel, esto se hizo para comprobar que se habían guardado correctamente los nombres.

Ahora implementamos las imágenes que vamos a utilizar, son la imagen de la estación de metro, los dos semáforos y los trenes que vamos a utilizar.

Poner las imágenes de los trenes que vamos a utilizar en un ImageIcon, le ponemos el mismo tamaño que número de trenes hay a un Jlabel para estas imágenes y a un Icon para después añadir las imágenes.

Con un bucle for que recorra el número de trenes declaramos el Jlabel al que hemos puesto tamaño. Hacemos invisibles los Jlabel y los colocamos con un setBounds. A continuación implementamos las ImageIcon como Icon consiguiendo la imagen y la escala para después averiguar el ancho y el alto. Introducimos el Icon en el Jlabel y añadimos el Jlabel a la capa contenedora.

Declaramos en un Jlabel e implementamos las imágenes de las estaciones. Hacemos lo mismo con los semáforos verdes y rojos.

Con un bucle for que recorra todos los semáforos y ponemos como visible el semáforo en verde y en invisible el rojo.

Creamos la vía por la que va a pasar el tren, la creamos con un JTextField y con setBounds la ponemos donde queremos con un ancho y un alto. Con un bucle for le escribimos barras verticales para una mejor apariencia. Hacemos que no se pueda escribir sobre el textField, el color de texto se lo ponemos en blanco y el color de fondo en gris y añadimos la vía a la capa contenedora.

Ahora vamos colocando los nombres de las estaciones, las imágenes de la estación y las imágenes de los semáforos con un setBouns y los añadimos a la capa contenedora, así con hasta haber colocado todas las estaciones y todos los semáforos donde queremos.

Una vez creados todas las estaciones, hacemos visible la ventana y hacemos que se muestre al frente.

Con un Main propio de la ventana podemos ir viendo como va quedando la ventana.

\*Anotación: cuando añadimos el primer setBounds para poner el nombre de las estaciones, da error porque no sabe donde sacar las inicializaciones que las hace el programa principal.

**4. Explicación**

**a. Clases**

Tenemos una clase principal llamada Main de la cual se hacen las inicializaciones y las llamadas a las otras clases.

Una clase Inicialización que es donde inicializamos todo lo que podemos utilizar en varias clases, hacemos esta clase para tener todas las variables locales juntas.

Una clase método que utilizamos para inicializar los arrays y la matriz.

Una clase trenes que es donde vamos a manejar los hilos y el movimiento de los trenes.

**b. Estructura**

Como hemos explicado anteriormente se ha estructurado en varios bloques.

El primer bloque hacemos las inicializaciones.

El segundo bloque llamamos a los métodos.

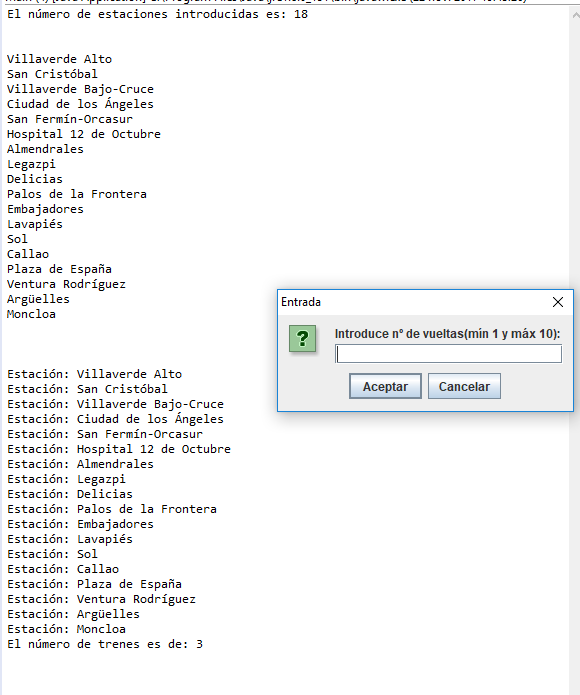
El tercer bloque inicializamos los arrays de los hilos y los lanzamos.

El cuarto bloque manejamos los hilos y dentro de este bloque tendríamos un quinto bloque que es utilizar la ventana y hacer que los trenes se muevan.

**5. Listado de pruebas**

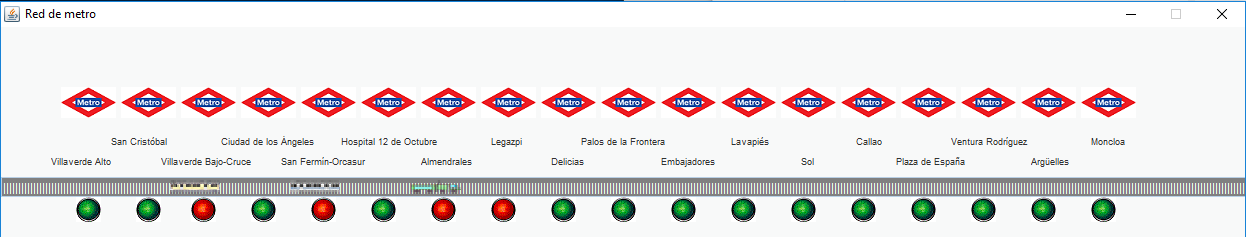
**a. Entrada**

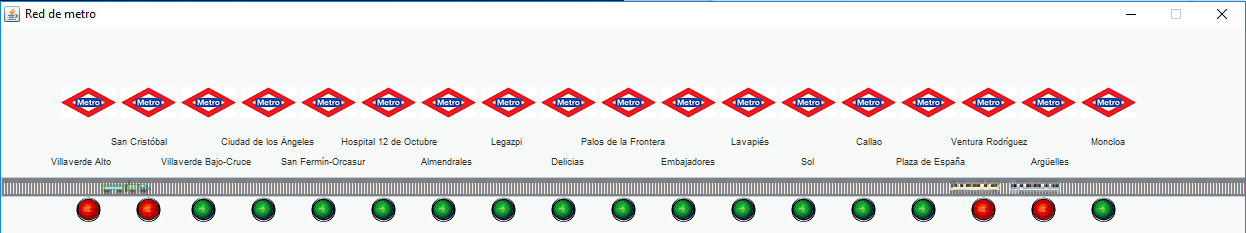
Introducimos los datos y sacamos por la consola los datos para ver si se han introducido correctamente.



**b. Ejecución**

Después de ver que está todo correctamente funcionando, vemos que sale por consola como se van desplazando los trenes y como van entrando los trenes en las estaciones, además en el entorno gráfico se puede ver también la ejecución del programa.

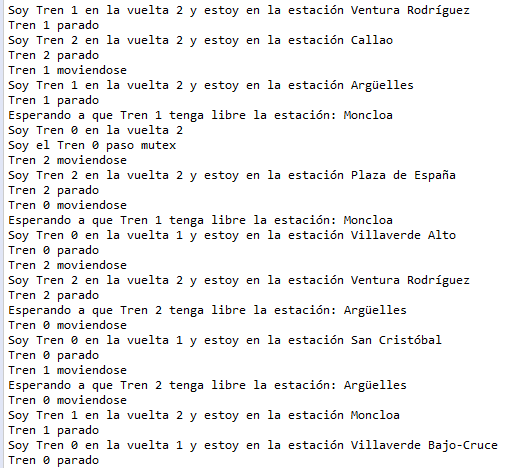
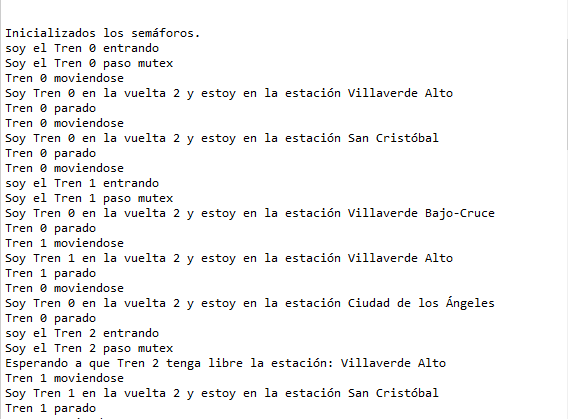


****

****

Aquí tenemos varios ejemplos de como funciona gráficamente.

A continuación mostraremos como funciona el programa con los mensajes en la consola.



Por consola podemos ver que es algo más lioso porque a veces los mensajes no salen en el orden correcto, a veces se retrasan y parece que el orden no es correcto o que hay algo mal.

**c. Salida**

Al acabar los trenes su recorrido mostramos por consola que los trenes han acabado el recorrido.





En el entorno gráfico simplemente dejaremos de ver como pasan los trenes.

**6. Conclusiones**

La práctica ha resultado más difícil de lo que parecía. El mayor problema ha sido el entorno gráfico porque todavía no tenemos los conocimientos necesarios y se ha procedido a investigar como realizar ciertas partes de dicho entorno.