TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE OAXACA

ASIGNATURA: FUNDAMENTOS DE TELECOMUNICACIONES

CATEDRÁTICO: MCC. VALVERDE JARQUÍN REYNA

ALUMNO: GARCÍA GARCÍA JOSÉ ÁNGEL

GRUPO: ISB

HORA: 12:00 – 13:00

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES

REPORTE DEL TRABAJO DE UNIDAD 4 Y 5

OAXACA DE JUÁREZ, OAX, 23/DICIEMBRE/2020

ÍNDICE GENERAL

ÍNDICE DE IMÁGENES

PLANTEAMIENTO DEL PROBLEMA

Para realizar la práctica de esta unidad, el catedrático nos brindó la información necesaria para trabajar mediante la plataforma Moodle, que consistía en una breve descripción del problema:

> *Realizar un programa en java que simule la Multiplexación por división de tiempo síncrona*
>
> *Que el programa le pida al usuario cuantas estaciones van a trasmitir y cuantos caracteres trasmitirá cada estación.*
>
> *Que muestre la secuencia de mensajes enviados(TRAMAS).*
>
> *Que muestre Cuántos bits se transmiten.*
>
> *Que muestre Cuántos bits útiles se transmiten(son los que representan los datos que envía el origen al destino).*
>
> *Repetir para la Multiplexación Asíncrona.*

Entonces, se puede ver claramente cuáles son los puntos que debe tener el programa y también completó la información sobre el programa a realizar mediante explicaciones en las clases realizadas por Meet.

DISEÑO DE LA SOLUCIÓN

Para darle solución a lo planteado anteriormente fue relativamente sencillo, ya que en clases anteriores se nos había explicado a detalle este tipo de multiplexación e inclusive la maestra, explicó muy detallado como se creaba las tramas, que justamente es lo más importante del programa, el poder mostrar las tramas. Por ello, que con explicaciones previas era sencillo el poder crear esta parte de TDM Síncrono, por otra parte, se solicitó el TDM Asíncrono, donde fue que estuvo más compleja la forma de obtener los parámetros necesarios para construir la trama ya que se requieren de cálculos extras.

La idea, del programa se centra en que, a partir de la cantidad de estaciones y los caracteres de cada una, te muestre las tramas que se generan y también información respecto a los bits transmitidos y bits útiles transmitidos de acuerdo con la TDM correspondiente.

He de aclarar que para la solución implementé los principios básicos de POO para hacer una optimización de código, y también el manejo de ArrayList mediante Stream, una peculiaridad de Java que nos permite hacer cosas de forma más simple. Y respecto a la información para crear las tramas y mostrar la respectiva información, en su mayoría la tomé del libro proporcionado por el catedrático, y en el caso de TDM Asíncrono, lo complementé con información de internet y otras fuentes bibliográficas.

CLASE ESTACIÓN

Lo primero que realicé fue una abstracción de lo que es una estación y es por ello que mediante una clase **Estacion** doy la representación de lo que es una estación, por lo que tendrá información que posee esta, la cual es un identificador, la cantidad y cuales caracteres envía y también los bits. Y, por otra parte, estará toda la parte restante del programa, que pide los datos al usuario para poder crear las estaciones, rellenar las estaciones, generar la trama y mostrarla, así también estará la parte donde se muestre la información de los bits transmitidos y de los bits útiles transmitidos. Es decir, en esta parte se encontrará la parte lógica del programa y en la otra el modelo de la estación.

Empezando por crear la clase **Estacion** con sus respectivos atributos que son útiles para identificar cada estación y tener la información necesaria de esta.

Esta declaración se visualiza en la siguiente imagen:

```
/**
 * Clase utilizada como una abstracción de una estación
 * */
class Estacion{

    private ArrayList<String> data; // Información que transmite
    private int bits, // Cantidad de bits que transmite
                numCaracter, // Número de caracteres de la estacion
                id; // Identificador
}
```

Ilustración 1 – Declaración inicial de clase Estación

Como toda clase, requiere de un constructor, en este caso la nuestra no es una excepción, por lo que se implementa su constructor. Que como sabemos, para crear una **Estacion** se necesita un identificador y el número máximo de caracteres que este podrá enviar.

Aquí mismo se puede calcular el número de bits que envía la estación, simplemente haciendo una operación de multiplicación. Que consiste en el número de caracteres por 8 y eso nos arrojaría los bits totales que envía la estación.

Esto implementando en nuestra clase Estación se visualiza así :

```
    /**
     * Constructor de una estación
     * */
    public Estacion(int numCaracter, int id){
        this.id = id;
        this.data = new ArrayList<>();
        this.numCaracter = numCaracter;
        bits = numCaracter * 8; // Se supone que cada caracter
es de 8 bits
    }
```

Ilustración 2 - Constructor de Estacion

Ahora, ya podemos crear nuestras estaciones con su respectivo identificador y la cantidad de caracteres que puede enviar. Lo que nos faltaría son sus métodos para generar esos caracteres enviar.

Entonces, la parte de agregar caracteres la hice de forma automática, para que el usuario no tuviera que hacerlo de forma manual. El método que nos permite hacer esto se llama **addCaracters**, lo que hace es agregar los caracteres de acuerdo a la estación y al final los muestra. Para agregar caracteres, se lo dejamos a criterio del usuario, que este pueda agregar el carácter que sea de su agrado o el que guste enviar.

```
    /**
     * Permite agregar los respectivos caracteres a la estación
     * */
    public void addCaracters(){
        Scanner sc = new Scanner(System.in); // Scanner para leer de teclado
        System.out.println("Caracteres de Estación " + id);
        for(int i = 0; i < numCaracter; i++){
            System.out.printf("Caracter [" + (i + 1) + "] = ");
            data.add(new Character(sc.nextLine().charAt(0))); // Agrega el caracter a los datos de la
estación
        }
        System.out.println("-------------------------------------");
    }
```

Ilustración 3 - Método addCaracters

Hasta ahora ya tenemos la información para la estación, pero esta información solo se puede visualizar desde aquí y no donde está nuestro programa principal, por lo que procedemos a hacer los métodos para retornar la información para cuando sea necesario. Para esto se crean los famosos getters: **getNumCaracter**, **getBits**, **getData**.

```
/**
 * Retorna el número de caracteres que envía la estación
 * */
public int getNumCaracter(){
    return numCaracter;
}

/**
 * Retorna el total de bits que envía la estación
 * */
public int getBits(){
    return bits;
}

/**
 * Retorna el array de caracteres que envía la estación
 * */
public ArrayList<Character> getData(){
    return data;
}
```

Ilustración 4 - Getters de Estacion

Y, por último, nos quedaría crear un método que sea capaz de imprimir los datos que envía la estación y como los envía, con esto último me refiero al orden en que viajan los caracteres. Para ello se creó el método **imprimirDatos** que al ser ejecutado muestra la información mencionada de acuerdo a la estación, y su implementación es:

```
    /**
     * Imprime los datos que envía y como los envía la
estación (orden)
     * */
    public void imprimirDatos(){
        System.out.printf("ESTACIÓN " + id + " --> ENVÍA --> [
");
        StringBuilder info = new StringBuilder();
        data.forEach(e -> info.append(e.toString()+ " "));
        System.out.printf(info + "] --> DE LA FORMA --> [" +
info.reverse() + " ]\n");
    }
```

Ilustración 5 - Método para imprimir datos

Y con esto, la clase **Estacion**, ya estaría lista para ser utilizada.

Hasta aquí ya tenemos la mitad del programa, porque justo esta clase nos va ser de mucha ayuda para realizar lo demás y también para explotar una característica particular de Java.

CLASE PRINCIPAL

Vamos a empezar por definir nuestra clase y lo necesaria en ella, esto es crear una clase que se llama **ProgramaUnidad** y en ella vamos a tener algunos atributos o variables que nos serán útiles, en este caso tenemos:

- estaciones: Este es un ArrayList<Estacion> que nos sirve para controlar todas las estaciones, es un array de estaciones.
- numEstaciones: Nos sirve para almacenar la cantidad de estaciones que se crearán
- sc: Es una instancia de la clase Scanner y es útil para leer datos desde teclado

```
import java.util.*;

public class ProgramaUnidad {

    private static ArrayList<Estacion> estaciones; // Array de estaciones
    private static int numEstaciones; // Número de estaciones
    static Scanner sc = new Scanner(System.in); // Scanner para leer de teclado
}
```

Ilustración 6 - Declaración de clase principal

Ahora, detallaré todos los métodos que tiene esta clase principal.

El primero de ellos es **crearEstaciones** que como su nombre lo dice, es para crear estaciones, lo que hace es crear un ArrayList con el número de estaciones indicadas por la variable *numEstaciones* y al crear a cada una, le solicita al usuario que introduzca la cantidad de caracteres que va a transmitir dicha estación.

La implementación de este método se puede visualizar en la siguiente imagen:

```
    /**
     * Crear las estaciones con su respectivo número de caracteres a transmitir
     * */
    public static void crearEstaciones(){
        estaciones = new ArrayList<>(numEstaciones); // Crea el array para almacenar las
estaciones
        System.out.println("\n*****************************************************************\n");
        for(int i = 1; i <= numEstaciones; i++) {
            System.out.print("Cantidad de caracteres a transmitir la estación [" + i + "] = ");
            estaciones.add(new Estacion(sc.nextInt(), i));  // Crea la estación y la Agrega a
estaciones
        }
        System.out.println("\n*****************************************************************\n");
    }
```

Ilustración 7 - Crear estaciones

Ya tenemos nuestro método para crear las estaciones, ahora explicaré un método que permite rellenar todas las estaciones con sus respectivos caracteres. Este método es muy sencillo de hacer, porque estoy haciendo uso de Stream, una peculiaridad de Java que nos permite trabajar un ArrayList de una forma más eficiente y sencilla.

En este caso, lo que hacemos es decirle, que recorra todas las estaciones mediante el **forEach** y luego de cada estación *estación* le indicamos con -> que le ejecute su método **addCaracter**, dicho método es de la clase Estación y que ya había explicado anteriormente. Y no se retorna ningún valor, simplemente se ejecuta.

De tal forma que el método nos queda:

```
    /**
     * Rellena todas las estaciones ejecutando su método para agregar caracteres de cada
 estación
     * */
    public static void rellenarEstaciones(){
        estaciones.stream().forEach(estacion -> estacion.addCaracters());
    }
```

Ilustración 8 - Rellenar estaciones

En este punto ya tenemos creado nuestras estaciones con sus respectivos datos.

Ahora, explicaré los métodos para el TDM Síncrona y luego los métodos para el TDM Asíncrona.

TDM SÍNCRONA

Para generar la trama de es tipo, es importante saber antes cual es la cantidad mayor de caracteres que transmite cualquiera de las estaciones. Es decir, tenemos que buscar de todas las estaciones, quien es la que transmite mayores caracteres, ya que justo ese valor, es el que nos indica el número de tramas a generar.

Entonces, el método que nos permite buscar ese valor más grande de caracteres emitidos por una de las estaciones es **getMaxCaracter** que también aplica Stream, entonces se usa **stream()**, y lo que se hace es mapear todos los valores de caracteres de cada estación, para eso se usa **mapToInt()**, recordando que el método **getNumCaracter()** nos retorna el máximo de caracteres de la estación, luego se busca el máximo de ellos mediante **max()** y se obtiene como valor entero mediante **getAsInt()** y es lo que se retorna.

Esto implementado, entonces sería como se muestra en la imagen:

```
    /**
     * Calcula y retorna el mayor número de caracteres que envía una
estación, para saber cuantas tramas se generan en TDM Asincrono
     * */
    public static int getMaxCaracter(){
        return
estaciones.stream().mapToInt(Estacion::getNumCaracter).max().getAsInt(
);
    }
```

Ilustración 9 - Caracteres máximo

Ahora, vamos a continuar con otros dos métodos que son refrentes a la información respecto a bits, antes de explicar cómo se genera y muestra la trama.

Para obtener los bits útiles, básicamente, es realizar la sumatoria de todos los bits que envía cada estación. Entonces, tenemos que recorrer todas las estaciones y obtener sus números de bits y luego sumarlos todos, pero haciendo uso de Stream, esto se vuelve muy fácil.

Lo que hacemos es crear nuestro método **getBitsUtiles** y utilizar ***stream()*** para luego mapear a enteros las cantidades de bits mediante ***mapToInt()*** pero primero, de cada estación, ejecutamos su método ***getBits()*** que nos devuelve los bits totales que transmite dicha estación, y son estos los que se mapean y, por último, ejecutamos ***sum()*** para sumar todos y es lo que retorna el método.

Implementado esto, sería de la siguiente forma:

```
    /**
     * Calcula y retorna los bits transmitidos que son utiles en TDM
Sincrona
     * */
    public static int getBitsUtilesSincrona(){
        return  estaciones.stream().mapToInt(Estacion::getBits).sum();
    }
```

Ilustración 10 - Método para obtener bits útiles transmitidos de TDM Síncrona

Ahora, para obtener los bits transmitidos, se tiene que realizar una operación sencilla. Para esto multiplicaremos el número de tramas que se generaron (esto es lo mismo que el número mayor de caracteres que envía una de las estaciones y obtenido mediante **getMaxCaracter()**) por las ranuras de tiempo de cada trama (o también llamado rodajas de trama) y luego multiplicado por 8, ya que se expresa el resultado en bits y, por último, se le suman los bits de sincronización que se haya utilizado. Recordando que los bits de sincronización son también llamados bits de tramado y se agregan en cada una de las tramas, por lo que su valor es igual al número de tramas generadas y esto a su vez, es igual al número ,mayor de caracteres que envía una de las estaciones, es decir, esto es **getMaxCaracter()** nuevamente.

De tal forma que la implementación del método es sencilla y queda de la siguiente forma:

```
/**
 * Calcula y devuelve los bits que se han transmitido
 * */
public static int getBitsTransmitidos(){
    return getMaxCaracter() * numEstaciones * 8 + getMaxCaracter();
}
```

Ilustración 11 - Método para obtener bits transmitidos

Hasta este punto, ya tenemos lo básico y queda pendiente explicar a detalle cómo es que se genera la trama. Para la parte de la trama, no implemente Stream porque se volvía más complicado ya que se tiene que jugar con los valores y recorridos del array.

Para generar a trama, se implementó el método ***generarTrama()*** que consiste a grandes rasgos a generar un String que es la trama y se pueda imprimir.

La idea es generar primero desde la última trama que se envía hasta la primera, esto para que se pueda visualizar bien al momento de imprimir, de lo contrario se vería todo invertido.

Entonces

- Se declaran dos variables de tipo String, *trama* y *tramas*,

  trama -> Para guardar la trama en cada iteración

  tramas -> Para almacenar cada trama por iteración y al final tener todas las tramas que se envían.

- Se declara una variable *bit_tramado* con valor inicial de 1 que servirá para indicar el bit de tramado que le corresponde a la trama, este valor irá intercalándose entre 1 y 0.
- Se inicia un ciclo for con la cantidad máxima de caracteres, para ello se ejecuta el método ***getMaxCaracter()*** que nos devuelve dicho valor y es esencial, porque justo este for nos indica la cantidad de tramas que se generan.

- Luego, dentro de cada iteración del ciclo, se inicializa una variable trama = “[“, para indicar que inicia una trama.
- Se inicia otro bucle for, que va desde el número de estaciones, dicho valor almacenado en la variable *numEstaciones* y este valor es llamado ranuras de tiempo y que indican la cantidad de rodajas de cada trama.
    - En este bucle, obtenemos de cada estación, sus caracteres que envía.
    - Luego se inicia un bloque *try-catch* porque si la estación no envía carácter o envía menos que otra, entonces tenemos un error en ejecución y para ello, lo resolvemos con esto
        - Aquí obtenemos el carácter en la posición que se refiere a la iteración del primer for, es decir, el número de carácter mayor y esto irá decrementando. El punto es obtener el carácter de la posición tal y colocarlo en la trama junto con “ | ” como un separador.
        - Si se tiene un error, esto nos lleva al *catch*, y además esto nos indica que no hay carácter por transmitir y en la trama se concatena un “_” para indicar que no hay carácter por transmitir y luego se concatena “|” como separador.
    - Se le quita a la *trama* el “|” que se agrega de más.
    - Se concatena nuestra *trama* a la variable *tramas*, para así seguir con la otra y además se cierra la trama recién calculada “] [” y se concatena su bit de tramado, el cual cambia de valor, y se invierte en cada trama. Finalmente se concatena “] – “ para cerrar toda la trama. De tal forma que todas las tramas se van concatenando en la variable *tramas* hasta terminar el primer ciclo for.
- Al final, solo retornamos nuestra variable tramas.

La implementación del método **generarTrama()** quedaría de la siguiente forma:

```
    /**
     * Genera las tramas de acuerdo con un TDM Sincrona
     * */
    public static String generarTramaSincrona(){
        String tramas = "", trama = ""; // Variables para almacenar todas las tramas y cada trama
        int bit_tramado = 1; // Para indicar el bit de tramado
        for(int t = getMaxCaracter() - 1; t >= 0; t--){
            trama = "["; // Indica el inicio de una trama
            for(int i = numEstaciones - 1; i >= 0; i--){
                ArrayList<Character> datos = estaciones.get(i).getData(); // Se obtiene los
caracteres de la estación
                try {
                    trama += " " + datos.get(t) + " |"; // Lo colocamos en la trama si es que
tiene
                }catch (Exception error){
                    trama += " " + "_" + " |"; // Colocamos un _ si no tiene caracter para
transmitir
                }
            }
            trama = trama.substring(0 ,trama.length() - 1); // Quitar el | al final de cada trama
            tramas += trama + "] [" + (bit_tramado = Math.abs(bit_tramado - 1)) + "] --- "; //
Concatenamos a las tramas y su bit de tramado
        }
        return tramas;
    }
```

Ilustración 12 – Método para generar tramas TDM Síncrona

TDM ASÍNCRONA

Para generar la trama de este tipo de multiplexación, necesitamos tomar en cuanto varios parámetros y varias consideraciones para ello, explicaré los métodos secundarios que se usan para generar la trama y por último, el método que genera la trama.

De entrada, como la idea es utilizar un ArrayList de Queue, ya que este tipo de TDM se comporta como un sistema de colas, entonces lo ideal fue programarlo pensando de esa forma.

Para eso, necesitamos un método que a partir de un ArrayList de Queue, determine si todas las que contienen están vacías o todavía contienen datos. Para esto se implementó el método que tiene de nombre **hayTodavia**, que básicamente recorre el ArrayList y verifica Queue por Queue si está vacía, devuelve false en el caso de que no haya datos.

La implementación de este método es:

```java
/**
 * Verifica si las Queue tienen datos o ya no de acuerdo aun ArrayList<Queue>
 */
public static boolean hayTodavia(ArrayList<Queue<Character>> datos){
    for(Queue<Character> d : datos)
        if(!d.isEmpty())
            return true;
    return false;
}
```

Ilustración 13 - Verificar datos en colas

Ahora, otro método importante, es que el TDM Asíncrono, necesita de un bit de direccionamiento, pero para obtener ese bit de direccionamiento, se requiere convertir un dígito a bits para luego estimar la cantidad de bits que utilizará para ese direccionamiento.

Entonces se implementa primero un método recursivo **toBinary** para convertir cualquier digito a binario.

```java
/**
 * Convierte un número a binario
 * */
public static String toBinary(int n){
    if(n == 1) return "1";
    return toBinary(n / 2) + n % 2;
}
```

Ilustración 14 - Conversión de digital a binario

Para luego, en un método llamado **getBitsDireccionamiento** poder obtener el valor de bits que se utilizarán como direccionamiento, que básicamente es la longitud de bits en binario del número de estaciones, esto es la variable *numEstaciones*.

```
    /**
     * Calcula y retorna el número de bits para direccionamiento de acuerdo a las estaciones para
el TDM Asincrona
     * */
    public static int getBitsDireccionamiento(){
        return toBinary(numEstaciones).length();
    }
```

Ilustración 15 - Método para obtener bits de direccionamiento

Otro método importante, es saber el número de caracteres que se están enviando por todas las estaciones. Para ello, se crea un método que recibe el nombre de **getNumCaracteresTotales**. Entonces se implementa, que básicamente lo que hace es mapear **mapToInt** a un arreglo de enteros, la cantidad de caracteres que envía cada estación que se obtiene mediante **getNumCaracter**, a lo último se suman estos valores con la función **sum** y es lo que se retorna.

```
/**
 * Retorna el número de caracteres que se envían por todas las estaciones
 * */
public static int getNumCaracteresTotales(){
    return estaciones.stream().mapToInt(Estacion::getNumCaracter).sum();
}
```

Ilustración 16 - Método para obtener el número de caracteres de todas las estaciones

También, se necesita saber que estaciones van a enviar datos, es decir, quienes van a participar en la TDM Asíncrona. Para eso se crea un método con el nombre de **getNumEstacionesEnvia**. La idea es sencilla, se recorre el ArrayList de estaciones y lo que se hace es un filtro con **filter**, en el que se selecciona a todas las estaciones que su número de caracteres sea mayor a 0 y de los seleccionados, se realiza un conteo mediante ***count*()** y este valor es el que retorna.

La implementación de este método sería:

```
/**
 * Retorna el número de estaciones que envían datos
 * */
public static int getNumEstacionesEnvian(){
    return (int)(estaciones.stream().filter(estacion -> estacion.getNumCaracter() > 0).count());
}
```

Ilustración 17 - Método para obtener el número de estaciones que envían datos

Ahora, viene algo importante, el estimar mediante un cálculo estadístico el valor de las ranuras de tiempo que tendrá cada trama. Para ello, se crea un método con el nombre de **getNumRanurasTiempoAsincrona** que retorna el valor del número de ranuras requerido y que más conviene.

Para esto, se aplica un formula de media de los dispositivos que transmiten a la vez. Al ser una media, es relativamente sencilla la operación, consiste en obtener los caracteres totales a enviar mediante **getNumCaracteresTotales** y luego obtener el número de estaciones que envían datos, este mediante **getNumEstacionesEnvian** y luego hacer la aproximación, pero hay que tener una consideración final, que si este valor estimado es mayor al número de estaciones *numEstaciones* entonces se tiene que buscar uno más bajo, eso simplemente se hace restándole uno.

```
/**
 * Calcula y retorna las ranuras de tiempo para un TDM Asincrona
 * */
public static int getNumRanurasTiempoAsincrona(){
    int caracteres = getNumCaracteresTotales(),
            numEstacionesEnvian = getNumEstacionesEnvian(),
            aprox = caracteres / numEstacionesEnvian;
    return  aprox < numEstacionesEnvian ? aprox : aprox - 1;
}
```

Ilustración 18 - Método para obtener el número de ranuras de tiempo de cada trama

Ahora, con este valor de ranuras, se puede hacer una estimación de las tramas que vayamos a tener, tampoco es tan exacto, pero en su totalidad es de una aproximación alta. Es simplemente una división del total de caracteres **getNumCaracteres** y el número de ranuras que tendrá cada trama, obtenido mediante **getNumRanurasTiempoAsincrona**. Este método no es utilizado como tal para generar la trama, pero si lo es para mostrar el número de tramas generados.

```
/**
 * Calcula y retorna el número de tramas recomendado para una TDM Asincrona
 * */
public static int getNumTramasAsincrona(){
    int caracteres = getNumCaracteresTotales(),
            ranuras = getNumRanurasTiempoAsincrona();
    return caracteres / ranuras;
}
```

Ilustración 19 - Método para calcular el número de tramas

Ahora, con esta información, se puede realizar el método principal para generar la trama.

El método para generar la trama tiene por nombre **generarTramaAsincrona**, y su implementación es sencilla.

- Primeramente, se declaran alguns variables:
  tramas: Contentdrá todas las tramas
  numRanuras: El número de ranuras que debe tener las tramas, son las ranuras de tiempo y son obtenidas mediante el método **getNumRanurasTiempoAsincrona()**
  numTramas: El número de tramas que se debe tener finalmente, esto es obtenido mediante el método **getNumTramasAsincrona()**
  caracteres: Contendrá todos las Queue con caracteres temporalmente que se deberán transmitir.
- Se recorren todas las estaciones, haciendo que guarden todos sus datos en Queue, es decir, las estaciones se pueden traducir como colas con sus respectivos datos.
- Luego se declara una variable para tener la trama temporalmente.
- Se declara una variable index, que nos permitirá estar recorriendo las estaciones.
- Después en un ciclo while, lo que se hace es vaciar todos los datos de las estaciones en orden, es como si tuviéramos todas las tramas juntas y sin separadores. Y se va guardando todo en la trama temporal que teníamos. Se hace en un try-catch porque no todas las colas tienen la misma cantidad de caracteres y puede haber problemas al obtener los datos, ya que no existe. El ciclo seguirá hasta que las colas estén vacias y ya no tengan carácter, esto se comprueba en cada loop mediante el método **hayTodavia** que ya había explicado anteriormente.
- Terminando el while, luego se agrega un bit de tramado, ya que es el primero a tener.
- Hasta ahora, tenemos toda la trama, pero no en orden, es decir, no está ni dividida ni organizada por tramas, solo es una general.
- La idea del for, es que podamos separar esta trama grande en pequeñas tramas. Entonces la idea es relativamente sencilla.
  - La variable aux que indica el número de rodajas hechas a la trama
  - Si aux es 0 es porque la trama empieza y entonces se agrega un corchete para indicar que inicia la trama y si no, no se agrega nada
  - Luego se agrega el carácter de la posición respecto a i, que recorre la trama grande y temporal que tenemos, así mismo le concatena el bit de direccionamiento que tiene.
  - Ahora se incrementa la variable aux, indicando que hemos colocado un carácter y esto indica que se ha hecho una ranura de tiempo.
  - Se verifica si vamos en la última ranura para indicar que se debe cerrar la trama:
    - Se procede a cerrar la trama con un corchete y además se concatena el bit de tramado que tiene.
    - Se reinicia el contado de rodajas otra vez a 0 para hacer otra trama
    - Se decrementa la variable del número de tramas, ya que ya se ha hecho una.

    - Luego si estamos en la última trama, entonces agrega el último corche para cerrarlo.
- Se termina de separar esa trama y ya estaría casi resuelto el problema, solo faltaría invertir, porque todo se agrega de forma inversa ya que se recorre de forma inversa.
- Al final, como se agrego de forma inversa todo, para que se vea bien, lo que se tiene que hacer es invertir la trama. Pero también habría que quitar algunos " | " que se agregan de más y por ello al final se aplica el método replace y finalmente esto se retorna, y todo lo guardamos en una variable llamada *nuevo* porque justo a esa se aplica algunos ajustes.
- Se calcula si hacen falta por completar la ultima trama, es decir, si hace falta agregar "_" para representar que no hay más elementos. Si en dado caso que haga falta, entonces se rellenan mediante un for.
- Finalmente, se retorna esta variable que contiene todas tramas.

```
    /**
     * Genera las tramas de acuerdo con un TDM Asincrona
     * */
    public static String generarTramaAsincrona(){
        StringBuilder tramas = new StringBuilder(); // Almacena las tramas
        int bit_tramado = 1, // Para indicar el bit de tramado
            numRanuras = getNumRanurasTiempoAsincrona(), // Para saber en cuantas ranuras partir la trama
            numTramas = getNumTramasAsincrona(); // El número de tramas a generar
        ArrayList<Queue<Character>> caracteres = new ArrayList<>(); // Contiene todos los caracteres que se envían
        for(int i = 0; i < numEstaciones; i++)
            caracteres.add(new LinkedList<>(estaciones.get(i).getData()));
        StringBuilder tmpTrama = new StringBuilder(); // La trama temporal
        int index = 0; // Para controla el index de cada caracter
        while(hayTodavia(caracteres)){
            index = index >= numEstaciones ? 0 : index;
            try {
                Queue<Character> datos = caracteres.get(index++);
                tmpTrama.append(datos.remove().toString() + (index)); // Se guarda todos los caracteres en
            }catch (Exception e){}
        }
        tramas.append("]1["); // Para la parte inicial, siempre inicia en bit tramado 1
        for(int i = 0, aux = 0; i < tmpTrama.length() - 1; i+=2){
            tramas.append( aux == 0 ? "] " : ""); // Inicia una trama dependiendo del número de rodajas
            tramas.append(" | " + tmpTrama.charAt(i+1) + "" + tmpTrama.charAt(i)); // Agrega el caracter a la trama
            aux += 1; // Indica que ya se colocó un caracter en la trama
            if(aux == numRanuras && numTramas != 0){ // Valida si es momento de cerrar la trama
                // Cierra la trama con su respectivo bit de tramado
                tramas.append( i < tmpTrama.length() - 2 ? " [ ---- ]" + (bit_tramado = Math.abs(bit_tramado - 1))
+ "[" : "");
                aux = 0; // Reinicia las ranuras hechas
                numTramas--; // Decrementa las tramas, indicando que se completó una
            }
            tramas.append(i == tmpTrama.length() - 2 ? " [" : ""); // Cierra la última trama recien calculada y la
agrega a tramas
        }
        StringBuilder nuevo = new StringBuilder(tramas.reverse().toString().replace("|  ]", "]"));
        int faltante = getNumTramasAsincrona() * numRanuras - getNumCaracteresTotales() ;
        for(int i = 1; i <= faltante; i++)
            nuevo.insert(i*2,"_ ");
        return nuevo.toString(); // Regresa las tramas
    }
```

Ilustración 20 - Método para generar tramas TDM Asíncrona

Finalmente, se realizan los métodos que calculan la información respecto a los bits, tanto de los trasmitidos y los útiles transmitidos.

Para obtener los bits transmitidos por parte del TDM Asíncrona, se implementó el método que tiene por nombre **getBitsTransmitidosAsincrona** que calcula dicho valor. Es mediante una formula sencilla, que corresponde obtener el total de caracteres que se envían de acuerdo al producto del número de ranuras y el número de tramas. Para luego multiplicar por la suma de 8 y los bits de direccionamiento, finalmente se le suma el número de tramas.

```
    /**
     * Calcula y retorna los bits que se han transmitido
     * */
    public static int getBitsTransmitidosAsincrona(){
        return (getNumRanurasTiempoAsincrona() * getNumTramasAsincrona()) * (8 +
getBitsDireccionamiento()) + getNumTramasAsincrona();
    }
```

Ilustración 21 - Método para obtener bits transmitidos en TDM Asíncrona

Y para obtener los bits útiles transmitidos, es tan sencillo cómo ejecutar el método de**getBitsUtilesSincrona** ya que son los mismos bits útiles los que se transmiten.

```
/**
 * Calcula y retorna los bits transmitidos que son utiles en TDM Asincrona
 * */
public static int getBitsUtilesAsincrona(){
    return  getBitsUtilesSincrona();
}
```

Ilustración 22 - Obtener el número de bits útiles de transmisión en TDM Asíncrona

IMPRESIÓN DE TRAMAS

Bueno, ya teniendo los métodos que calculan las tramas y sus métodos secundarios, lo que ahora queda pendiente es explicar cómo es que se implementan dichos métodos. Lo que hice fue, crear un método que se llama **mostrarDatos** con la finalidad de que el usuario viera que datos son los que se envían y en qué orden.

Entonces, la idea es sencilla, porque si recordamos, la clase ***Estacion***, tiene su método que imprime esto, por lo que se hace, es recorrer todo el ArrayList de estaciones (*estaciones*) e indicarle a cada estación que ejecute este método. Antes de realizar esto, se imprime en pantalla al usuario que se muestran los datos de cada estación, entonces la implementación de este método es:

```
/**
 * Muestra los datos que envía cada estación, ejectuando el respectivo método de la estación
 * */
public static void mostrarDatos(){
    System.out.println("\n DATOS DE CADA ESTACIÓN");
    estaciones.stream().forEach(e -> e.imprimirDatos());
}
```

Ilustración 23 - Método para mostrar datos al usuario

Bien, antes de acceder al método para mostrar las tramas, previamente debemos tener un método que sirva a modo de separador o decoración. Entonces cree un método llamado **decoración** que simplemente a partir de un String, imprime asteriscos con la misma longitud del String, esto servirá como decorador y su implementación es muy sencilla.

```
/**
 * Imprime una linea de asteriscos para mostrar la trama a modo de decoración
 * */
public static void decoracion(String trama){
    for(int i = 0; i < trama.length(); i++)
        System.out.printf("*");
}
```

Ilustración 24 - Método decorativo

El método se llama **mostrarTramas** y es sencillo de realizar.

- Mostrar al usuario la información respecto a que trama se imprimirá, en este caso primero sobre las tramas de TDM Síncrona.
- Luego se calcula la trama para ello ocupamos el método **generarTramaSincrona()**.
- Se ejecuta una decoración de asteriscos.
- Luego se imprime la trama recién calculada.
- Se ejecuta otra decoración.

- Ahora, se imprime la información respecto a las tramas generadas recientemente, esta información es respecto a los bits transmitidos y el número de bits útiles transmitidos. Mediante **getBitsUtilesSincrona()** y **getBitsTransmitidosSincrona().**
- Se imprime que ahora se mostrará la trama asíncrona.
- Se repiten los mismos pasos, solo que ahora para calcula las tramas se ocupa el método **generarTramasAsincronas()**.
- Se imprime la decoración, luego se imprimen las tramas calculadas y se vuelve a imprimir una decoración.
- Y finalmente, se muestran todos los datos referentes a bits respecto a las tramas asíncronas, que para ello se ejecutan sus respectivos métodos como **getBitsTransmitidosAsincrona(), getBitsUtilesAsincrona()** y **getBitsDireccionamiento()**.

```
    /**
     * Permite mostrar las tramas generadas tanto de TDM Sincrona como ASincrona
     * */
    public static void mostrarTramas(){
        System.out.println("\nTRAMA(S) DE TDM SINCRONA: \n");
        String trama = generarTramaSincrona(); // Obtenemos las tramas mediante TDM Asincrona
        decoracion(trama);
        System.out.println("\n\n" + trama + "\n"); // Mostramos la tramas sincrona
        decoracion(trama);
        System.out.println("\n\nINFORMACIÓN  DE TDM SINCRONA: ");
        System.out.println("\n" + getBitsTransmitidosSincrona() + " bps = " + getMaxCaracter()
                + " tramas/segundo x " + (8 * numEstaciones + 1) + " bits/trama"); // Información de la
tasa de datos
        System.out.println("Número de bits transmitidos: " + getBitsTransmitidosSincrona()); // Información
de bits transmitidos
        System.out.println("Número de bits útiles transmitidos: " + getBitsUtilesSincrona()); //
Información de bits utiles

        System.out.println("\n\nTRAMA(S) DE TDM ASINCRONA: \n");
        String tramaAsincrona = generarTramaAsincrona(); // Obtenemos las tramas mediante TDM Asincrona
        decoracion(tramaAsincrona);
        System.out.println("\n\n" + tramaAsincrona + "\n"); // Mostramos la tramas asincrona
        decoracion(tramaAsincrona);
        System.out.println("\n\nINFORMACIÓN DE TDM ASINCRONA: \n");
        System.out.println("Número de bits transmitidos: " + getBitsTransmitidosAsincrona()); //
Información de bits transmitidos
        System.out.println("Número de bits útiles transmitidos: " + getBitsUtilesAsincrona()); //
Información de bits utiles
        System.out.println("Bits para el direccionamiento: " + getBitsDireccionamiento()); // Información
del bit de direccionamiento
    }
```

Ilustración 25 - Método para obtener y mostrar las tramas

MAIN

Ya tenemos todo lo necesario, ahora solo falta el método principal, este método es el main. Aquí vamos a juntar todos los métodos que acabamos de crear y explicar.

- Primero se solicita el número estaciones al usuario y lo guardamos en la variable *numEstaciones.*
- Luego, mandamos a crear nuestras estaciones, esto con el método ***crearEstaciones()***
- Y posterior a ello, ejecutamos el otro método para rellenarlas, esto lo hacemos ejecutando el método ***rellenarEstaciones().***
- Ahora, ejecutamos el método de ***mostrarDatos()*** que muestra los datos de las estaciones.
- Finalmente, el método de ***mostrarTramas()***, que calcula ambas tramas y las muestra.

De tal forma, que lo dicho anteriormente se puede observar en la siguiente imagen, en la que se implemente el método.

```
/**
 * Función principal del programa para ejecutar todos las fucniones
 * */
public static void main(String[] args) {
    System.out.print("Introduce el numero de estaciones: ");
    numEstaciones = sc.nextInt(); // Lectura del número de estaciones
    crearEstaciones(); // Creación de las estaciones
    rellenarEstaciones(); // Relleno de las estaciones
    mostrarDatos(); // Muestra los datos que se envíarán por cada estación
    mostrarTramas(); // Obtiene y muestra las tramas
}
```

Ilustración 26 - Main

Y listo, terminamos nuestro método **main(String[] args)** y el principal de toda la clase.

Ahora hemos terminado con todos los métodos y la ejecución de todo el programa consiste en ejecutar este dicho último método. Y sería todo por detallar sobre el diseño de la solución.

HERRAMIENTAS UTILIZADAS

- Computadora con Sistema Operativo Windows o donde te permite tener Java
- Java 1.8 para compilar y ejecutar programas .java
- IntelliJ IDEA o Blue J como IDE

CÓDIGO

```
/**
 * Programa de Unidad 4 y 5
 * @author: García García José Ángel
 * @version: 22/12/2020
 *
 * */

import java.util.*;

public class ProgramaUnidad {

    private static ArrayList<Estacion> estaciones; // Array de estaciones
    private static int numEstaciones; // Número de estaciones
    static Scanner sc = new Scanner(System.in); // Scanner para leer de teclado

    /**
     * Función principal del programa para ejecutar todos las fucniones
     * */
    public static void main(String[] args) {
        System.out.print("Introduce el numero de estaciones: ");
        numEstaciones = sc.nextInt(); // Lectura del número de estaciones
        crearEstaciones(); // Creación de las estaciones
        rellenarEstaciones(); // Relleno de las estaciones
        mostrarDatos(); // Muestra los datos que se envíarán por cada estación
        mostrarTramas(); // Obtiene y muestra las tramas
    }
```

```
    /**
     * Crea las estaciones con su respectivo número de caracteres a transmitir
     * */
    public static void crearEstaciones(){
        estaciones = new ArrayList<>(numEstaciones); // Crea el array para almacenar las estaciones
        System.out.println("\n************************************************************\n");
        for(int i = 1; i <= numEstaciones; i++) {
            System.out.print("Cantidad de caracteres a transmitir la estación [" + i + "] = ");
            estaciones.add(new Estacion(sc.nextInt(), i));  // Crea la estación y la Agrega a estaciones
        }
        System.out.println("\n************************************************************\n");
    }

    /**
     * Rellena todas las estaciones ejecutando su método para agregar caracteres de cada estación
     * */
    public static void rellenarEstaciones(){
        estaciones.stream().forEach(estacion -> estacion.addCaracters());
    }

    /**
     * Genera las tramas de acuerdo con un TDM Sincrona
     * */
    public static String generarTramaSincrona(){
        String tramas = "", trama = ""; // Variables para almacenar todas las tramas y cada trama
```

```
        int bit_tramado = 1; // Para indicar el bit de tramado
        for(int t = getMaxCaracter() - 1; t >= 0; t--){
            trama = "["; // Indica el inicio de una trama
            for(int i = numEstaciones - 1; i >= 0; i--){
                ArrayList<Character> datos = estaciones.get(i).getData(); // Se obtiene los caracteres de la estación
                try {
                    trama += " " + datos.get(t) + " |"; // Lo colocamos en la trama si es que tiene
                }catch (Exception error){
                    trama += " " + "_" + " |"; // Colocamos un _ si no tiene caracter para transmitir
                }
            }
            trama = trama.substring(0 ,trama.length() - 1); // Quitar el | al final de cada trama
            tramas += trama + "] [" + (bit_tramado = Math.abs(bit_tramado - 1)) + "] --- "; // Concatenamos a las tramas y su bit de tramado
        }
        return tramas;
    }

    /**
     * Calcula y retorna el mayor número de caracteres que envía una estación, para saber cuantas tramas se generan en TDM Asincrono
     * */
    public static int getMaxCaracter(){
        return estaciones.stream().mapToInt(Estacion::getNumCaracter).max().getAsInt();
    }

    /**
     * Calcula y retorna los bits transmitidos que son utiles en TDM Sincrona
```

```
     * */
    public static int getBitsUtilesSincrona(){
        return  estaciones.stream().mapToInt(Estacion::getBits).sum();
    }

    /**
     * Calcula y retorna los bits que se han transmitido
     * */
    public static int getBitsTransmitidosSincrona(){
        return getMaxCaracter() * numEstaciones * 8 + getMaxCaracter();
    }

    /**
     * Genera las tramas de acuerdo con un TDM Asincrona
     * */
    public static String generarTramaAsincrona(){
        StringBuilder tramas = new StringBuilder(); // Almacena las tramas
        int bit_tramado = 1, // Para indicar el bit de tramado
            numRanuras = getNumRanurasTiempoAsincrona(), // Para saber en cuantas ranuras partir la trama
            numTramas = getNumTramasAsincrona(); // El número de tramas a generar
        ArrayList<Queue<Character>> caracteres = new ArrayList<>(); // Contiene todos los caracteres que se envían
        for(int i = 0; i < numEstaciones; i++)
            caracteres.add(new LinkedList<>(estaciones.get(i).getData()));
        StringBuilder tmpTrama = new StringBuilder(); // La trama temporal
        int index = 0; // Para controla el index de cada caracter
        while(hayTodavia(caracteres)){
```

```
            index = index >= numEstaciones ? 0 : index;
            try {
                Queue<Character> datos = caracteres.get(index++);
                tmpTrama.append(datos.remove().toString() + (index)); // Se guarda todos los caracteres en
            }catch (Exception e){}
        }
        tramas.append("]1["); // Para la parte inicial, siempre inicia en bit tramado 1
        for(int i = 0, aux = 0; i < tmpTrama.length() - 1; i+=2){
            tramas.append( aux == 0 ? "] " : ""); // Inicia una trama dependiendo del número de rodajas
            tramas.append(" | " + tmpTrama.charAt(i+1) + "" + tmpTrama.charAt(i)); // Agrega el caracter a la trama
            aux += 1; // Indica que ya se colocó un caracter en la trama
            if(aux == numRanuras && numTramas != 0){ // Valida si es momento de cerrar la trama
                // Cierra la trama con su respectivo bit de tramado
                tramas.append( i < tmpTrama.length() - 2 ? " [ ---- ]" + (bit_tramado = Math.abs(bit_tramado - 1)) + "[" : "");
                aux = 0; // Reinicia las ranuras hechas
                numTramas--; // Decrementa las tramas, indicando que se completó una
            }
            tramas.append(i == tmpTrama.length() - 2 ? " [" : ""); // Cierra la última trama recien calculada y la agrega a tramas
        }
        StringBuilder nuevo = new StringBuilder(tramas.reverse().toString().replace("|  ]", "]"));
        int faltante = getNumTramasAsincrona() * numRanuras - getNumCaracteresTotales() ;
        for(int i = 1; i <= faltante; i++)
            nuevo.insert(i * 2,"_ ");
        return nuevo.toString(); // Regresa las tramas
```

```
}

/**
 * Verifica si las Queue tienen datos o ya no de acuerdo aun ArrayList<Queue>
 */
public static boolean hayTodavia(ArrayList<Queue<Character>> datos){
    for(Queue<Character> d : datos)
        if(!d.isEmpty())
            return true;
    return false;
}

/**
 * Calcula y retorna las ranuras de tiempo para un TDM Asincrona
 * */
public static int getNumRanurasTiempoAsincrona(){
    int caracteres = getNumCaracteresTotales(),
            numEstacionesEnvian = getNumEstacionesEnvian(),
            aprox = caracteres / numEstacionesEnvian;
    return  aprox < numEstacionesEnvian ? aprox : aprox - 1;
}

/**
 * Calcula y retorna el número de tramas recomendado para una TDM Asincrona
 * */
public static int getNumTramasAsincrona(){
    int caracteres = getNumCaracteresTotales(),
            ranuras = getNumRanurasTiempoAsincrona();
```

```
        return (int)Math.ceil((double)caracteres / ranuras);
    }

    /**
     * Retorna el número de caracteres que se envían por todas las estaciones
     * */
    public static int getNumCaracteresTotales(){
        return estaciones.stream().mapToInt(Estacion::getNumCaracter).sum();
    }

    /**
     * Retorna el número de estaciones que envían datos
     * */
    public static int getNumEstacionesEnvian(){
        return (int)(estaciones.stream().filter(estacion -> estacion.getNumCaracter() >
0).count());
    }

    /**
     * Calcula y retorna el número de bits para direccionamiento de acuerdo a las estaciones
para el TDM Asincrona
     * */
    public static int getBitsDireccionamiento(){
        return 2;
       //return toBinary(numEstaciones).length();
    }

    /**
```

```
     * Convierte un número a binario
     * */
    public static String toBinary(int n){
        if(n == 1) return "1";
        return toBinary(n / 2) + n % 2;
    }

    /**
     * Calcula y retorna los bits que se han transmitido
     * */
    public static int getBitsTransmitidosAsincrona(){
        return (getNumRanurasTiempoAsincrona() * getNumTramasAsincrona()) * (8 +
getBitsDireccionamiento()) + getNumTramasAsincrona();
    }

    /**
     * Calcula y retorna los bits transmitidos que son utiles en TDM Asincrona
     * */
    public static int getBitsUtilesAsincrona(){
        return  getBitsUtilesSincrona();
    }

    /**
     * Permite mostrar las tramas generadas tanto de TDM Sincrona como ASincrona
     * */
    public static void mostrarTramas(){
        System.out.println("\nTRAMA(S) DE TDM SINCRONA: \n");
```

```
        String trama = generarTramaSincrona(); // Obtenemos las tramas mediante TDM Asincrona

        decoracion(trama);

        System.out.println("\n\n" + trama + "\n"); // Mostramos la tramas sincrona

        decoracion(trama);

        System.out.println("\n\nINFORMACIÓN DE TDM SINCRONA: ");

        System.out.println("\n" + getBitsTransmitidosSincrona() + " bps = " + getMaxCaracter()

                + " tramas/segundo x " + (8 * numEstaciones + 1) + " bits/trama"); // Información de la tasa de datos

        System.out.println("Número de bits transmitidos: " + getBitsTransmitidosSincrona()); // Información de bits transmitidos

        System.out.println("Número de bits útiles transmitidos: " + getBitsUtilesSincrona()); // Información de bits utiles


        System.out.println("\n\nTRAMA(S) DE TDM ASINCRONA: \n");

        String tramaAsincrona = generarTramaAsincrona(); // Obtenemos las tramas mediante TDM Asincrona

        decoracion(tramaAsincrona);

        System.out.println("\n\n" + tramaAsincrona + "\n"); // Mostramos la tramas asincrona

        decoracion(tramaAsincrona);

        System.out.println("\n\nINFORMACIÓN DE TDM ASINCRONA: \n");

        System.out.println("Número de bits transmitidos: " + getBitsTransmitidosAsincrona()); // Información de bits transmitidos

        System.out.println("Número de bits útiles transmitidos: " + getBitsUtilesAsincrona()); // Información de bits utiles

        System.out.println("Bits para el direccionamiento: " + getBitsDireccionamiento()); // Información del bit de direccionamiento

    }


    /**
```

```
     * Imprime una linea de asteriscos para mostrar la trama a modo de decoración
     * */
    public static void decoracion(String trama){
        for(int i = 0; i < trama.length(); i++)
            System.out.printf("*");
    }

    /**
     * Muestra los datos que envía cada estación, ejectuando el respectivo método de la estación
     * */
    public static void mostrarDatos(){
        System.out.println("\nDATOS DE CADA ESTACIÓN");
        estaciones.stream().forEach(e -> e.imprimirDatos());
    }

}

/**
 * Clase utilizada como una abstracción de una estación
 * */
class Estacion{

    private ArrayList<Character> data; // Información que transmite
    private int bits, // Cantidad de bits que transmite
                numCaracter, // Número de caracteres de la estacion
                id; // Identificador
```

```
    /**
     * Constructor de una estación
     * */
    public Estacion(int numCaracter, int id){
        this.id = id;
        this.data = new ArrayList<>();
        this.numCaracter = numCaracter;
        bits = numCaracter * 8; // Se supone que cada caracter es de 8 bits
    }

    /**
     * Permite agregar los respectivos caracteres a la estación
     * */
    public void addCaracters(){
        Scanner sc = new Scanner(System.in); // Scanner para leer de teclado
        System.out.println("Caracteres de Estación " + id);
        for(int i = 0; i < numCaracter; i++){
            System.out.printf("Caracter [" + (i + 1) + "] = ");
            data.add(new Character(sc.nextLine().charAt(0))); // Agrega el caracter a los datos de
la estación
        }
        System.out.println("-----------------------------------");
    }

    /**
     * Retorna el número de caracteres que envía la estación
     * */
    public int getNumCaracter(){
```

```
        return numCaracter;
    }

    /**
     * Retorna el total de bits que envía la estación
     * */
    public int getBits(){
        return bits;
    }

    /**
     * Retorna el array de caracteres que envía la estación
     * */
    public ArrayList<Character> getData(){
        return data;
    }

    /**
     * Imprime los datos que envía y como los envía la estación (orden)
     * */
    public void imprimirDatos(){
        System.out.printf("ESTACIÓN " + id + " --> ENVÍA --> [");
        StringBuilder info = new StringBuilder();
        data.forEach(e -> info.append(e.toString()+ " "));
        System.out.printf(info + " ] --> DE LA FORMA --> [" + info.reverse() + " ]\n");
    }
}
```

PRUEBA DE ESCRITORIO

Voy a hacer la prueba de escritorio, con un ejemplo arbitrario. El ejemplo que tomé para hacer la prueba de escritorio es el que se muestra en la siguiente imagen:

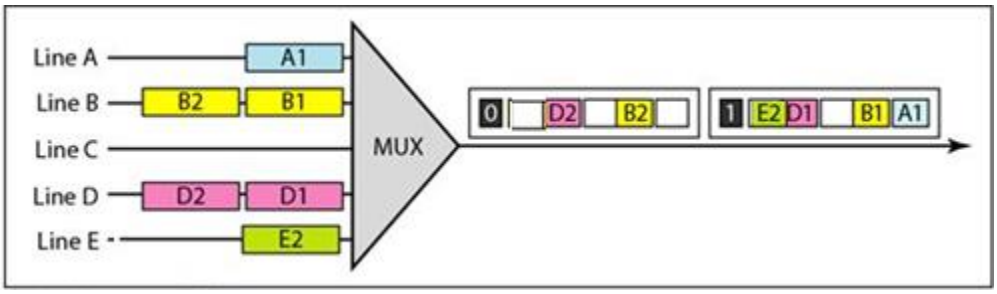

Ilustración 27 - Ejemplo para prueba de escritorio

- El número de estaciones es 5, entonces

  Introduce el número de estaciones: 5

  numEstaciones = 5

- Luego, se procede a ejecutar el método ***crearEstaciones()*** y lo que hace es pedir los datos de cada estación, en la que tenemos el id de la estación el número de caracteres que transmite. La entrada del usuario se vería algo así:

  Cantidad de caracteres a transmitir la estación [1] = 1

  Cantidad de caracteres a transmitir la estación [2] = 2

  Cantidad de caracteres a transmitir la estación [3] = 0

  Cantidad de caracteres a transmitir la estación [4] = 2

  Cantidad de caracteres a transmitir la estación [5] = 1

  Guardamos toda la información en nuestra variable *estaciones*, esto es lo que contiene el ArrayList<Estado> estaciones, tiene en estos momentos 5 instancias de la clase **Estacion**. Lo representamos mediante una tabla lo que contiene *estaciones* donde las filas representan una instancia de Estacion y las columnas id, numCaracter, data y bits indican el valor que tiene cada estación en estos momentos.

Posición en ArrayList	id	numCaracter	data	bits
0	1	*1*		8
1	2	*2*		16
2	3	*0*		0
3	4	*2*		16
4	5	*1*		8

- Podemos observar, que ya tenemos algo de información importante como los bits, lo que nos falta es la *data* , se procede a ejecutar el método de **rellenarEstaciones()** y vamos obtener la parte de data.

Lo que el usuario visualiza es lo siguiente, donde en este caso ya se introdujeron los datos:

Caracteres de Estación 1

Caracter [1] = A

Caracteres de Estación 2

Caracter [1] = B

Caracter [2] = B

Caracteres de Estación 3

Caracteres de Estación 4

Caracter [1] = D

Caracter [2] = D

Caracteres de Estación 5

Caracter [1] = E

Lo que pasa en nuestro ArrayList es que se actualiza la variable data de cada estación. Y se obtiene la siguiente tabla, en la que data ya tiene la información, nótese que data es un ArrayList<Character>.

Posición en ArrayList	id	numCaracter	data	bits
0	1	1	[‘A’]	8
1	2	2	[‘B’, ‘B’]	16
2	3	0	[‘‘]	0
3	4	2	[‘D’, ‘D’]	16
4	5	1	[‘E’]	8

- Luego se ejecuta el método de ***mostrarDatos()*** que simplemente mostrá los datos que se envían y el orden, entonces, al ejecutar ese método se despliega en pantalla.

  DATOS DE CADA ESTACIÓN
  ESTACIÓN 1 --> ENVÍA --> [A] --> DE LA FORMA --> [A]
  ESTACIÓN 2 --> ENVÍA --> [B B] --> DE LA FORMA --> [B B]
  ESTACIÓN 3 --> ENVÍA --> [ ] --> DE LA FORMA --> [ ]
  ESTACIÓN 4 --> ENVÍA --> [D D] --> DE LA FORMA --> [D D]
  ESTACIÓN 5 --> ENVÍA --> [E] --> DE LA FORMA --> [E]

- Ahora ejecutamos el método de ***mostrarTrama()***. Este método mostró al usuario

  TRAMA(S) DE TDM SINCRONA:

Luego va a ejecutar el método ***generarTramaSincrona()***. Por lo que esto nos lleva a lo siguiente, en el que vamos desglosar como funciona mediante una tabla. En la que se podrá observar mejor cada iteración. Antes de ejecutar el primer for, se observa que se declara las variables iniciales

tramas = “”, trama = ““, bit_tramado = 1

Y recordando, en el for, se ejecuta el método ***getMaxCaracter()***. Entonces antes de hacer la tabla, vamos que valor nos retorna este método.

- Lo que hace, es prácticamente tomar los valores del array *estaciones* y lo recorre hasta encontrar el máximo. Y lo único que recorre es como lo que se muestra en la tabla:

Posición de estación en ArrayList	getNumCaracter
0	1
1	***2***
2	0
3	***2***
4	1

- Y de los valores de la columna ***getNumCaracter()*** toma el valor máximo de todos ellos, en este caso se trata del valor ***2*** y es lo que retorna.

Entonces, podemos decir que ***getMaxCaracter()*** siempre nos va a retornar ***2*** y si revisamos, el primer for le resta 1 a este valor para inciar el valor de *t,* entonces ***t = 2 – 1 = 1***. Y el segundo for ocupa *numEstaciones* que tiene el valor de 5 pero como se le resta 1, entonces el segundo for inicia con ***i = 4.*** Ahora sí, podemos proceder a ejecutar nuestra demás parte del código para generar la trama.

t	*tramas*	*i*	*bit_tramado*	*datos = estaciones.get(i).getData( )*	*datos.get(t)*	*¿error?*	*Trama*
1	“ ”	4	1	[‘E’]	null	SI	“[_ |”
1	“”	3	1	[‘D’, ‘D’]	‘D’	NO	“[_	D	“
1	“”	2	1	[‘]	Null	SI	“[_	D	_	”
1	“”	1	1	[‘B’, ‘B’]	‘B’	NO	“[_	D	_	B	”
1	“”	0	0	[‘A’]	null	SI	“[_	D	_	B	_	”
0	“[_	D	_	B	_] [0] ---”	4	0	[‘E’]	“E’	NO	“[E	”
0	“[_	D	_	B	_] [0] ---”	3	0	[‘D’, ‘D’]	‘D’	NO	“[E	D	”
0	“[-	D	_	B	-] [0] ---”	2	0	[‘]	null	SI	“[E	D	_	”
0	“[_	D	_	B	_] [0] ---”	1	0	[‘B’, ‘B’]	‘B’	NO	“[E	D	_	B	”
0	“[_	D	_	B	_] [0] ---”	0	0	[‘A’, ‘A’]	‘A’	NO	“[E	D	_	B	A |”
0	“[_ | D | _ | B | _] [0] --- [E | D | _ | B | A] [1] ---”	0	1	YA NO SE EJECUTA			

- Entonces, el resultado final de este método ***generarTrama()***, es que el valor de la variable tramas es :
  - *tramas* = “[_ | D | _ | B | _] [0] --- [E | D | _ | B | A] [1] ---”

Lo que ejecutó el método anterior, se almacena en una variable llamada *trama* y con esta usamos para obtener su longitud e imprimir unos asteriscos mediante ***decoracion()***, de tal forma que visualiza.

****

Y aquí se imprime el valor de la variable *trama*

[_ | D | _ | B | _] [0] --- [E | D | _ | B | A] [1] ---

Y nuevamente se muestra la línea de asteriscos mediante ***decoracion()***.

****

- Luego muestra la información respecto a los bits. Para eso se muestra en pantalla al usuario:

  INFORMACIÓN DE TDM SINCRONA::

- Y ahora se muestra información respecto a la tasa de datos. Esto se muestra utilizando los métodos de **getBitsTransmitidosSincrona()**
  - Este método, lo que hace es:
    Llamar a **getMaxCaracter() = 2** , luego usar a ***numEstaciones = 5*** y realizar la operación:
    - *getMaxCaracter() * numEstaciones * 8 + getMaxCaracter()*
  - De tal forma que si sustuimos los valores:
    - ***2 * 5 * 8 + 2 = 82***
  - Y este valor es el que retorna y es el que indica el tota de bits transmitidos. Y el otro valor que se calcula ahí es *8 * numEstaciones + 1* el cual nos da de resultado ***41***.
  - De tal forma que el usuario visualiza esto:

    82 bps = 2 tramas/segundo x 41 bits/trama

- Luego se muestra el número de bits transmitidos, que anteriormente ya se había explicado como lo calcula, entonces este método retorna el valor de ***82***. Y, por ende, la salida que ve el usuario es:

  Número de bits transmitidos: 82

- Finalmente, se muestra el número de bits útiles transmitidos, para esto se ejecuta el método ***getBitsUtilesSincrona()***.
  - Este método lo que haces recorrer todo el Array y sumar los números de bits que tiene cada estación.

Posición en ArrayList	bits
0	8
1	16
2	0
3	16
4	8

  - Donde lo que hace, es sumar todos los valores de bits, esta suma da el total de ***48***. Y ese es el valor que retorna el método ***getBitsUtilesSincrona()***.

- Entonces, el usuario finalmente ve una salida como la siguiente:

  Número de bits útiles transmitidos: 48

- Y ahora, se muestra la otra parte que corresponde a la TDM Asíncrona, que primeramente.

  TRAMA(S) DE TDM ASINCRONA:

- Luego se genera la trama, esto mediante el método ***generarTramaAsincrona(),*** que tiene el siguiente funcionamiento.
  - Primero, se declaran las variables tramas

    tramas = "", trama = "", bit_tramado = 1,

    Y aquí mismo se ejecuta el método ***getNumRanurasTiempoAsincrona()*** y ***getNumTramasAsincrona()***.

    Donde el primero método obtiene:

    numRanuras = 1

    Y el segundo obtiene:

    numTramas = 6

  - Luego se crea un ArrayList de Queue y se le asigna el nombre de caracteres. Luego se rellena mediante los datos que tienen la estación. Este contendrá algo como la siguiente imagen.

Index	Queue
0	['A']
1	['B','B']
2	[]
3	['D','D']
4	['E']

  - Ahora, lo que se hace es vaciar esas colas, aplicando el principio de FIFO, ya que el TDM Asíncrona es como un sistema de colas, bajo esa idea es que se vacian las colas en un StringBuilder llamado *tmpTrama*

tmpTrama	hayTodavia(caracteres)	index	Index >= nunEstaciones	datos	¿error?	datos.remove()	datos
“”	true	0	false	[‘A’]	No	‘A’	[]
A	true	1	false	[‘B’,’B’]	No	‘B’	[‘B’]
A1B2	true	2	false	[]	Si	’	
A1B2	true	3	false	[‘D’,’D’]	No	‘D’	[‘D’]
A1B2D4	true	4	true	[‘E’]	No	‘E’	[]
A1B2D4E5	true	0	false	[]	SI		
A1B2D4E5	true	1	false	[‘B’]	No	‘B’	[]
A1B2D4E5B2	true	2	false	[]	Si		
A1B2D4E5B2	true	3	false	[‘D’]	No	‘D’	[]
A1B2D4E5B2D4	false	NO SE EJECUTA					

- Al final, la variable tmpTrama, tiene los datos de A1B2D4E5B2D4
- Se agrega el primer bit de paridad para el inicio a la variable tramas, entonces tramas = “]1[“
- Se inicia el bucle for, esto se visualiza en la tabla

i	aux	aux == 0	numTramas	bitTramado	tmpTrama.charAt(i+1)	tmpTrama.charAt(i)	aux == numRanuras && numTramas != 0	i == tmpTrama.length()-2	tramas
0	0	true	6	0	1	A	true	false	“]1[] | 1A [----]0[”
2	0	true	5	1	2	B	true	false	“]1[] | 1A [----]0[] | 2B [----]1[”
4	0	true	4	0	4	D	true	false	“]1[] | 1A [----]0[] | 2B [----]1[] | 4D [----]0[”
6	0	true	3	1	5	E	true	false	“]1[] | 1A [----]0[] | 2B [----]1[] | 4D [----]0[] | 5E [----]1[”
8	0	true	2	0	2	B	true	false	“]1[] | 1A [----]0[] | 2B [----]1[] | 4D [----]0[] | 5E [----]1[] | 2B [----]0[”
10	0	true	1	1	4	D	true	true	“]1[] | 1ª [----]0[] | 2B [----]1[] | 4D [----]0[] | 5E [----]1[] | 2B [----]0[] | 4D [”

- En este punto, tramas está invertido si lo observamos bien en la tabla, esto porque el se recorre de forma inversa tmpTram. Lo que se hace es invertirla con *reverse* y luego se remplaza “|]” que tiene de más por simplemente “]”. Y esto se alamacena en *nuevo*.
- Ahora, nuevo ya tiene casi toda nuestra trama bien, solo faltaría verificar si es que faltan espacios por agregar, según sea el caso.

- Entonces, se calcula esto, del total de caracteres enviados menos los caracteres que ya se han colocado. Esto se almacena en una variable llamada *faltante.*
- Se hace un recorrido para rellenar lo faltante en la trama con "_" para indicar que son vacíos, pero esto depende si la trama lo necesita, es decir, si faltante es mayor que 0.
- Finalmente, se retorna el valor de *nuevo.* Que justo ahora nuevo es ya la trama:

[D4][0] ---- [B2][1] ---- [E5][0] ---- [D4][1] ---- [B2][0] ---- [A1][1]

- Una vez calculada la trama de TDM Asíncrona, se almacena en *tramaAsincrona*.
- Se ejecuta el método ***decoracion()*** que muestra una línea de asteriscos.

****

- Ahora, se imprime la variable *tramaAsincrona* y es la que contiene la trama recién calculada.

[D4][0] ---- [B2][1] ---- [E5][0] ---- [D4][1] ---- [B2][0] ---- [A1][1]

- Se vuelve a ejecutar el método ***decoración()*** para indicar que se ha mostrado la trama.

****

- Luego se imprime:

  INFORMACIÓN DE TDM ASINCRONA:

- Se ejecuta el método ***getBitsTransmitidosAsincrona()***, ***getBitsUtilesAsincrona()*** y ***getBitsDireccionamiento()***. Donde el valor del **getBisTransmitidosAsincrona(),** se calcula mediante:

  - 6 * (8 + 2) + 6 = **66**
  - El otro reutiliza el del TDM Sincrono, por lo que igual es **48**

- Entonces, lo que usuario ve al final:

  Número de bits transmitidos: 66

  Número de bits útiles transmitidos: 48

  Bits para el direccionamiento: 2

- Finaliza el método ***mostrarTramas()*** y también el programa.

RESULTADOS OBTENIDOS

Para la demostración de los resultados, voy a apoyarme del libro.

PRUEBA 1

En este caso para visualizar la trama generada así coma del siguiente esquema:

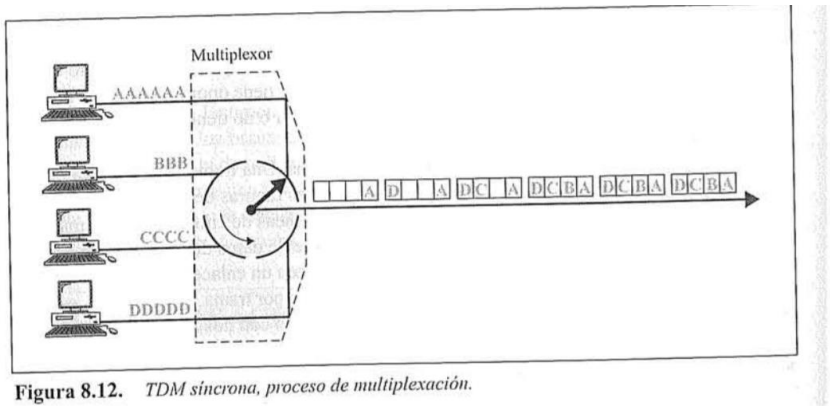

Figura 8.12. *TDM síncrona, proceso de multiplexación.*

Ilustración 28 - Prueba 1

Los datos de estas que nos interesa, son que tenemos ***4*** estaciones y en la primera estación se envían ***6*** caracteres, en la segunda se envían ***3***, en la tercera se envían ***4*** y en la última se envía ***5***. Con esos datos podemos ejecutar el programa y visualizar los resultados.

Al ejecutar podemos rellenar los datos con los que detallamos previamente.

```
Introduce el numero de estaciones: 4

************************************************************

Cantidad de caracteres a transmitir la estación [1] = 6
Cantidad de caracteres a transmitir la estación [2] = 3
Cantidad de caracteres a transmitir la estación [3] = 4
Cantidad de caracteres a transmitir la estación [4] = 5

************************************************************

Caracteres de Estación 1
Caracter [1] = A
Caracter [2] = A
Caracter [3] = A
Caracter [4] = A
Caracter [5] = A
Caracter [6] = A
-------------------------------------
Caracteres de Estación 2
Caracter [1] = B
Caracter [2] = B
Caracter [3] = B
-------------------------------------
Caracteres de Estación 3
Caracter [1] = C
Caracter [2] = C
Caracter [3] = C
Caracter [4] = C
-------------------------------------
Caracteres de Estación 4
Caracter [1] = D
Caracter [2] = D
Caracter [3] = D
Caracter [4] = D
Caracter [5] = D
```

Ilustración 29 - Parte 1 de prueba 1

Luego podemos visualizar que el programa nos genera los mismos datos que el libro con respecto a los datos que envía cada estación. Y también se puede observar la trama respecto al TDM Síncrono, en el que se puede comprobar que la trama generada es la misma que la del libro. Inclusive el bit de tramado corresponde a que va intercalado.

```
 DATOS DE CADA ESTACIÓN
ESTACIÓN 1 --> ENVÍA --> [ A A A A A A ] --> DE LA FORMA --> [ A A A A A A ]
ESTACIÓN 2 --> ENVÍA --> [ B B B ] --> DE LA FORMA --> [ B B B ]
ESTACIÓN 3 --> ENVÍA --> [ C C C C ] --> DE LA FORMA --> [ C C C C ]
ESTACIÓN 4 --> ENVÍA --> [ D D D D D ] --> DE LA FORMA --> [ D D D D D ]

TRAMA(S) DE TDM SINCRONA:

*************************************************************************************************************************************************************

[ _ | _ | _ | A ] [0] --- [ D | _ | _ | A ] [1] --- [ D | C | _ | A ] [0] --- [ D | C | B | A ] [1] --- [ D | C | B | A ] [0] --- [ D | C | B | A ] [1] ---

*************************************************************************************************************************************************************

INFORMACIÓN  DE TDM SINCRONA:

198 bps = 6 tramas/segundo x 33 bits/trama
Número de bits transmitidos: 198
Número de bits útiles transmitidos: 144
```

Ilustración 30 - Parte 2 de prueba 1

Y finalmente, se puede observar la TDM Asíncrona. Que, si verificamos los bits útiles transmitidos, son los mismos, lo que varía es la cantidad de bits transmitidos y también la trama.

```
TRAMA(S) DE TDM ASINCRONA:

*************************************************************************************************************************************************************

[ A1 | D4 | A1 ][0] ---- [ D4 | C3 | A1 ][1] ---- [ D4 | C3 | B2 ][0] ---- [ A1 | D4 | C3 ][1] ---- [ B2 | A1 | D4 ][0] ---- [ C3 | B2 | A1 ][1]

*************************************************************************************************************************************************************

INFORMACIÓN DE TDM ASINCRONA:

Número de bits transmitidos: 186
Número de bits útiles transmitidos: 144
Bits para el direccionamiento: 2
```

Ilustración 31 - Parte 3 de prueba 1

Se puede confirmar que el programa funciona correctamente, luego de realizar esta primera prueba.

PRUEBA 2

Ahora, haré la prueba para el TDM Asíncrono. Para eso, tomaré la siguiente imagen del libro.

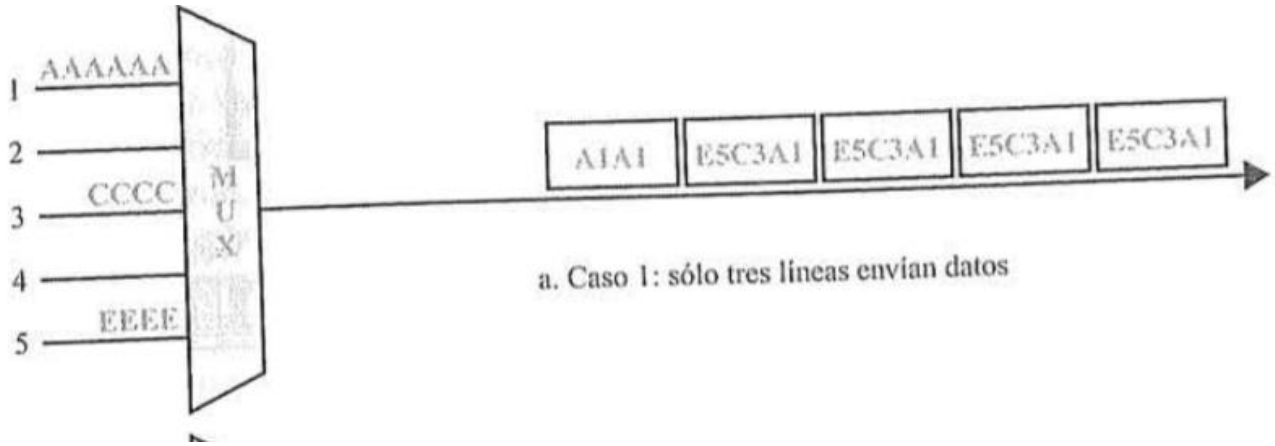

Ilustración 32 - Prueba 2

Entonces, rellenando los respectivos datos.

```
Introduce el numero de estaciones: 5

************************************************************

Cantidad de caracteres a transmitir la estación [1] = 6
Cantidad de caracteres a transmitir la estación [2] = 0
Cantidad de caracteres a transmitir la estación [3] = 4
Cantidad de caracteres a transmitir la estación [4] = 0
Cantidad de caracteres a transmitir la estación [5] = 4

************************************************************

Caracteres de Estación 1
Caracter [1] = A
Caracter [2] = A
Caracter [3] = A
Caracter [4] = A
Caracter [5] = A
Caracter [6] = A
-------------------------------------
Caracteres de Estación 2
-------------------------------------
Caracteres de Estación 3
Caracter [1] = CC
Caracter [2] = C
Caracter [3] = C
Caracter [4] = C
-------------------------------------
Caracteres de Estación 4
-------------------------------------
Caracteres de Estación 5
Caracter [1] = E
Caracter [2] = E
Caracter [3] = E
Caracter [4] = E
```

Ilustración 33 - Parte 1 de la prueba 2

Y ahora, si vemos el resto arrojado por el programa.

```
DATOS DE CADA ESTACIÓN
ESTACIÓN 1 --> ENVÍA --> [ A A A A A A ] --> DE LA FORMA --> [ A A A A A A ]
ESTACIÓN 2 --> ENVÍA --> [ ] --> DE LA FORMA --> [ ]
ESTACIÓN 3 --> ENVÍA --> [ C C C C ] --> DE LA FORMA --> [ C C C C ]
ESTACIÓN 4 --> ENVÍA --> [ ] --> DE LA FORMA --> [ ]
ESTACIÓN 5 --> ENVÍA --> [ E E E E ] --> DE LA FORMA --> [ E E E E ]

TRAMA(S) DE TDM SINCRONA:

************************************************************************************************************************************************************************************

[ _ | _ | _ | _ | A ] [0] --- [ _ | _ | _ | _ | A ] [1] --- [ E | _ | C | _ | A ] [0] --- [ E | _ | C | _ | A ] [1] --- [ E | _ | C | _ | A ] [0] --- [ E | _ | C | _ | A ] [1] ---

************************************************************************************************************************************************************************************

INFORMACIÓN DE TDM SINCRONA:

246 bps = 6 tramas/segundo x 41 bits/trama
Número de bits transmitidos: 246
Número de bits útiles transmitidos: 112


TRAMA(S) DE TDM ASINCRONA:

*********************************************************************************************************************

[ _ A1 | A1 ][1] ---- [ E5 | C3 | A1 ][0] ---- [ E5 | C3 | A1 ][1] ---- [ E5 | C3 | A1 ][0] ---- [ E5 | C3 | A1 ][1]

*********************************************************************************************************************

INFORMACIÓN DE TDM ASINCRONA:

Número de bits transmitidos: 155
Número de bits útiles transmitidos: 112
Bits para el direccionamiento: 2
```

Ilustración 34 - Parte 2 de la prueba 2

Si verificamos la TDM Asíncrona, concuerda con la del libro, entonces esto nos indica que está bien, Además si vemos las diferencias entre la TDM Síncrona y la TDM Asíncrona, se ve claramente como esta última es más efectiva.

PRUEBA 3

Para esta prueba, tomé un ejercicio de internet, ya que me pareció bien explicado todo el tema de Multiplexación, y además que igual toma de base al mismo libro que estamos usando en esta asignatura.

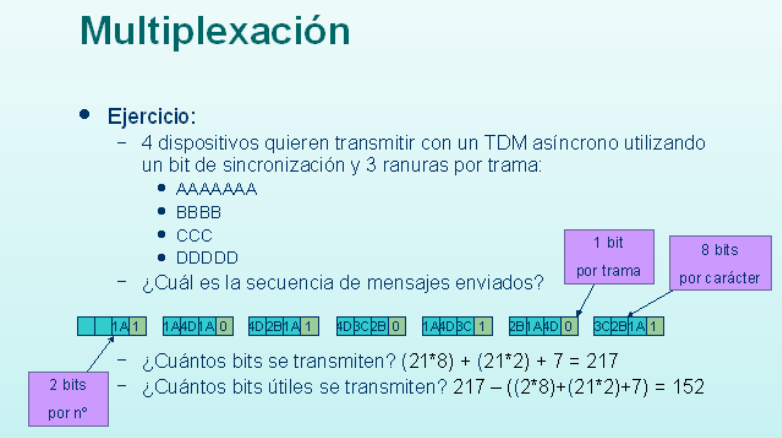

Ilustración 35 - Prueba 3

Introduciendo los respectivos datos

```
Introduce el numero de estaciones: 4

************************************************************

Cantidad de caracteres a transmitir la estación [1] = 7
Cantidad de caracteres a transmitir la estación [2] = 4
Cantidad de caracteres a transmitir la estación [3] = 3
Cantidad de caracteres a transmitir la estación [4] = 5

************************************************************

Caracteres de Estación 1
Caracter [1] = A
Caracter [2] = A
Caracter [3] = A
Caracter [4] = A
Caracter [5] = A
Caracter [6] = A
Caracter [7] = A
-------------------------------------
Caracteres de Estación 2
Caracter [1] = B
Caracter [2] = B
Caracter [3] = B
Caracter [4] = B
-------------------------------------
Caracteres de Estación 3
Caracter [1] = C
Caracter [2] = C
Caracter [3] = C
-------------------------------------
Caracteres de Estación 4
Caracter [1] = D
Caracter [2] = D
Caracter [3] = D
Caracter [4] = D
Caracter [5] = D
-------------------------------------
```

Ilustración 36 - Parte 1 de prueba 3

Y ahora, visualizando el resto:

```
DATOS DE CADA ESTACIÓN
ESTACIÓN 1 --> ENVÍA --> [ A A A A A A A ] --> DE LA FORMA --> [ A A A A A A A ]
ESTACIÓN 2 --> ENVÍA --> [ B B B B ] --> DE LA FORMA --> [ B B B B ]
ESTACIÓN 3 --> ENVÍA --> [ C C C ] --> DE LA FORMA --> [ C C C ]
ESTACIÓN 4 --> ENVÍA --> [ D D D D D ] --> DE LA FORMA --> [ D D D D D ]

TRAMA(S) DE TDM SINCRONA:

*****************************************************************************************************************************************************************************************

[ _ | _ | _ | A ] [0] --- [ _ | _ | _ | A ] [1] --- [ D | _ | _ | A ] [0] --- [ D | _ | B | A ] [1] --- [ D | C | B | A ] [0] --- [ D | C | B | A ] [1] --- [ D | C | B | A ] [0] ---

*****************************************************************************************************************************************************************************************

INFORMACIÓN DE TDM SINCRONA:

231 bps = 7 tramas/segundo x 33 bits/trama
Número de bits transmitidos: 231
Número de bits útiles transmitidos: 152


TRAMA(S) DE TDM ASINCRONA:

*************************************************************************************************************************************************************

[ _ _ A1 ][1] ---- [ A1 | D4 | A1 ][0] ---- [ D4 | B2 | A1 ][1] ---- [ D4 | C3 | B2 ][0] ---- [ A1 | D4 | C3 ][1] ---- [ B2 | A1 | D4 ][0] ---- [ C3 | B2 | A1 ][1]

*************************************************************************************************************************************************************

INFORMACIÓN DE TDM ASINCRONA:

Número de bits transmitidos: 217
Número de bits útiles transmitidos: 152
Bits para el direccionamiento: 2
```

Ilustración 37 - Parte 2 de prueba 3

Se puede comprobar que las tramas de TDM Asíncrona es la misma que la del ejemplo presentado, entonces esto indica que el programa funciona correctamente. Además, que la cantidad de bits transmitidos y bits útiles transmitidos son iguales. Se coloca unos “_” a modo de indicar que faltaron espacios para completar la trama.

ENLACE DEL VÍDEO

El vídeo en el que se explica a detalle cada una de las fases del programa desde su desarrollo hasta su implementación lo puede visualizar en el siguiente enlace:

https://youtu.be/_dI38zGzPyI

Le dejo una tabla para que pueda revisar de forma más rápida el vídeo, ya que es algo extenso, debido a que me di a la tarea de explicar cada parte del código y fuera entendible, así como también detallaba los resultados obtenidos y las consideraciones que tomé al programarlo.

Minuto	Acción
00:50	Ejemplo de TDM Síncrona
8:58	Ejemplo de TDM Asíncrona
15:04	Explicación del bit de direccionamiento
16:22	Explicación del código
17:05	Explicación de la clase Estacion
19:07	Explicación de la clase ProgramaUnidad
22:55	Explicación de metodos previos a TDM Síncrona
26:30	Explicación de la TDM Síncrona
32:34	Explicación de metodos previos a TDM Asíncrona
40:14	Explicación de metodos previos a TDM Asíncrona

CONCLUSIÓN

La evidencia que presenté anteriormente demuestra que en esta práctica se cumplió con el objetivo, que era crear un programa que fuera capaz de realizar Multiplexación por División del Tiempo (TDM) Síncrona y Asíncrona, bueno, el poder mostrar las tramas que se transmitían era lo más importante del programa, además que también se muestra información sobre los bits transmitidos y los bits útiles que se transmiten. Justamente la información presentada en el apartado de resultados obtenidos demuestra las pruebas de ejecución del programa y verifica exitosamente que funciona correctamente, además que se presenta un enlace a un vídeo en el que se explica cada uno de los detalles del programa, haciendo así que la explicación del funcionamiento del programa sea más fácil de entender y que se recalque que funciona correctamente haciendo lo solicitado, es decir, dando solución al planteamiento del problema.

Poder cumplir con realizar este programa fue sencillo, ya que se programó en Java, un lenguaje de programación que se conoce en la carrera desde el principio y a esta altura de la carrera ya se tiene mayor conocimiento sobre el mismo, y habilidades que permiten hacer la optimización del código. Además, que no se implementa una interfaz como tal, por lo que desde esta parte la implementación fue sencilla.

En cuanto a realizar el funcionamiento de la multiplexación y poder generar la trama no fue algo complicado para la síncrona, ya que con explicaciones previamente en clases y un poco de lectura del libro fue suficiente para poder crear un algoritmo capaz de generar las tramas. También que con la práctica de hacer varias tramas e ir probando de forma manual o con ejercicios prácticos, fue que pude optimizar y encontrar una mejor solución, ya que claramente se puede dar solución al problema de muchas formas y la que presenté me pareció la más organizada, adecuada y agradable. Por otra parte, la Asíncrona, fue la que me costó más poder comprender, porque el libro no era tan claro y tuve que buscar mayor información y practicar con algunos ejercicios para poder entender y comprender su funcionamiento. También el pensar cómo organizar todo el código para que funcionara en conjunto fue algo tedioso porque eran muchos los parámetros o características tomar en cuenta, esto va desde obtener las estaciones y las cantidades de caracteres que transmitía cada una, calcular las tramas necesarias a mostrar, el número de rodajas de cada trama y por supuesto, cada una de las tramas a mostrar, la parte de sincronización de los datos, por último el calcular los bits transmitidos y bits útiles transmitidos.

Ahora bien, quedando satisfecho con los resultados obtenidos, el trabajo presentado y reforzando el conocimiento de la asignatura respecto al tema de multiplexación. Me pareció una buena práctica el poder realizar esto, ya que nos da una perspectiva para confirmar el conocimiento adquirido durante clases y también para tener una idea de cómo funciona internamente un TDM síncrono y, sobre todo, para entender mejor el TDM asíncrono. y las diferencias que existe entre cada tipo de multiplexación.