ST0254 – Organización de computadores
Paralelismo 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 se desarrollará una simulación de procesamiento paralelo, que involucra la ejecución de múltiples tareas de manera simultánea. El procesamiento en paralelo es crucial en la computación moderna para mejorar el rendimiento y la eficiencia. La simulación requerirá del diseño de un sistema con múltiples procesadores, memoria compartida y mecanismos de comunicación entre procesadores.

PALABRAS CLAVE
Procesamiento paralelo, arquitectura paralela, concurrencia, sincronización, paralelismo, SIMD, MIMD, memoria compartida, comunicación entre procesos, balanceo de carga.

CONTEXTO
Procesamiento en paralelo

Este estilo de procesamiento permite a los computadores ejecutar muchas tareas de manera concurrente, incrementando la velocidad de los cálculos y aumentando el rendimiento. En este proyecto, van a crear una simulación de procesamiento en paralelo usando una arquitectura paralela simplificada. Esta simulación consistirá en múltiples procesadores (Robots) trabajando juntos para resolver tareas computacionales.

Para empezar, se requiere diseñar la arquitectura del sistema paralelo. Esto incluye determinar el número de procesadores, organizar la memoria compartida y establecer canales de comunicación entre los robots. Considere diferentes arquitecturas paralelas como SIMD o MIMD y elegir una que sea acorde con los objetivos del proyecto.

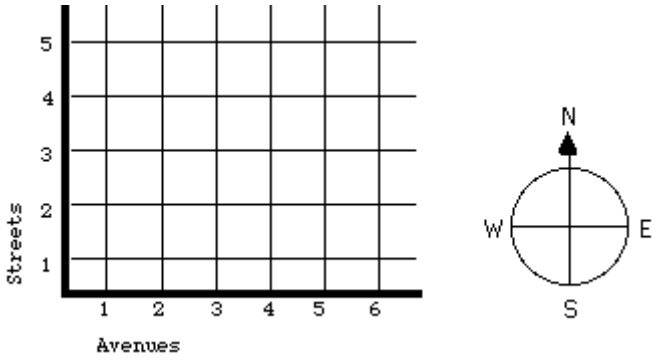
Luego, desarrolle programas en paralelo que ejecuten en esa arquitectura simulada. Esos programas deben ser diseñados para explotar el paralelismo dividendo las tareas en tareas más pequeñas que pueden ser ejecutadas de manera simultánea en diferentes procesadores. Usted debe implementar algún mecanismo de sincronización para asegurar la coordinación adecuada entre procesadores y evitar conflictos de datos.

A través de este proyecto, explorarás conceptos como la concurrencia, sincronización, balanceo de carga y paralelismo. Además, ganarás perspectivas en los desafíos y beneficios del procesamiento en paralelo en aplicaciones del mundo real.

Parte 1. Instalación del robot

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

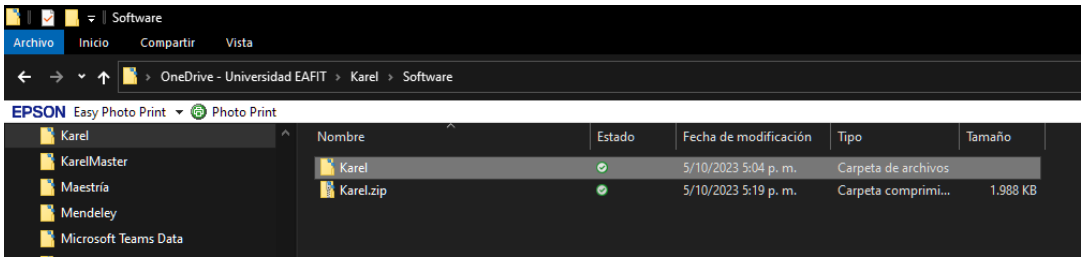
Prerequisitos

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.

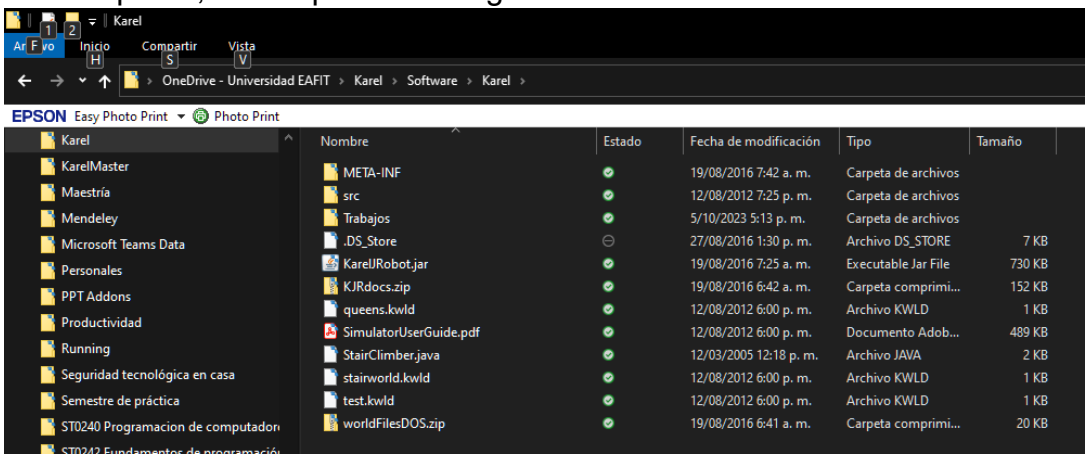
Pasos

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

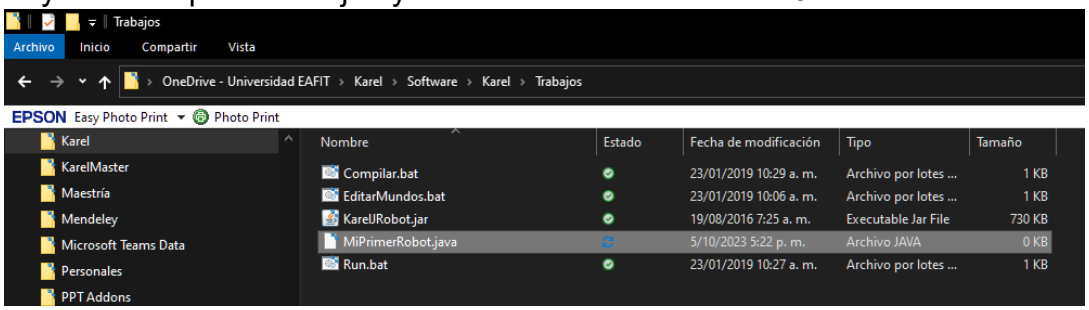
2. Cree en el directorio que usted desee una carpeta y llámela, Karel



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();
        }
}
```

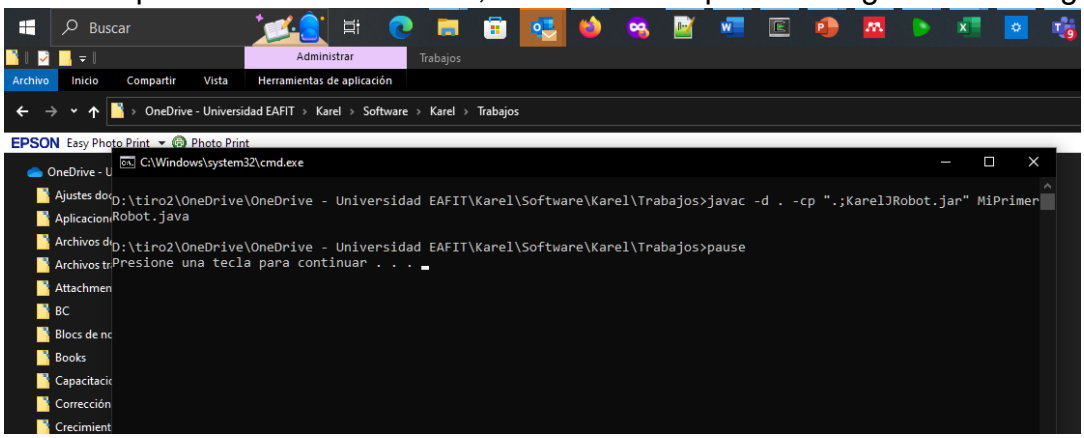
6. Usuarios Windows: Graben el archivo y editen el archivo llamado “Compilar.bat”. El archivo tiene el siguiente contenido:
   javac -d . -cp ".;KarelJRobot.jar" %1
   pause
   exit

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

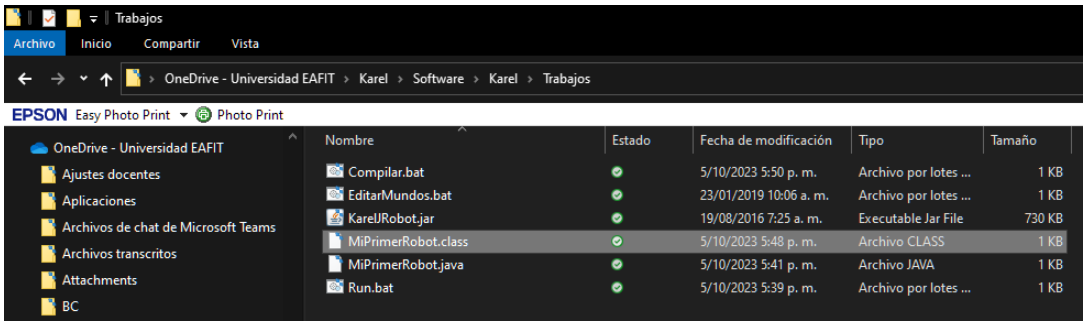
   Usuarios MAC y Linux: Deben ejecutar en la carpeta y por la consola, el comando:
   javac -d . -cp ".:KarelJRobot.jar" MiPrimerRobot.java

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:



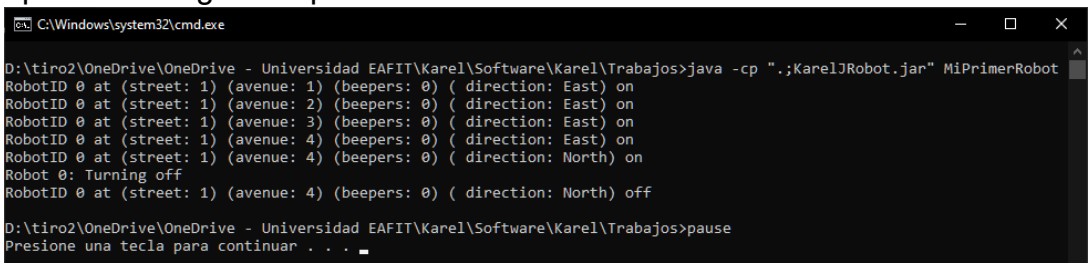
Si esto no sucedió, revise bien los archivos desde el paso 5 para que se aseguren que todo haya quedado como se pide en los pasos anteriores.

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 y Linux: 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:

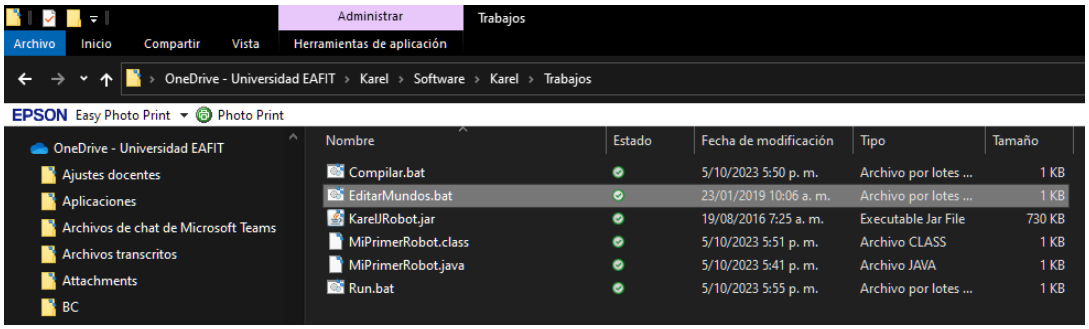


Esto significa que el programa corrió sin problemas. Si no aparecen estos mensajes, primero revise el archivo Run.bat del paso 10 y luego los pasos desde el 5.

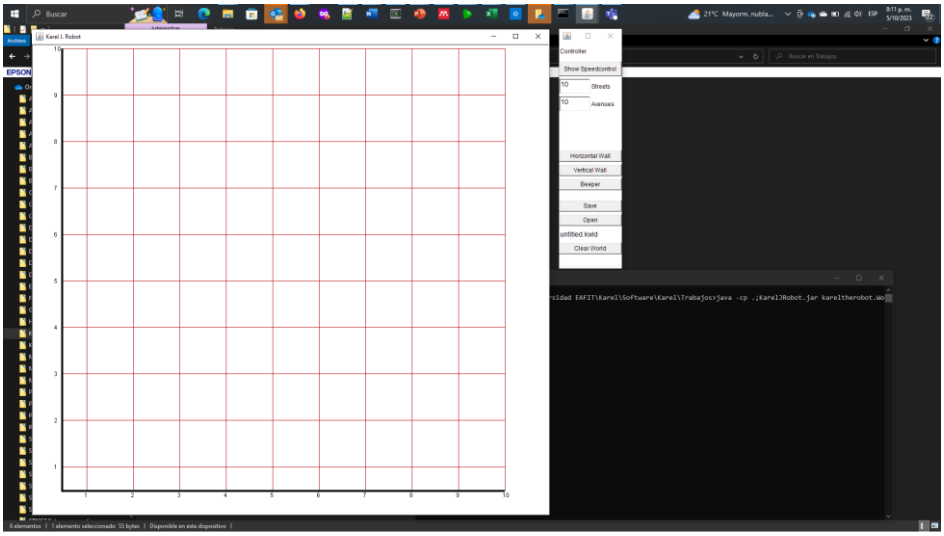
12. Tome un screenshot o una foto de la ejecución de su robot y póngala en la bitácora como evidencia del primer ejercicio.

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

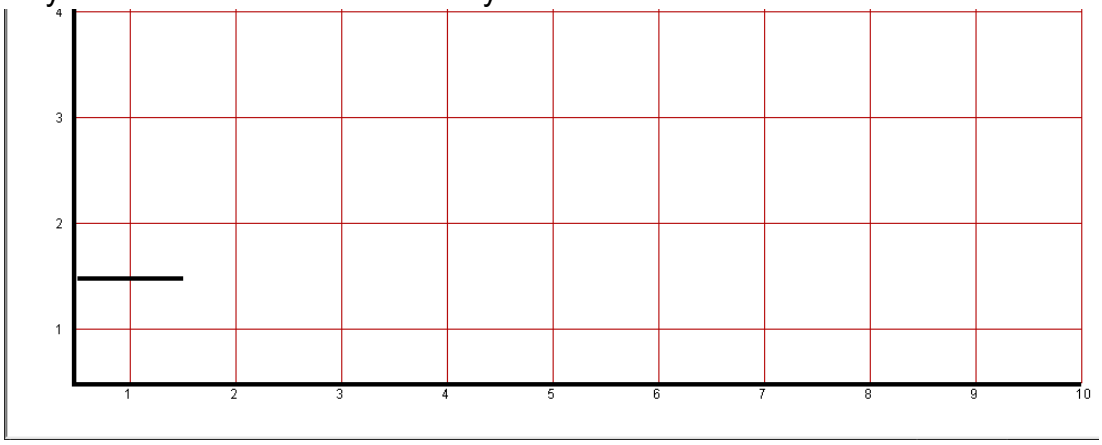


2. Aparecen tres ventanas: primero una negra donde aparecerán mensajes si hay errores y dos blancas donde se muestra el mundo del robot:

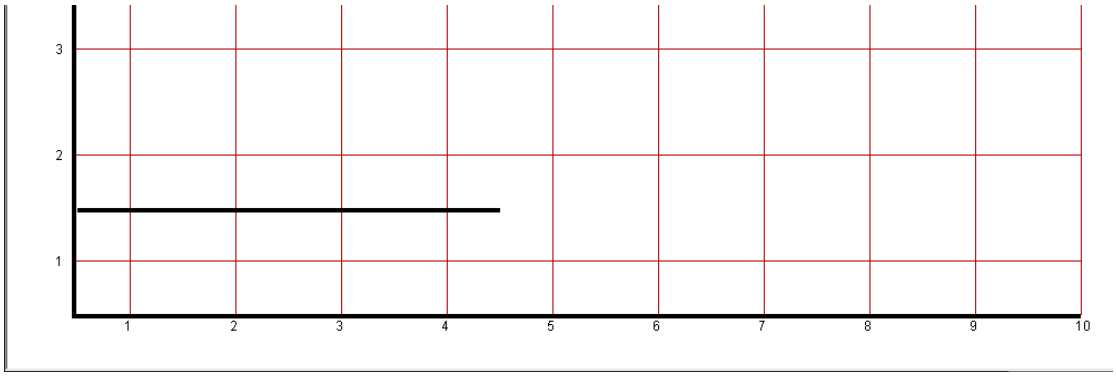


3. En esta interfaz se pueden crear los mundos. Vamos a agregar elementos al mundo. Empezamos con muros horizontales. Damos clic en el botón Horizontal Wall…

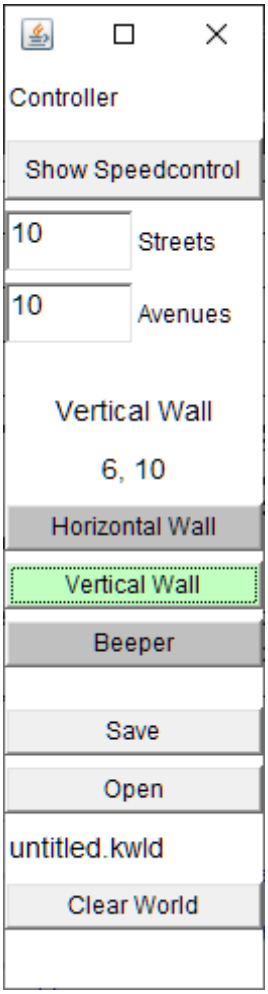
… y damos clic entre la vertical 1 y 2 en la coordenada 1:



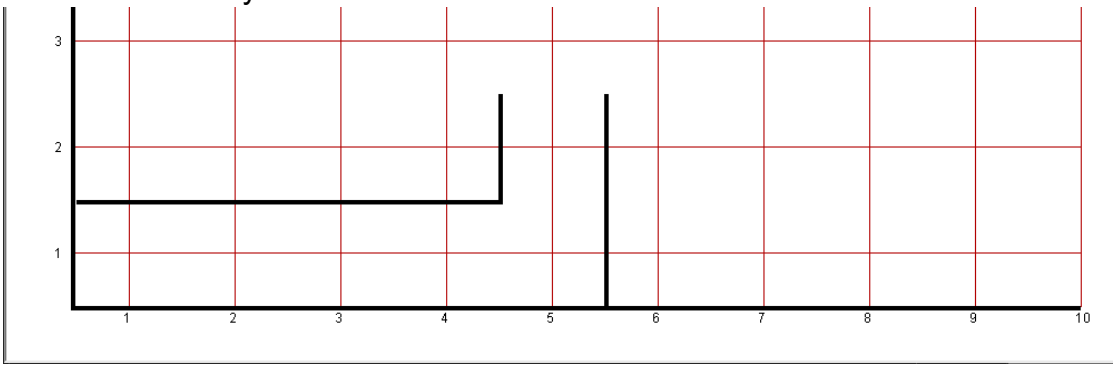
Repetimos lo mismo hasta la avenida 4.



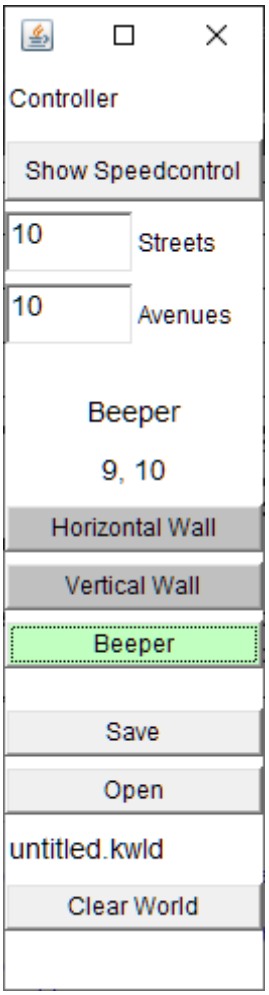
4. Ahora vamos a agregar muros verticales. Damos clic en el botón Vertical Wall…



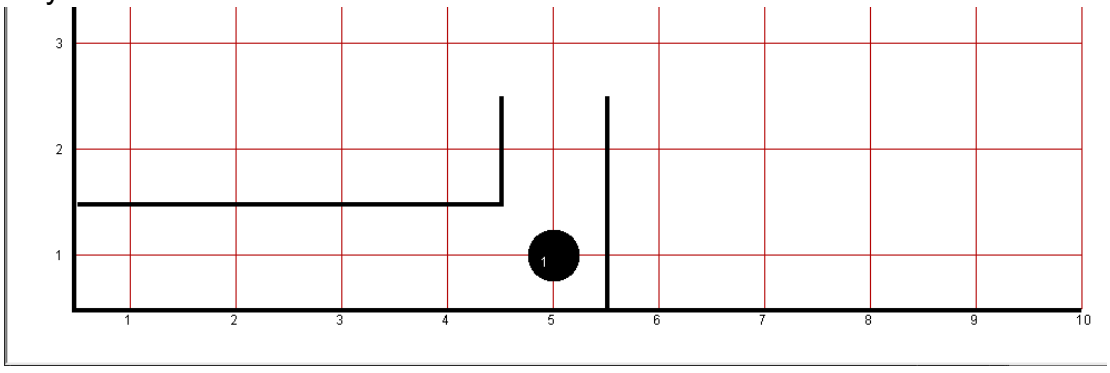
… y agregamos muros entre las avenidas 4 y 5 cruzando la calle 2 y entre las avenidas 5 y 6 cruzando las calles 2 y 1.



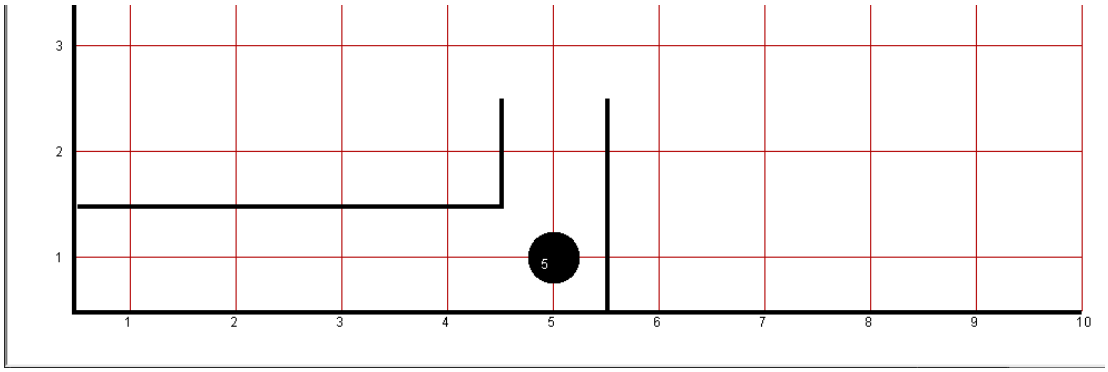
5. Por último, vamos a agregar 5 sirenas en la calle 1 avenida 5. Damos clic en el botón Beeper…



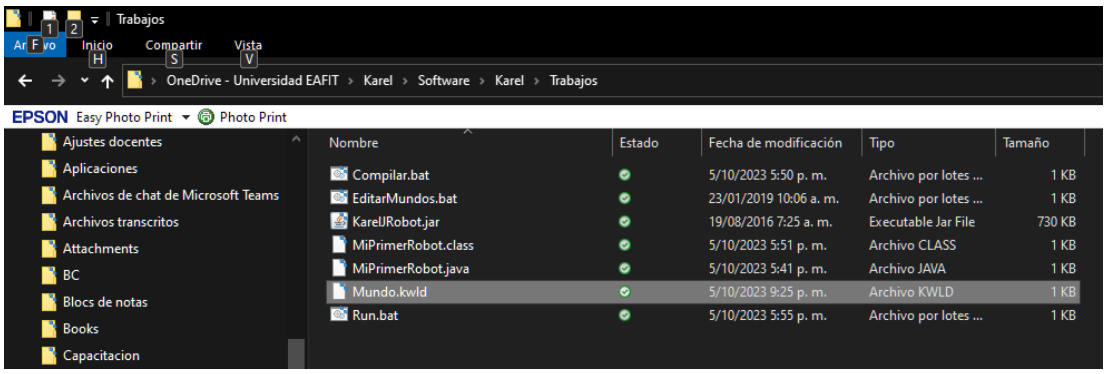
… y damos clic en la avenida 5 calle 1…



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



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

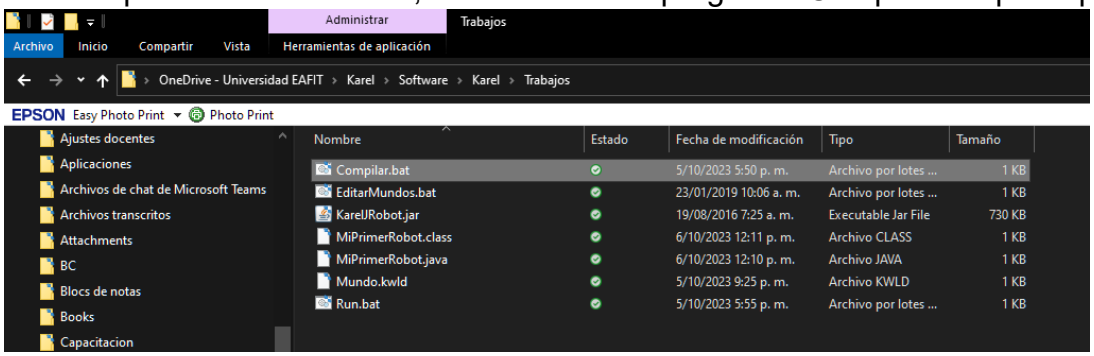


7. Cerramos las ventanas y vamos al código para incorporar el mundo en el código. Debemos modificar el archivo MiPrimerRobot.java de la siguiente manera:

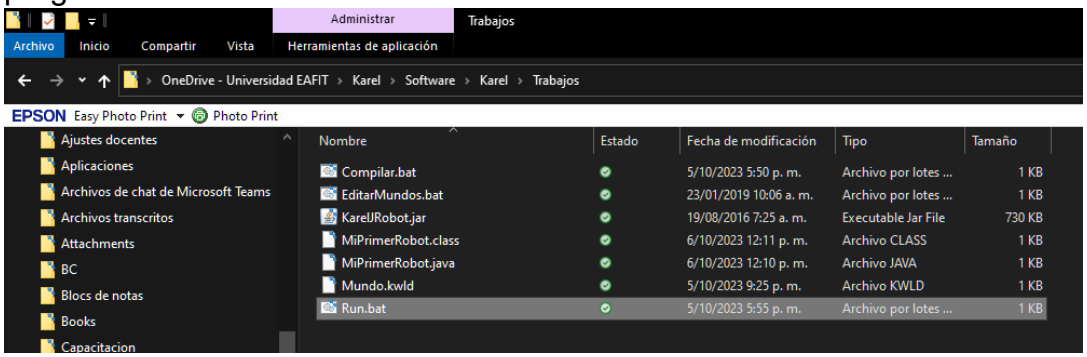
```
import karelTherobot.*;
import java.awt.Color;

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();
        }
}
```

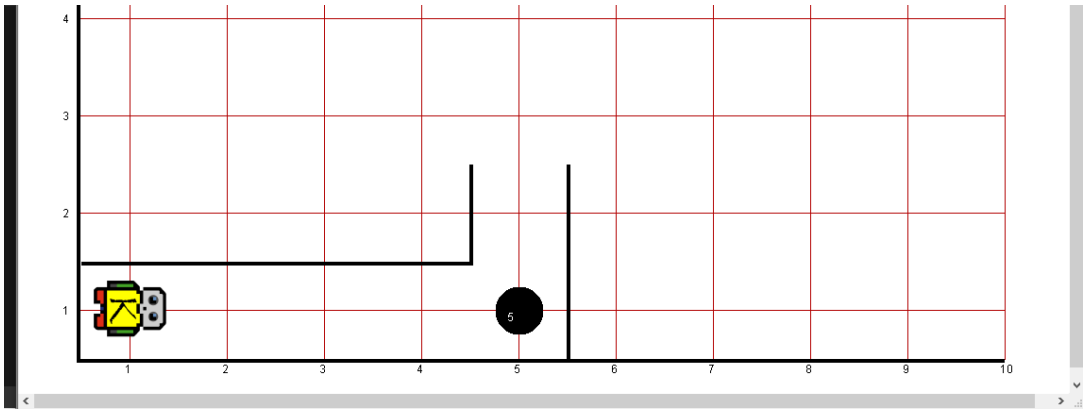
8. En el Explorador de archivos, den dobleclic al programa Compilar.bat para que compile:

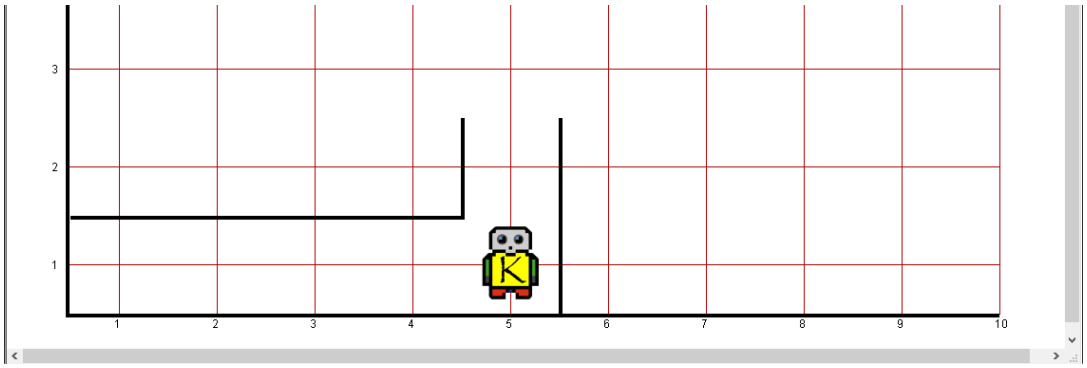


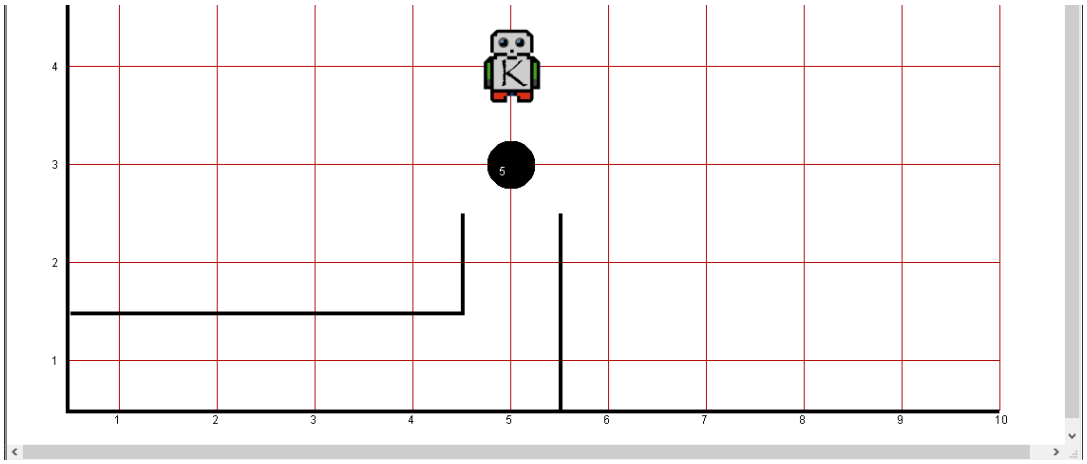
9. Y ejecutamos el programa dando dobleclic en el programa Run.bat para ejecutar el programa:



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 beepers)
        {       super(Street, Avenue, direction, beepers);
                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.

PARTE 2 – Simulación de SIMD y MIMD

Vamos a suponer que tenemos un mundo de 8 calles por 10 avenidas en KarelJRobot y que este mundo tiene regadas diferentes sirenas en diferentes posiciones del mundo entre las avenidas 3 y la 10 y las calles 1 a 10. El objetivo será recoger las sirenas de todo el mundo y dejarlas en un sitio específico del mundo.

Cada robot que se cree puede transportar un número específico de sirenas. Lo llamaremos n, el cual será mayor o igual a 1 y menor o igual a 8 con la condición de que si n > 2, el número n válido elegido debe ser múltiplo de 2 (es decir, los n válidos son 1, 2, 4 y 8).

Los robots empiezan a recorrer el mundo desde la avenida 2 desde la calle que ustedes quieran, teniendo en cuenta que solo puede haber máximo cuatro robots en el mundo. Las sirenas que recoja el robot deben ser transportadas desde su ubicación (cuando llegue a su número n) de la siguiente manera:

1. Si el robot tiene que recoger de a una sirena, las debe llevar a la calle 1 avenida 1.
2. Si el robot tiene que recoger de a dos sirenas, las debe llevar a la calle 2 avenida 1.
3. Si el robot tiene que recoger de a n sirenas (con n <= 8 y n múltiplo de 2), las debe llevar a la calle n avenida 1.

Cuando los robots se den cuenta que no hay más sirenas que transportar, deben ir a la posición donde iniciaron y apagarse.

CONDICIONES GENERALES

1. Sea r el número de robots a crear. r debe ser múltiplo de 2, es decir, no pueden haber tres robots en el mundo y la cantidad de sirenas en el mundo será r * 100. Cuando se crea el robot, recibe de manera aleatoria su cantidad de sirenas máxima a transportar (1, 2, 4 ó 8).

2. Las sirenas se colocarán de manera aleatoria en cada posición del mundo hasta que se asignen las r * 100.

3. Deben encontrar la manera que ningún robot esté con otro en la misma posición donde están recogiendo sirenas. De pasar esto, podrían generar problemas “inesperados” en los funcionamientos que no son del resorte de este curso. Cuando pasen a Sistemas Operativos aprenderán como resolver ese tipo de situaciones.

4. Para la ejecución del programa se debe recibir por parámetros de consola la cantidad de robots r que se tendrán. La cantidad de sirenas que puede transportar el robot (n) será generada de manera aleatoria para cada robot. El argumento a leer de la línea de comandos -r y debe venir acompañado de un número que debe ser 1, 2 ó 4 como se explicó en el punto 1 de condiciones generales.

5. Se puede agregar un argumento opcional que será -e y si está presente, debe asignarse la misma cantidad de sirenas a recoger a cada robot (asignada aleatoriamente pero la misma para todos).

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

7. Cualquier entrega relacionada con la práctica se realizará entregando los fuentes y el informe por el buzón recepción de trabajos de Eafit Interactiva (cualquier otro medio no será admitido).

8. Se debe informar al profesor a más tardar el 2 de mayo a las 6:00 p.m. los integrantes del grupo.

9. El informe final será una presentación que deberá contener una breve descripción de cómo funciona el programa, que dificultades debieron superar para el desarrollo de la práctica y posibles mejoras que consideran, se puede hacer a la misma.

10. La práctica se debe realizar en Java con la plataforma de KarelJRobot. La documentación oficial la pueden consultar en el sitio web https://csis.pace.edu/~bergin/KarelJava2ed/KJRdocs/index.html

11. Criterios de evaluación (ver Anexo 1)

FECHA DE ENTREGA
Domingo 26 de mayo, hasta las 11:59 p.m. a través de Eafit Interactiva

SUSTENTACIÓN
Semana del 27 al 31 de mayo.

Nombre de la asignatura: Organización de computadores

Competencia a la que aporta la asignatura: Conocer el computador para un mejor desarrollo, diseño y ejecución de las aplicaciones

Resultado de asignatura evaluado: Construcción de una aplicación de mezclador de sonido.

Evento evaluativo: Práctica final Porcentaje del evento evaluativo: 30%

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 solución: Paralelismo 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 patrones de paralelismo y argumenta la elección de su solución. La solución elegida es la óptima para el problema.	El diseño tiene en cuenta patrones de paralelismo y argumenta la elección de su solución.	Aunque se tuvieron en cuenta patrones de paralelismo, no hubo argumentación correcta en la elección de la solución.	No se tuvieron en cuenta patrones de paralelismo.	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.	La solución elegida no es la óptima pero el programa funciona correctamente	El programa con patrones de paralelismo no funciona correctamente.	El programa no tiene en cuenta ningún patrón de paralelismo y la solución funciona correcta o parcialmente.	Se entrega la solución parcialmente o no se entrega ninguna solución.	20%