ST0257 – Sistemas operativos
Proyecto 3: Paralelismo con bloqueos en un ambiente controlado

MBA, I.S. José Luis Montoya Pareja
Departamento de Informática y Sistemas
Universidad EAFIT
Medellín, Colombia, Suramérica

RESUMEN
En este proyecto, los estudiantes deberán diseñar e implementar un sistema que sea capaz de manejar múltiples hilos concurrentes, utilizando métodos de control de concurrencia para sincronizar sus acciones. Deberán identificar y evitar posibles abrazos mortales (deadlocks) a través del uso de las técnicas de concurrencia adecuadas. El proyecto pondrá a prueba su capacidad para aplicar conceptos de concurrencia y sincronización en un entorno de programación simulado buscando garantizar la eficiencia.

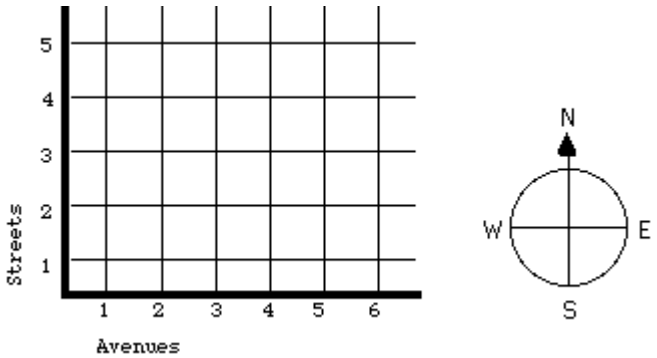
PALABRAS CLAVE
Procesamiento en paralelo, paradigmas de programación en paralelismo, eficiencia, procesos, virtualización, memoria, Metro de medellín.

CONTEXTO
KarelJRobot

Este es un complemento que se monta en Java para tener un entorno donde visualizar el robot y poder programarlo. Lo vamos a utilizar para complementar los conocimientos en Java.

Conceptos básicos

KarelJRobot opera en un mundo que es una malla de calles y avenidas que permiten al robot moverse.



Las calles van de Este a Oeste y las Avenidas van de Norte a Sur. Tiene cosas especiales que el robot puede sensar y manipular. El mundo es esencialmente plano, empieza en la avenida 1 con calle 1 y puede extenderse hasta el infinito

dado que tiene al sur y al oeste muros infinitos que se extienden hacia el norte y hacia el este. El robot no puede atravesar esos muros dados que son de vibranio.

Las intersecciones entre las calles y las avenidas se denominan esquinas. En cada esquina se puede ubicar uno o varios robots, que tienen una brújula interna que les indica a donde están mirando.

Adicionalmente, podemos ubicar en una esquina una sirena (beeper). Las sirenas son unos pequeños conos plásticos que emiten un silencioso sonido de beep. Estas sirenas se pueden recoger, llevar y descargar en otra esquina o en la misma.

El robot es móvil, se puede desplazar en la dirección que está indicando su brújula (Norte, sur, este u oeste) y avanzan un paso a la vez. Pueden percibir eventos desde su entorno usando sensores rudimentarios de vista, sonido, orientación y tacto.

Los robots tienen una cámara que solo mira hacia adelante y está pensada para detectar un muro cuando está exactamente en frente de él (es miope el lente de la cámara). También puede escuchar una sirena cuando está parado encima de ella. Tiene un brazo que le permite recoger del piso las sirenas y colocarlas en su morral y extraerlas del morral y colocarlas en la esquina donde está parado. En el morral puede tener infinitas sirenas (como el morral de Dora, la exploradora). Los siguientes son los métodos disponibles para usar con el robot:

boolean anyBeepersInBeeperBag()

Determina si el robot tiene algún beeper en su bolsa de beepers.

boolean facingEast()

Determine si el robot está mirando hacia el Este o no.

boolean facingNorth()

Determine si el robot está mirando hacia el Norte o no.

boolean facingSouth()

Determine si el robot está mirando hacia el Sur o no.

boolean facingWest()

Determine si el robot está mirando hacia el Oeste o no.

boolean frontIsClear()

Determina si el frente del robot está libre o si está mirando un muro.

boolean nextToABeeper()

Determina si el robot está en la misma esquina de un beeper.

void move()

Mueve el robot al siguiente cruce dependiendo de donde esté mirando.

void pickBeeper()

Recoge un beeper de la esquina donde se encuentra el robot y lo lleva a la bolsa de beepers.

void putBeeper()

Coloca un beeper que se encuentra en la bolsa de beepers y lo coloca en la esquina donde se encuentra el robot.

void turnLeft()

Gira el robot 90 grados a la izquierda.

void turnOff()

Apaga el robot y lo coloca en color gris.

EJERCICIO 1: Instalación de KarelJRobot

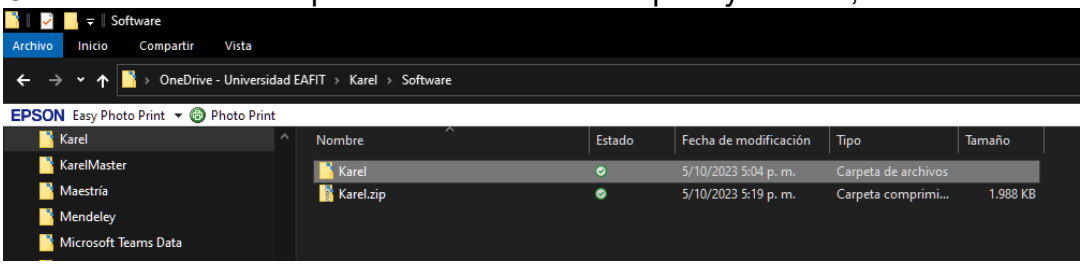
Prerequisitos

Debe tener la última versión del JDK (kit de instalación de Java donde está el compilador de java, javac). Los usuarios de MAC deben poder tener acceso vía consola.

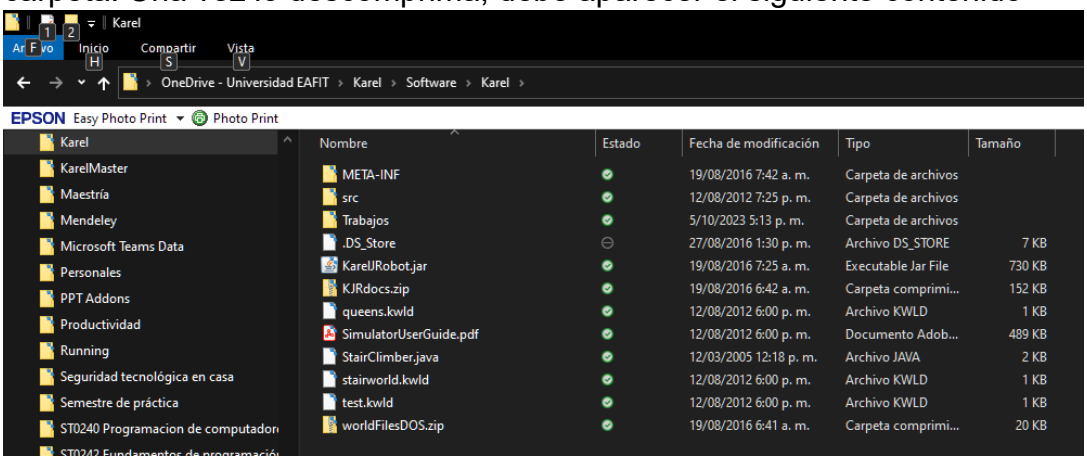
Pasos

1. Vaya al Teams del curso, carpeta Materiales de clase y descargue a su máquina el archivo Karel.zip

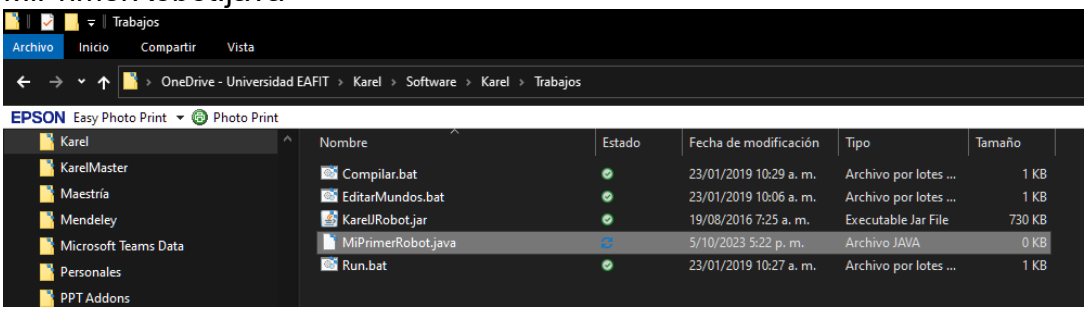
2. Cree en el directorio que usted desee una carpeta y llámela, Karel



3. Extraiga el contenido del archivo KJRDistribution160819.zip en dicha carpeta. Una vez lo descomprima, debe aparecer el siguiente contenido



4. Vaya a la carpeta Trabajos y ahí cree un nuevo archivo Java llamado MiPrimerRobot.java



5. Con el IDE que tenga en su máquina (Visual Studio o NetBeans o el que tenga) edite su programa con el siguiente contenido:

```
import kareltherobot.*;
import java.awt.Color;

public class MiPrimerRobot implements Directions
{
  public static void main(String [] args)
  {
    Robot Karel = new Robot(1, 1, East, 0);
// Coloca el robot en la posición inicial del mundo (1,1),
// mirando al Este, sin ninguna sirena.

// Mover el robot 3 pasos, girar hacia el norte y apagar el robot.
    Karel.move();
    Karel.move();
    Karel.move();
    Karel.turnLeft();
    Karel.turnOff();
  }
}
```

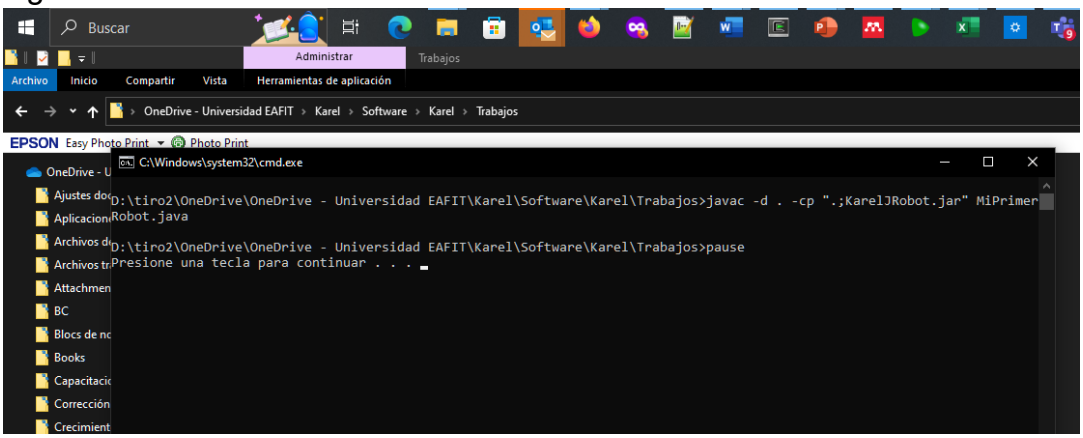
6. Usuarios Windows: Graben el archivo y editen el archivo llamado “Compilar.bat”. El archivo tiene el siguiente contenido:
   javac -d . -cp ".;KarelJRobot.jar" %1
   pause
   exit

   Reemplacen el %1 por el nombre del archivo Java (MiPrimerRobot.java)
   javac -d . -cp ".;KarelJRobot.jar" MiPrimerRobot.java

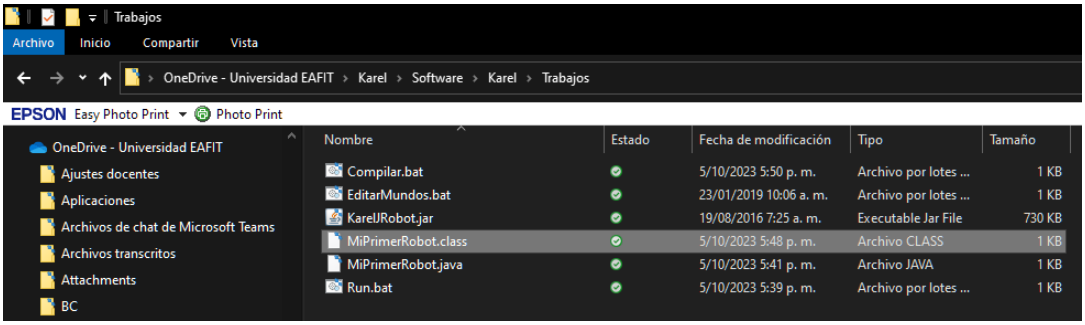
   Usuarios MAC: Deben ejecutar en la carpeta y por la consola, el comando:
   javac -d . -cp ".;KarelJRobot.jar" MiPrimerRobot.java

7. Guarden el archivo y en el Explorador de archivos, den dobleclic al programa para que compile:

8. Si la compilación no tiene errores, debe salir una pantalla negra como la siguiente:



9. Presione cualquier tecla para cerrar la ventana negra. Luego revise que debe haber quedado el archivo MiPrimerRobot.class en la carpeta:



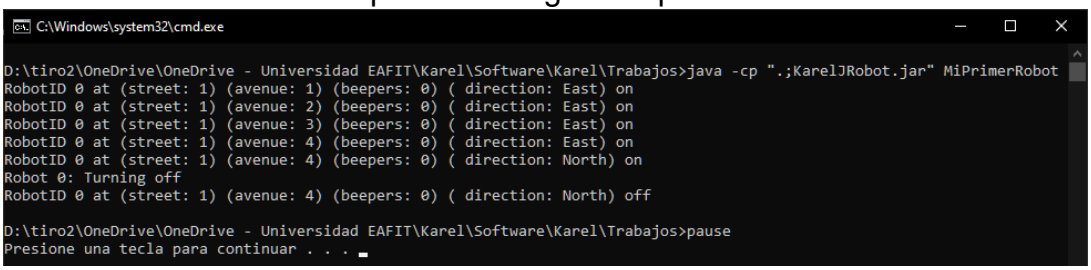
Si esto no sucedió, revise bien los archivos desde el paso 5 para que se aseguren que todo haya quedado como se pide en los pasos anteriores.

10. Usuarios Windows: Edite el archivo “Run.bat” el cual tiene el siguiente contenido:
    java -cp ".;KarelJRobot.jar" %1
    pause
    exit

    Cambie el %1 por el nombre de la clase (MiPrimerRobot), el archivo debe quedar así:
    java -cp ".;KarelJRobot.jar" MiPrimerRobot
    pause
    exit

    Usuarios MAC: en la consola y en la carpeta Trabajos, deben ejecutar el comando:
    java -cp ".;KarelJRobot.jar" MiPrimerRobot

11. Guarden el archivo y en el Explorador de archivos dar dobleclic en el archivo “Run.bat”. Debe aparecer la siguiente pantalla:

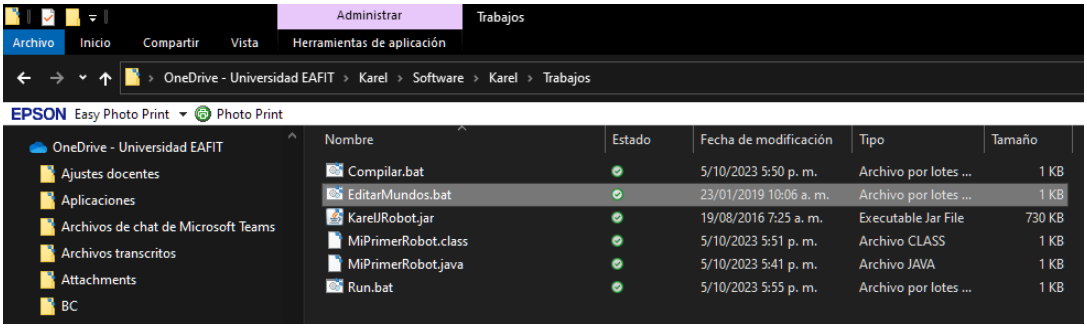


Esto significa que el programa corrió sin problemas. Si no aparecen estos mensajes, primero revise el archivo Run.bat del paso 10 y luego los pasos desde el 5.

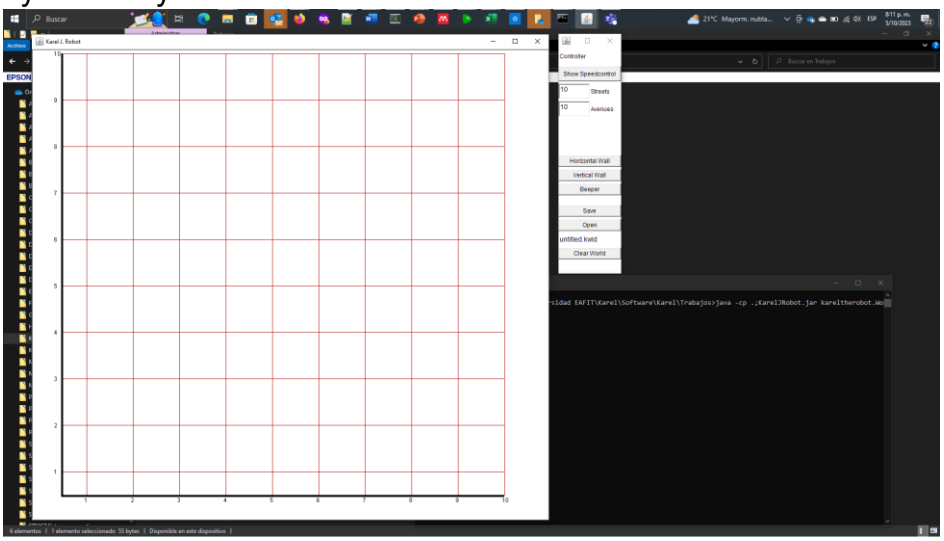
12. Tome un screenshot o una foto de la ejecución de su robot y póngala en la bitácora como evidencia del primer ejercicio.

EJERCICIO 2: Correr el robot visualizando el mundo

1. Use el editor de mundos para crear un mundo. Esto se hace dando dobleclic en el archivo EditarMundos.bat

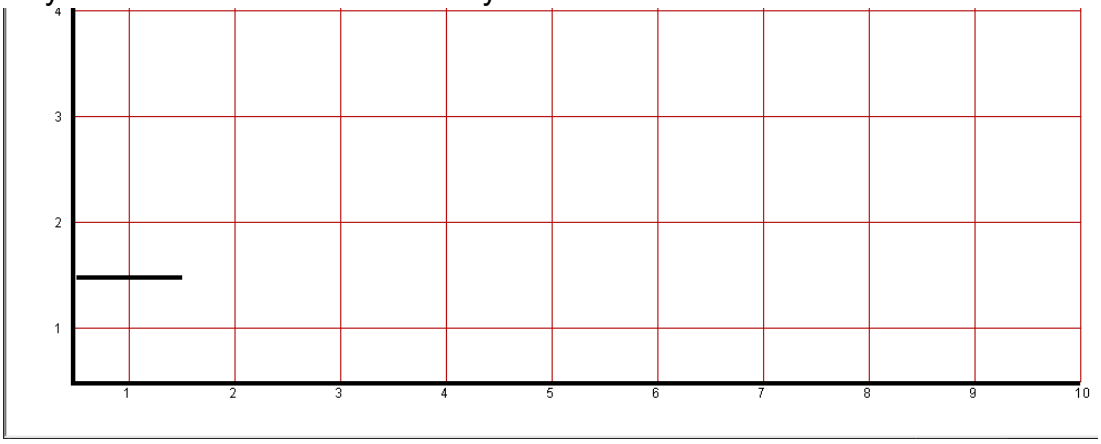


2. Aparecen tres ventanas: primero una negra donde aparecerán mensajes si hay errores y dos blancas donde se muestra el mundo del robot:

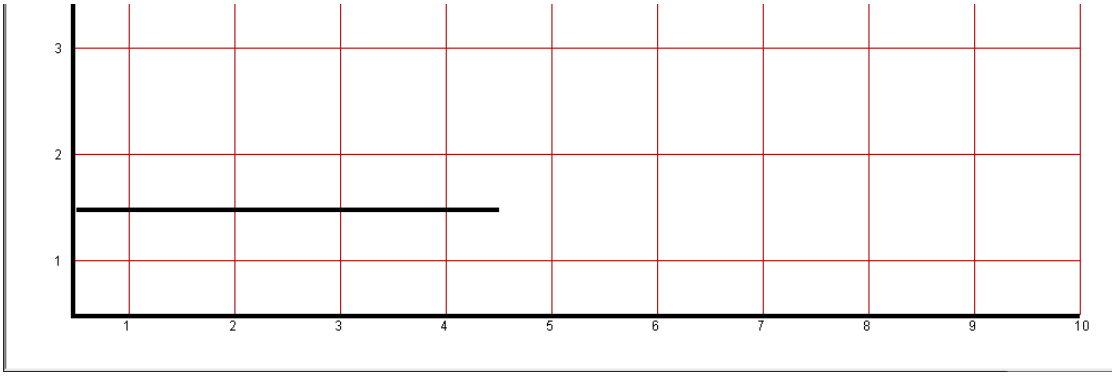


3. En esta interfaz se pueden crear los mundos. Vamos a agregar elementos al mundo. Empezamos con muros horizontales. Damos clic en el botón Horizontal Wall…

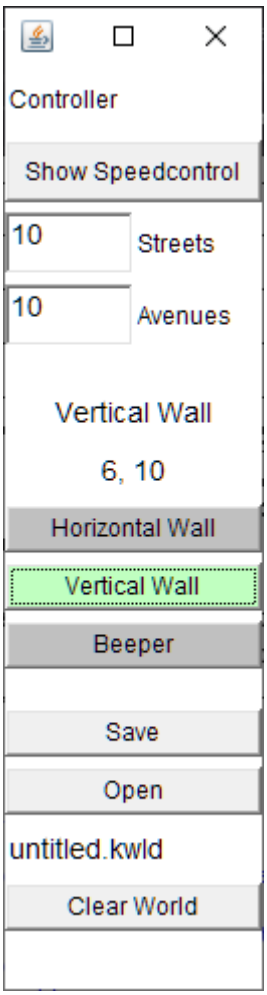
… y damos clic entre la vertical 1 y 2 en la coordenada 1:



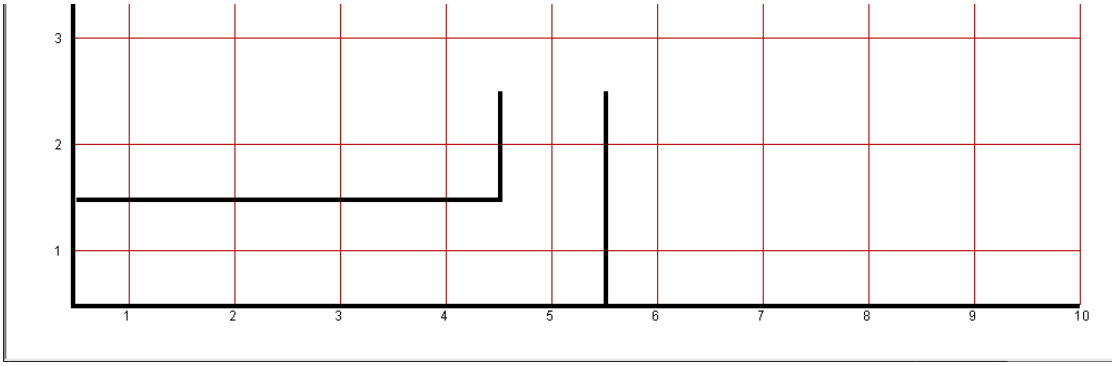
Repetimos lo mismo hasta la avenida 4.



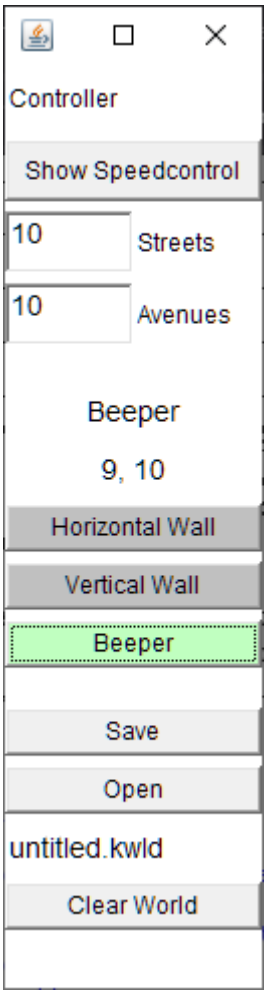
4. Ahora vamos a agregar muros verticales. Damos clic en el botón Vertical Wall…



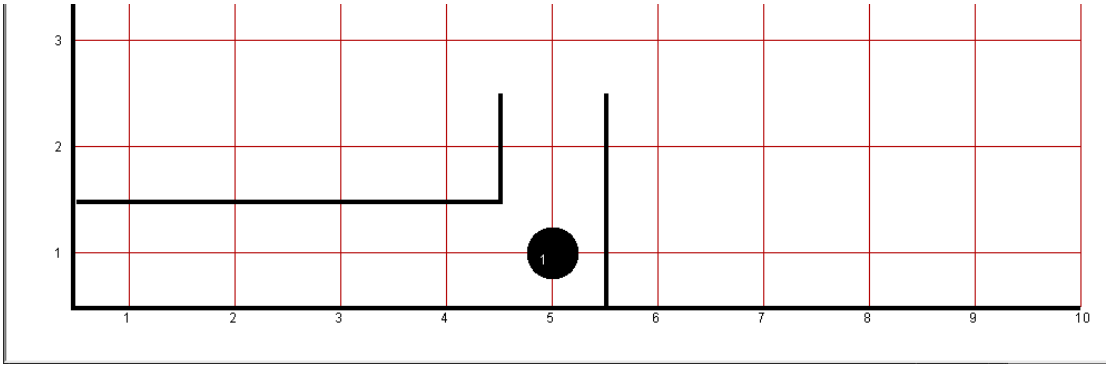
… y agregamos muros entre las avenidas 4 y 5 cruzando la calle 2 y entre las avenidas 5 y 6 cruzando las calles 2 y 1.



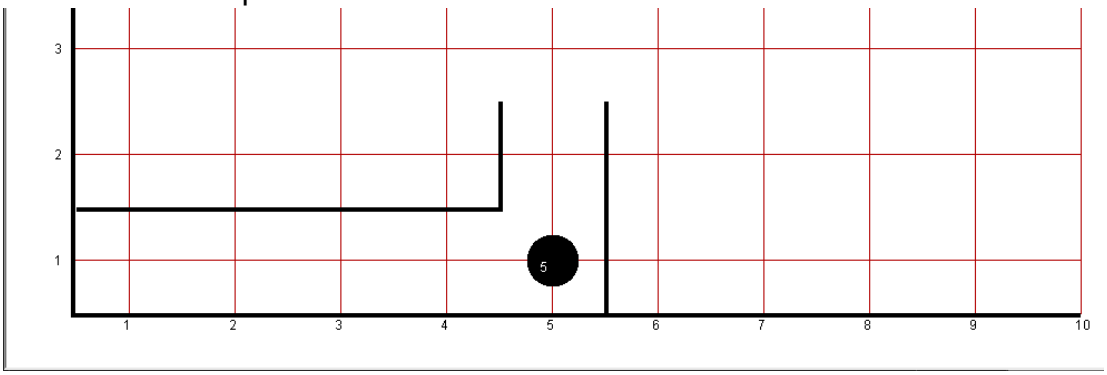
5. Por último, vamos a agregar 5 sirenas en la calle 1 avenida 5. Damos clic en el botón Beeper…



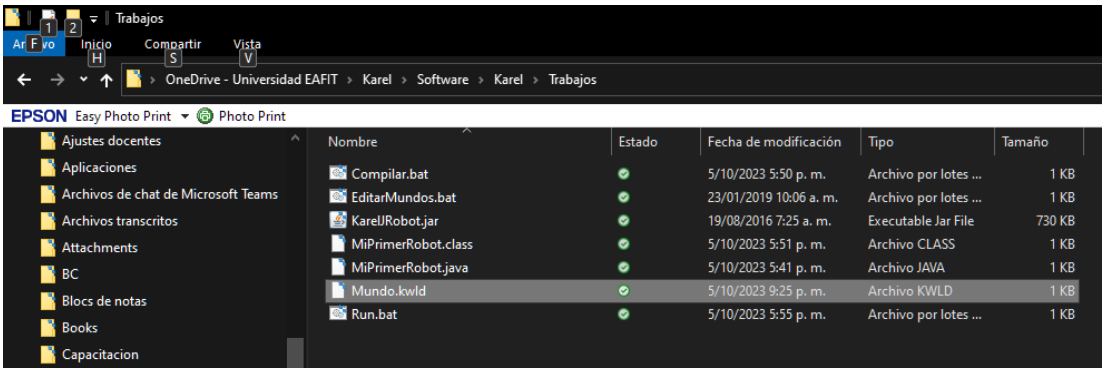
… y damos clic en la avenida 5 calle 1…



… como vemos, aparece un beeper (marcado con el número 1). Damos cuatro clics más para colocar las cinco sirenas.



6. Por último, guardamos el mundo dando clic en el botón Save. Le colocamos el nombre “Mundo.kwld”.

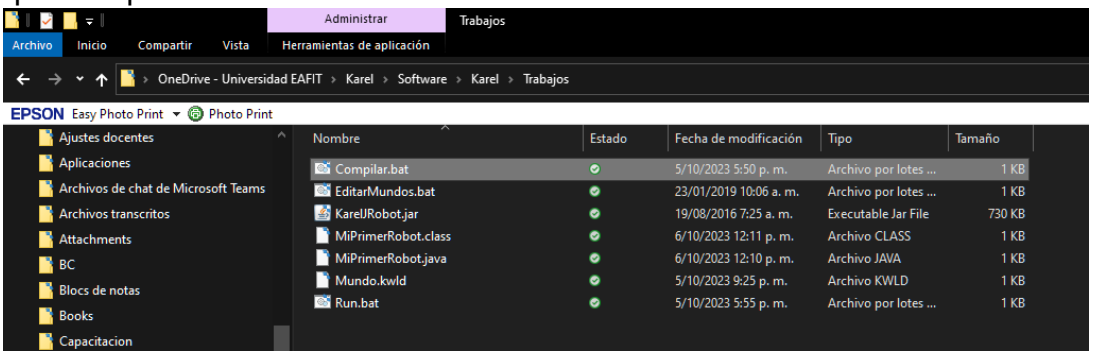


7. Cerramos las ventanas y vamos al código para incorporar el mundo en el código. Debemos modificar el archivo MiPrimerRobot.java de la siguiente manera:

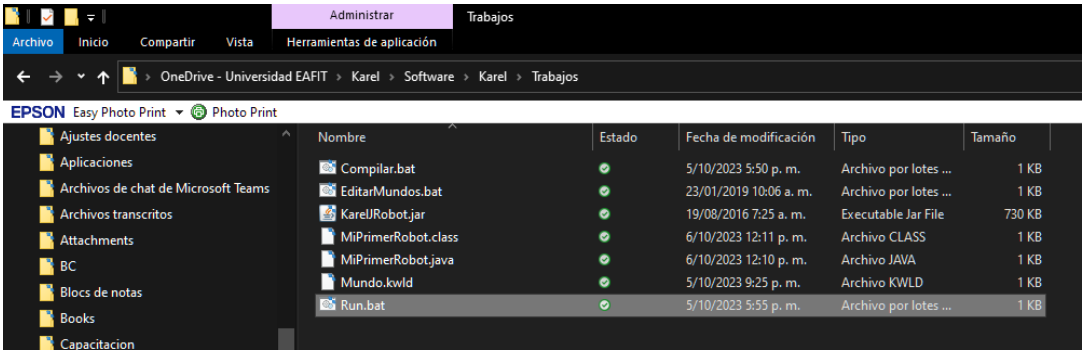
```
import kareltherobot.*;
import java.awt.Color;

public class MiPrimerRobot implements Directions
{
        public static void main(String [] args)
        {
// Usamos el archivo que creamos del mundo
        World.readWorld("Mundo.kwld");
```

```
        World.setVisible(true);

// Coloca el robot en la posición inicial del mundo (1,1),
// mirando al Este, sin ninguna sirena.
                Robot Karel = new Robot(1, 1, East, 0);

// Mover el robot 4 pasos
                Karel.move();
                Karel.move();
                Karel.move();
                Karel.move();
// Recoger los 5 beepers
                Karel.pickBeeper();
                Karel.pickBeeper();
                Karel.pickBeeper();
                Karel.pickBeeper();
                Karel.pickBeeper();
// Girar a la izquierda y salir de los muros
                Karel.turnLeft();
                Karel.move();
                Karel.move();
// Poner los beepers fuera de los muros
                Karel.putBeeper();
                Karel.putBeeper();
                Karel.putBeeper();
                Karel.putBeeper();
                Karel.putBeeper();
// Ponerse en otra posición y apagar el robot
                Karel.move();
                Karel.turnOff();
        }
}
```

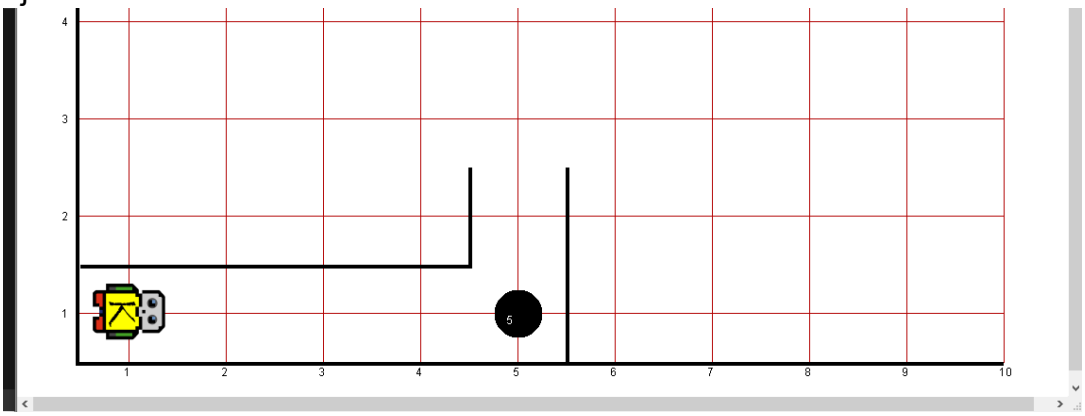
8. En el Explorador de archivos, den dobleclic al programa Compilar.bat para que compile:

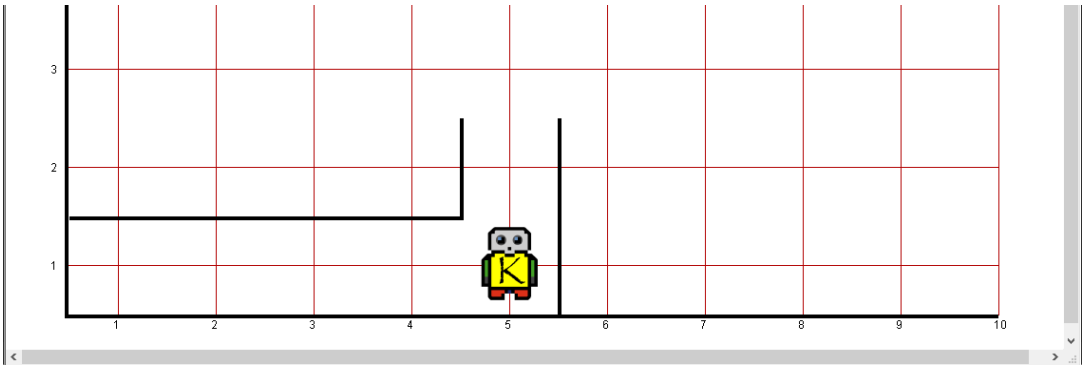


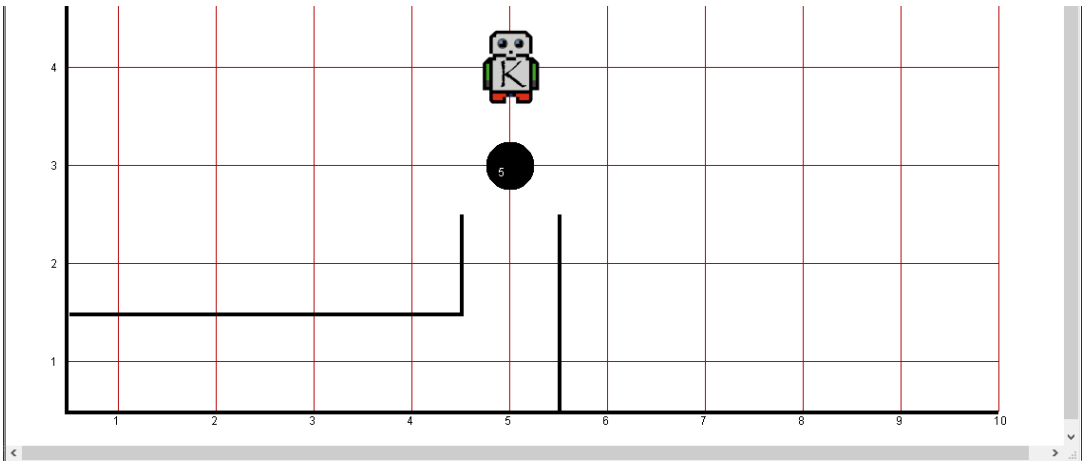
9. Y ejecutamos el programa dando dobleclic en el programa Run.bat para ejecutar el programa:



10. Debe aparecer el mundo y ya el robot en la posición 1,1 y arranca a ejecutar el robot.







EJERCICIO 3

Prerequisitos

Ejercicios 1 y 2

Agregue al código del ejercicio 2, otro robot de color azul que comience en la misma posición del primer robot creado pero que se mueva uno después del otro (en el mismo hilo).

EJERCICIO 4

Prerequisitos

Ejercicio 3 y paralelismo en Java y en el robot.

Para que un robot pueda correr en su propio hilo, se debe crear una clase aparte que extienda la clase Robot:

```
class Racer extends Robot
{
  public Racer(int Street, int Avenue, Direction direction, int beeps)
  {
    super(Street, Avenue, direction, beeps);
    World.setupThread(this);
  }

  public void race()
  {
    while(! nextToABeeper())
      move();
    pickBeeper();
    turnOff();
  }

  public void run()
  {
    race();
  }
}
```

El método que activa el hilo es el run. En el ejemplo, éste llama al método race que es donde está el código que ejecuta el robot en su hilo. Luego, el main queda de la siguiente manera:

```
public static void main(String [] args)
{
  Racer first = new Racer(1, 1, East, 0);
  Racer second = new Racer(2, 1, East, 0);
}
```

Entonces solo es activar dos objetos de la clase Racer para que estos corran en hilos independientes y puedan realizar las tareas de manera independiente.

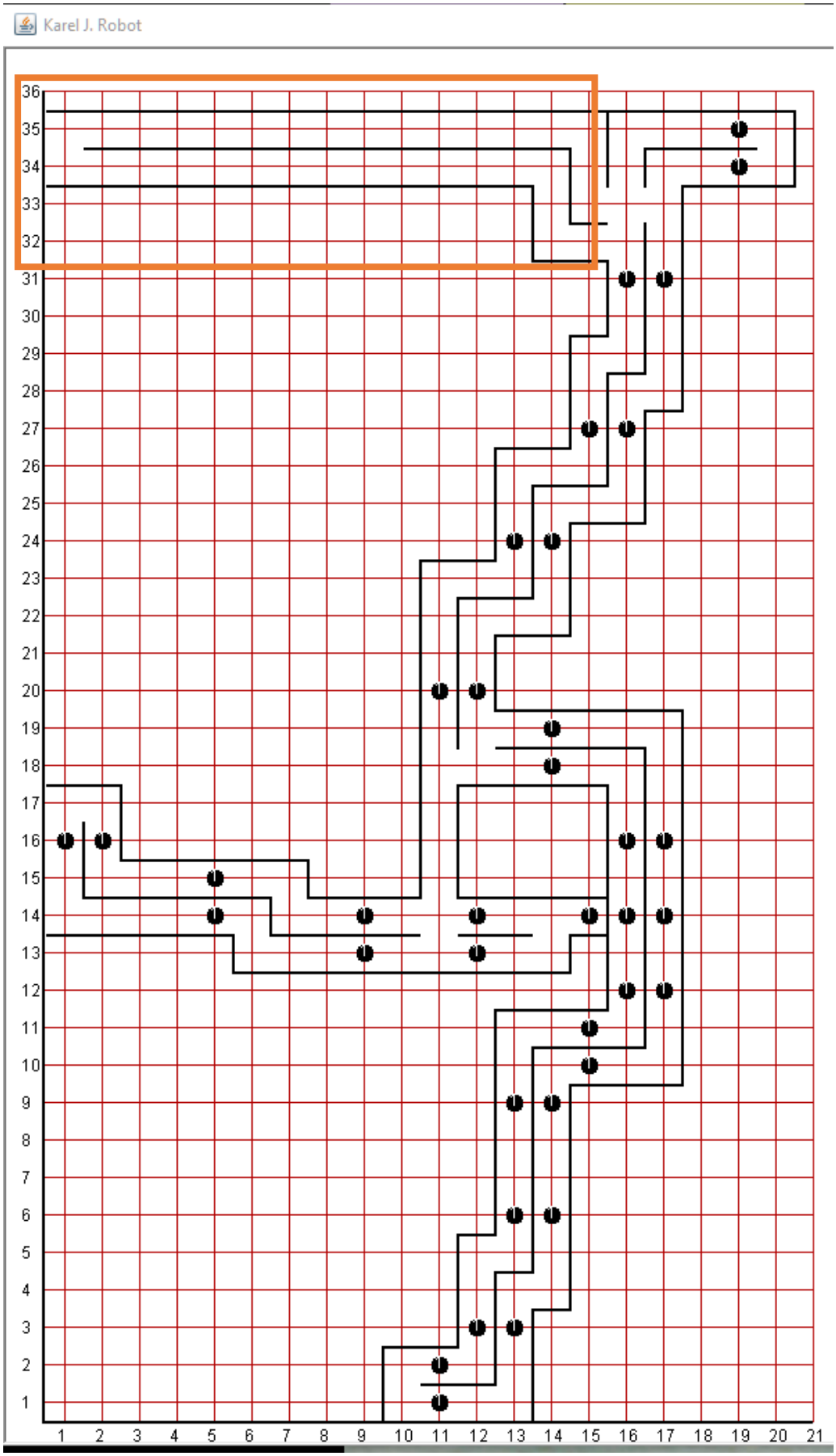
Programe dos robots según el ejemplo en el ejercicio dos para que hagan el mismo recorrido. El segundo robot debe tener color azul para distinguirlos.

Esto completará la primera parte de la práctica. La revisaremos en clase después de Semana santa.

Parte 2 – Simulación del Metro de Medellín

Cómo ustedes bien saben, el Metro tiene dos líneas (A y B), un taller donde les hacen reparaciones durante la noche, unas estaciones, unos trenes y una línea C que permite que los trenes pasen de la línea A a la B para que puedan operar entre San Javier y San Antonio B.

En el siguiente gráfico estará el mundo que representa el metro de Medellín:


Karel J. Robot
36
35
34
33
32
31
30
29
28
27
26
25
24
23
22
21
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21

El rectángulo naranja son los talleres. Donde hay beepers son las estaciones. La más al norte arriba a la derecha es Niquía; la más al sur al centro es La Estrella y la más al occidente en el centro es San Javier. La estación San Antonio A está en la avenida 16 y 17 con calle 14 y la San Antonio B en la avenida 13 entre calles 13 y 14.

Los trenes serán los robots. Estos estarán inicialmente almacenados todos en el taller (representado en naranjado en el gráfico).

Reglas del sistema Metro

1. Los trenes de la línea A serán de color Azul; los de color verde son los de la línea B.
2. No se permite que dos trenes estén en la misma posición del mundo. Si esto se da, es un choque. No habrá simulación de condiciones extremas como pasajeros en la vía.
3. Cuando el metro inicia funcionamiento, salen trenes desde el taller hacia Niquia, San Javier y la Estrella.
4. A las 4:20 a.m. (al mismo tiempo) salen los tres primeros trenes desde las tres estaciones hacia el otro extremo. Es decir, desde Niquía hacia la Estrella y viceversa y desde San Javier a San Antonio. En San Antonio B no hay tren inicial.
5. Solo puede haber un tren por estación en la misma dirección. Tener en cuenta que en San Antonio B solo hay una plataforma, por lo que el tren debe llegar, descargar, girar, cargar pasajeros y salir en sentido San Javier. Desde Cisneros no puede salir el tren hasta que el tren haya salido de San Antonio B.
6. Cuando llega a una estación, cada tren pasa 3 segundos antes de partir a la siguiente estación.
7. No hay infinitos trenes (porque en el taller no hay espacio infinito). En el taller hay espacio hasta 32 trenes. La línea B solo es capaz de manejar 10 trenes en su pico máximo.
8. La conexión entre la línea A y B se llama la línea C. Esa solo tiene un carril y no pueden pasar trenes en sentido contrario al mismo tiempo.
9. No se va a simular el hecho que la cantidad de trenes en circulación varía con el tiempo durante el día normal de operación del Metro.
10. La operación del metro termina a las 11 p.m. A esa hora, sale el último tren desde Niquía y desde la Estrella. Desde San Antonio, sale el último tren a San Javier luego que el último tren en cualquier dirección pasa por San Antonio.
11. Se debe enviar al programa por teclado una señal de que son las 11 de la noche. Luego de dada la señal, La señal de que son las 11 p.m. se da por

teclado al programa. Luego que se de la señal, los trenes que lleguen a las estaciones extremas (Niquía, La Estrella y San Javier).

12. Tener en cuenta los cruces. Para entrar al taller y para salir de la línea B a la línea A de regreso al taller por la línea C.

CONSIDERACIONES GENERALES

1. El desarrollo de la práctica puede ser individual o en equipos de máximo tres personas.

2. El desarrollo completo es en Java.

3. La práctica se entrega subiendo los fuentes en archivos (no comprimidos) y el informe por el buzón recepción de trabajos de Eafit Interactiva (cualquier otro medio no será admitido).

4. Se deben inscribir en Eafit Interactiva los grupos a más tardar el 28 de abril a las 9:00 p.m.

5. El informe final es una presentación que deberá contener una breve descripción de cómo funciona el programa, tablas o gráficos donde se muestre la ejecución de su programa y unas conclusiones que ustedes hagan sobre los datos obtenidos.

6. Cada semana se sacará un espacio de 10 a 20 minutos al inicio de la clase para hablar de la práctica y resolver dudas.

7. Criterios de evaluación (ver Anexo 1)

FECHA DE ENTREGA

Viernes 23 de mayo en clase a través de Eafit Interactiva.

SUSTENTACIÓN

Viernes 23 de mayo en clase. El mecanismo de sustentación es el siguiente:

1. Al inicio de la clase, se realiza la evaluación de 15 minutos.
2. Cada equipo muestra su desarrollo ejecutando en vivo, dejando ver que el programa se ejecuta con los tres modos solicitados.
3. Debe mostrar cómo solucionó cada uno de los retos y cómo lo relacionaron con conceptos vistos en clase.
4. Mostrar los resultados y explicar las conclusiones.

5. Se realizarán preguntas por parte del docente para validar el entendimiento individual de los conceptos aplicados. Esto significa que, aunque el trabajo es en equipo, la nota del trabajo puede ser diferente para cada uno de acuerdo con la calidad de las respuestas.

Nombre de la asignatura: Sistemas Operativos

Competencia a la que aporta la asignatura: Conocer el sistema operativo del computador para un mejor desarrollo, diseño y ejecución de las aplicaciones y aplicar nuevas soluciones.

Resultado de asignatura evaluado: Simulación del Metro de Medellín. Evento evaluativo: Proyecto 3 % del evento evaluativo: 25%

Criterios (que tributen al RA de asignatura)	**Cumple con altos estándares (4.5 -5)**	**Cumple a satisfacción (4 -4.4)**	**Cumple parcialmente (3.5-3.9)**	**Incumple parcialmente (2.5-3.4)**	**Incumple totalmente (0 -2.4)**	**Peso asignado al criterio sobre la calificación.**
Análisis de fundamentación para la creación de la biblioteca. Claridad en el concepto	Entiende completamente la necesidad y plantea varias opciones de solución. Utiliza conceptos adecuadamente para la solución.	Entiende completamente la necesidad y plantea una opción de solución.	Omitió un elemento clave para el entendimiento de la necesidad	Omitió varios elementos para el entendimiento de la necesidad.	Demuestra poco o nulo entendimiento del problema.	40%
Diseño de la solución Solución óptima	El diseño tiene en cuenta los conceptos vistos en clase y argumenta la elección de su solución. La solución elegida es la óptima para el problema.	El diseño tiene en cuenta los conceptos vistos en clase y argumenta la elección de su solución.	Aunque se tuvieron en cuenta conceptos vistos en clase, no hubo argumentación correcta en la elección de la solución.	No se tuvieron en cuenta los conceptos vistos en clase.	Demuestra poco o nulo entendimiento de patrones de paralelismo al momento de explicar la solución.	40%
Funcionalidad Calidad de la solución frente a necesidad planteada	El programa funciona correctamente y se entrega de la forma descrita en el documento.	La solución elegida no es la óptima pero el programa funciona correctamente y se entrega de la forma descrita en el documento.	El programa con los conceptos vistos en clase no funciona correctamente o no se entrega de la forma descrita en el documento.	El programa no tiene en cuenta ningún concepto visto en clase y la solución funciona correcta o parcialmente o no se entrega correctamente.	Se entrega la solución parcialmente o no se entrega ninguna solución o no se entrega correctamente.	20%