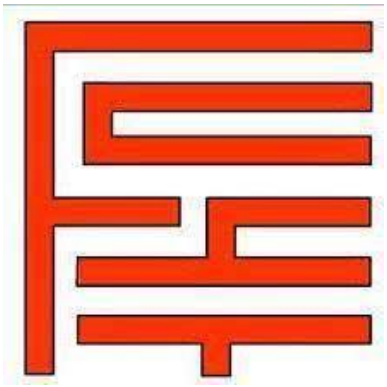


Actividad 2

Segundo parcial

Resolución de ejercicios de Pro Model en java

Nombre: Abel Alejandro Pacheco Quispe

Materia: Taller de Simulación de Sistemas

Docente: Ayoroa Cardozo Jose Richard

Contenido

Ejercicio 10:

Descripción:

A las cajas de la cafetería de una universidad llegan 800 estudiantes por día. La tasa de entrada durante el día es variable y se comporta de la siguiente forma:

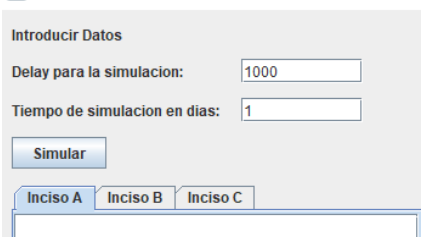
De:	A:	Porcentaje
6:00	7:00	5
7:00	10:00	20
10:00	12:00	10
12:00	14:00	50
14:00	18:00	5
18:00	19:00	10

Existen 4 empleados para cobrar y una fila común donde los estudiantes pueden hacer fila antes de ser atendidos. El tiempo de atención es de 45 ± 20 segundos. Simule el proceso anterior durante 30 días. Incluya en la simulación un gráfico dinámico en el que se observe la cantidad de estudiantes en la fila a través del tiempo, y determine:

a. La utilización de los empleados.
b. El tiempo promedio de permanencia de los estudiantes en la fila.
c. Con base en los resultados anteriores ¿agregaría más empleados?
d. Con base en la gráfica ¿qué recomendaciones haría?

De:	A:	Tiempo (horas)	Tiempo Acumulado (horas)	Porcentaje
6:00	7:00	1	1	5
7:00	10:00	3	4	20
10:00	12:00	2	6	10
12:00	14:00	2	8	50
14:00	18:00	4	12	5
18:00	19:00	1	13	10
19:00	06:00	11	24	0

Código e interfaces:



Para la interfaz se pide insertar los datos de Delay para la velocidad de ejecución y el tiempo de simulación en días.

```
private void simularEjercicioA(java.awt.event.ActionEvent evt) {
    try {
        int duracion = Integer.parseInt(tiempoSimulacion.getText());
        int delay = Integer.parseInt(numeroIteraciones.getText());
        Ejecucion1 hilo = new Ejecucion1(duracion, delay, textAreaA, textAreaB);
        hilo.start(); // Ejecuta el hilo con el valor de duración recibido
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Por favor, introduce valores numéricos v
    }
}
```

En la parte de inicialización del programa se recogen los datos de la interfaz como el tiempo de delay y el tiempo o cantidad de objetos a simular, luego se crea una instancia para ejecutar el hilo para poder ver la simulación en simultaneo a la ejecución.

```
import org.apache.commons.math3.distribution.UniformRealDistribution;
import org.apache.commons.math3.distribution.PoissonDistribution;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Ejecucion1 extends Thread {
    private int duracion;
    private SimulacionPanel simulacionPanel;
    private int numeroEstudiantesEnFila = 0; // Variable para el conteo de estudiantes en f
    private int contadorFila = 0; // Contador para estudiantes que pasaron por la fila
    private int[] contadorEmpleado = new int[4]; // Contadores para los empleados
    private int delay = 0;
    private JTextArea textAreaA;
    private JTextArea textAreaB;

    public Ejecucion1(int duracion, int delay, JTextArea textAreaA, JTextArea textAreaB) {
        this.duracion = duracion;
        this.simulacionPanel = new SimulacionPanel();
        this.delay = delay;
        this.textAreaA = textAreaA;
        this.textAreaB = textAreaB;
        initGrafico();
    }
```

En la parate de la ejecución primero esportamos las distribuciones que utilizaremos para realizar la simulación, luego exportamos las herramientas necesarias para las graficas y finalmente inicializamos variables importantes como el numero de estudiantes en fila los contadores y otros.

En la creación de la ejecución lo iniciamos pasando los datos de duración, delay, y las áreas de texto donde se introducirán las respuestas y también iniciamos el grafico.

```
private void initGrafico() {
    JFrame frame = new JFrame("Simulación Dinámica de Atención a Estudiantes");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(new BorderLayout());
    frame.add(simulacionPanel, BorderLayout.CENTER);
    frame.setSize(800, 400);
    frame.setVisible(true);
}

@Override
public void run() {
    List<Integer> tiempoAtencion = new ArrayList<>();
    List<Integer> tiempoSis = new ArrayList<>();
    double[] porcentaje = {0.05, 0.20, 0.10, 0.50, 0.05, 0.10, 0.0};
    int[] duracionInt = {60 * 60, 180 * 60, 120 * 60, 120 * 60, 240 * 60, 60 * 60, 660 * 60};
    Locacion empleado1 = new Locacion("Empleado 1", 1);
    Locacion empleado2 = new Locacion("Empleado 2", 1);
    Locacion empleado3 = new Locacion("Empleado 3", 1);
    Locacion empleado4 = new Locacion("Empleado 4", 1);
    for(int dias = 1; dias <= duracion; dias++){
        PoissonDistribution distribucionPoisson = new PoissonDistribution(700);
        int estudiantesLlegadosTotales = distribucionPoisson.sample();
        int tiempoTotal = 1440 * 60; // 1 día en minutos
        int int1 = (int) (porcentaje[0] * estudiantesLlegadosTotales);
        int int2 = (int) (porcentaje[1] * estudiantesLlegadosTotales);
        int int3 = (int) (porcentaje[2] * estudiantesLlegadosTotales);
        int int4 = (int) (porcentaje[3] * estudiantesLlegadosTotales);
        int int5 = (int) (porcentaje[4] * estudiantesLlegadosTotales);
        int int6 = (int) (porcentaje[5] * estudiantesLlegadosTotales);
        int int7 = (int) (porcentaje[6] * estudiantesLlegadosTotales);
        int suma = int1 + int2 + int3 + int4 + int5 + int6 + int7;
        Locacion fila = new Locacion("Fila Principal", suma);
```

Luego le damos el método para iniciar el grafico dinámico, y finalmente ejecutamos el run donde se realizarán todas las operaciones necesarias, primero iniciamos variables, listas, y pasamos los datos de probabilidad de las llegadas de estudiantes y calculamos cuantos llegan y en que periodos.

```
int tiempoSalida1 = 0;
int tiempoSalida2 = 0;
int tiempoSalida3 = 0;
int tiempoSalida4 = 0;
int id = 0;

for (int hora = 1; hora <= tiempoTotal; hora++) {
    try {
        Thread.sleep(delay);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    simulacionPanel.setFilaCount(numeroEstudiantesEnFila);
    simulacionPanel.setEmpleadoCounts(
        empleado1.getCantidadEntidades(),
        empleado2.getCantidadEntidades(),
        empleado3.getCantidadEntidades(),
        empleado4.getCantidadEntidades()
    );
    simulacionPanel.setContadorFila(contadorFila); // Actualiza e
    simulacionPanel.setContadorEmpleado(contadorEmpleado); // Act
    simulacionPanel.repaint();
```

Seguimos inicializando variables para almacenar datos, luego iniciamos un ciclo for para que funcione como el reloj interno, luego lo primero tenemos es que le pasamos el delay para la velocidad de la simulación y luego realizamos las actualizaciones de los gráficos.

```
switch (hora) {
    case 1 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[0] * estudiantesLlegadosTotales), duracionInt[0], 0, fila, textAreaA);
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 61 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[1] * estudiantesLlegadosTotales), duracionInt[1], 60 * 60, fila, textAr
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 241 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[2] * estudiantesLlegadosTotales), duracionInt[2], 240 * 60, fila, textA
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 361 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[3] * estudiantesLlegadosTotales), duracionInt[3], 360 * 60, fila, textA
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 480 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[4] * estudiantesLlegadosTotales), duracionInt[4], 480 * 60, fila, textA
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 721 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[5] * estudiantesLlegadosTotales), duracionInt[5], 720 * 60, fila, textA
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
    }
    case 781 -> {
        int llegadas = simularLlegadaEstudiantes((int) (porcentaje[6] * estudiantesLlegadosTotales), duracionInt[6], 780 * 60, fila, textA
        numeroEstudiantesEnFila += llegadas;
        contadorFila = contadorFila + llegadas;
```

Luego pasamos a un switch que si es el tiempo de intervalo hará llegar a los nuevos clientes y los almacenara en un variable llegadas, para la llegada necesitamos los datos la cantidad de estudiantes, duración, el tiempo de inicio del intervalo, la fila a la cual llegaran y el text área donde imprimir los resultados.

```
if (id < estudiantesLlegadosTotales) {
    if (empleado1.getCantidadEntidades() == 0 && hora >= tiempoSalida1) {
        tiempoSalida1 = simularAtencionEmpleado(fila.getEntidadPorIndice(id), hora, empleado1, id + 1, textAreaA);
        tiempoAtencion.add(tiempoSalida1);
        tiempoSis.add(tiempoSalida1);
        numeroEstudiantesEnFila--;
        contadorEmpleado[0]++; // Incrementa el contador del empleado 1
        id++;
    } else if (empleado2.getCantidadEntidades() == 0 && hora >= tiempoSalida2) {
        tiempoSalida2 = simularAtencionEmpleado(fila.getEntidadPorIndice(id), hora, empleado2, id + 1, textAreaA);
        tiempoAtencion.add(tiempoSalida2);
        numeroEstudiantesEnFila--;
        contadorEmpleado[1]++; // Incrementa el contador del empleado 2
        id++;
    } else if (empleado3.getCantidadEntidades() == 0 && hora >= tiempoSalida3) {
        tiempoSalida3 = simularAtencionEmpleado(fila.getEntidadPorIndice(id), hora, empleado3, id + 1, textAreaA);
        tiempoAtencion.add(tiempoSalida3);
        numeroEstudiantesEnFila--;
        contadorEmpleado[2]++; // Incrementa el contador del empleado 3
        id++;
    } else if (empleado4.getCantidadEntidades() == 0 && hora >= tiempoSalida4) {
        tiempoSalida4 = simularAtencionEmpleado(fila.getEntidadPorIndice(id), hora, empleado4, id + 1, textAreaA);
        tiempoAtencion.add(tiempoSalida3);
        numeroEstudiantesEnFila--;
        contadorEmpleado[3]++; // Incrementa el contador del empleado 4
        id++;
    }
```

Luego procedemos a verificar si el estudiante esta todavía en el rango y procedemos a simular el tiempo de atención del empleado y aumentando los contadores necesarios para el cálculo de datos correspondiente.

```
    if (hora == tiempoSalida1) {
        estudianteSale(empleado1, empleado1.getEntidadPorIndice(0), textAreaA);
    }
    if (hora == tiempoSalida2) {
        estudianteSale(empleado2, empleado2.getEntidadPorIndice(0), textAreaA);
    }
    if (hora == tiempoSalida3) {
        estudianteSale(empleado3, empleado3.getEntidadPorIndice(0), textAreaA);
    }
    if (hora == tiempoSalida4) {
        estudianteSale(empleado4, empleado4.getEntidadPorIndice(0), textAreaA);
    }
}
textAreaB.append("Locaciones: \n");
textAreaB.append("Clientes que llegaron a la fila: " + suma + "\n");
textAreaB.append("Clientes que atendio el empleado 1: " + contadorEmpleado[0] + "\n");
textAreaB.append("Clientes que atendio el empleado 2: " + contadorEmpleado[1] + "\n");
textAreaB.append("Clientes que atendio el empleado 3: " + contadorEmpleado[2] + "\n");
textAreaB.append("Clientes que atendio el empleado 4: " + contadorEmpleado[3] + "\n");
textAreaB.append("Capacidad de Fila: 99999.999 \n");
textAreaB.append("Capacidad de empleado1: 1 \n");
textAreaB.append("Capacidad de empleado2: 1 \n");
textAreaB.append("Capacidad de empleado3: 1 \n");
textAreaB.append("Capacidad de empleado4: 1 \n");
```

Luego simulamos la salida de los estudiantes y como ultimo procedemos a imprimir los datos necesarios de la simulación.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);

    // Cambia el color de la fila a celeste si hay estudiantes en la fila
    if (filaCount > 0) {
        g.setColor(Color.CYAN); // Celeste
    } else {
        g.setColor(Color.YELLOW);
    }
    g.fillRect(50, 50, 100, 50);
    g.setColor(Color.BLACK);
    g.drawString("Fila: " + filaCount, 60, 80);
    g.drawString("Total en fila: " + contadorFila, 60, 40); // Muestra el contador encima de la

    int xOffset = 200;
    for (int i = 0; i < 4; i++) {
        // Cambia el color del empleado a gris si está atendiendo
        if (empleadoCounts[i] == 1) {
            g.setColor(Color.GRAY); // Gris cuando está atendiendo
        } else {
            g.setColor(Color.GREEN); // Verde si no está atendiendo
        }
        g.fillRect(xOffset, 50, 90, 50);
        g.setColor(Color.BLACK);
        g.drawString("Empleado " + (i + 1) + ": " + empleadoCounts[i], xOffset + 5, 80);
        g.drawString("Atendidos: " + contadorEmpleado[i], xOffset + 5, 40); // Muestra el conta
        xOffset += 150;
    }
```

Luego creamos los componentes del grafico necesarios para la simulación y darles tamaño y colores para distinguirlos.

```
public int simularLlegadaEstudiantes(int numeroEntidades, int tiempoTotal, int minutoInicio, Locacion fila, JTextArea textArea) {
    Random random = new Random();
    int tiempoAnterior = minutoInicio;
    int nuevosEstudiantes = 0;

    for (int i = 1; i <= numeroEntidades; i++) {
        int tiempoLlegada = tiempoAnterior + random.nextInt(30) + 1;
        if (tiempoLlegada > tiempoTotal + minutoInicio) break;

        int estudianteId = fila.getCantidadEntidades() + 1;
        Entidad entidad = new Entidad("Entidad " + estudianteId, estudianteId, tiempoLlegada);

        if (fila.agregarEntidad(entidad)) {
            nuevosEstudiantes++;
            textArea.append("Estudiante " + estudianteId + " ha llegado en el segundo " + tiempoLlegada + "\n");
        }
        tiempoAnterior = tiempoLlegada;
    }
    return nuevosEstudiantes;
}
```

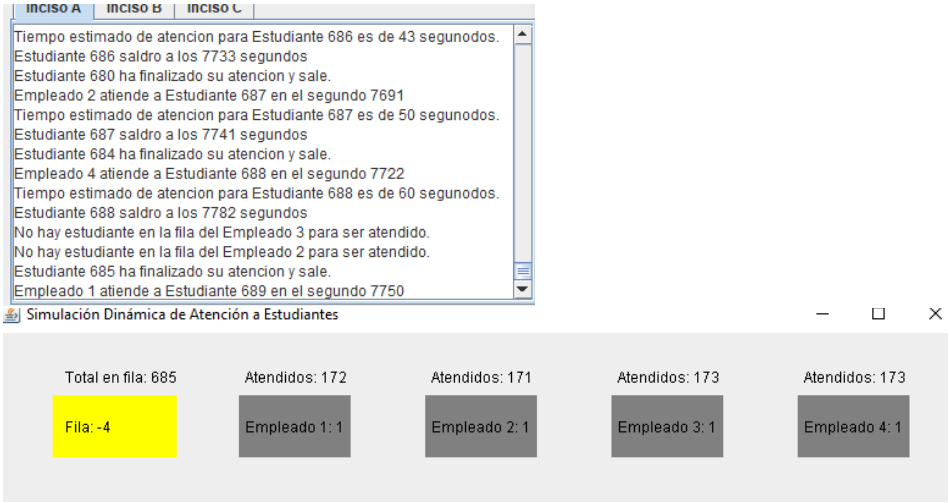
Luego vemos los métodos creados para simular la llegada de los estudiantes y por cada estudiante que llega se crea una entidad estudiante y se la añade a la locación fila.

```java
public int simularAtencionEmpleado(Entidad estudiante, int tiempoActual, Locacion empleado, int estudianteId, JTextArea textArea) {
    empleado.agregarEntidad(estudiante);
    UniformRealDistribution distribucionUniforme = new UniformRealDistribution(25, 65);
    int tiempoAtencion = (int) distribucionUniforme.sample();
    int tiempoSalida = tiempoActual + tiempoAtencion;

    textArea.append(empleado.getNombre() + " atiende a Estudiante " + estudianteId + " en el segundo " + tiempoActual+ "\n");
    textArea.append("Tiempo estimado de atencion para Estudiante " + estudianteId + " es de " + tiempoAtencion + " segunodos."+ "\n");
    textArea.append("Estudiante " + estudianteId + " saldro a los " + tiempoSalida + " segundos"+ "\n");

    return tiempoSalida;
}

public void estudianteSale(Locacion empleado, Entidad estudiante, JTextArea textArea) {
    if (estudiante != null) {
        textArea.append("Estudiante " + estudiante.getId() + " ha finalizado su atencion y sale."+ "\n");
        empleado.eliminarEntidad(estudiante);
    } else {
        textArea.append("No hay estudiante en la fila del " + empleado.getNombre() + " para ser atendido."+ "\n");
    }
}

public static double calcularMedia(List<Integer> numeros) {
    if (numeros == null || numeros.isEmpty()) {
        return 0;
    }
    int suma = 0;
    for (int numero : numeros) {
        suma += numero;
    }
    return (double) suma / numeros.size();
}
```

Luego tenemos los métodos para simular la atención que utiliza una distribución uniforme y se calculan los tiempos de atención y se imprimen dichos datos y retorna el tiempo de salida, luego se simula la salida de estudiante donde se elimina al estudiante de la locación simulando que salió.

El ultimo método tiene como objetivo calcular la media de una lista que se mande como parámetro.

Ejecución:



Respuestas y comparaciones:

a) La utilización de los empleados.

enido Máximo	Contenido Actual	% Utilización
9,00	0,00	0,00
1,00	0,00	14,88
1,00	0,00	6,78
1,00	0,00	3,13
1,00	0,00	1,45

Capacidad de empleado4: 1
Porcentaje de utilizacion de Empleado 1: 24.680851063829788%
Porcentaje de utilizacion de Empleado 2: 25.53191489361702%
Porcentaje de utilizacion de Empleado 3: 24.822695035460992%
Porcentaje de utilizacion de Empleado 4: 24.9645390070922%

El empleado 1 trabajo 24.68% el 2 trabajo 25.53% el 3 trabajo 24.82% y el 4 trabajo 24.96% se puede ver que existe una variación debido a que las distribuciones generan números de forma aleatoria.

b) El tiempo promedio de permanencia de los estudiantes en la fila.

Porcentaje de utilización de Empleado 4: 24.964539007092 %
Entidades:
Estudianrtes: 705
Tiempo promedio en operacion pieza1: 4014.273758865248

Locación Resumen				
Nombre	Tiempo Programado (Hr)	Capacidad	Total Entradas	Tiempo Por entrada Promedio (Min)
FILA CAJAS	709,01	999.999,00	20.616,00	0,51
EMPLEADOS.1	709,01	1,00	11.702,00	0,54

El tiempo por entrada promedio es de 4014 segundos más o menos 66 minutos.

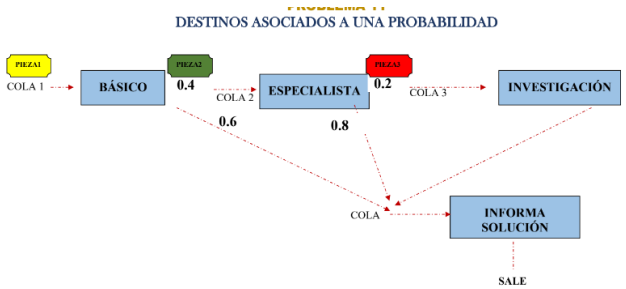
c) Con base en los resultados anteriores ¿agregaría más empleados?
Viendo que solo el empleado 1 trabajo mas y que el empleado 4 casi ni trabajo no conviene agregar mas empleados.

d) Con base en la gráfica ¿qué recomendaciones haría?
Viendo que el empleado 4 y 3 casi no trabajaron seria despedir a los 2 empleados para ahorrar costos.

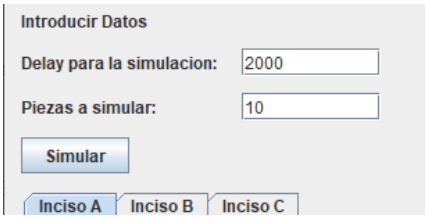
Ejercicio 11:

Descripción:

Tenemos 3 colas donde llegan las piezas donde la pieza 2 tiene un 40% de probabilidad de que la pieza 2 pase a especialista y 60 de que pase a informa solución, de especialista sale la pieza 3 y tiene un 20% de probabilidad de ir a investigación y un 80 de ir a informa solución y en investigación pasa directo a informa solución.



Código e interfaz:



Para la interfaz este igualmente recibe un numero en milisegundos para el delay y el número de piezas que se desea simular.

```
private void simularEjercicioA(java.awt.event.ActionEvent evt) {
    try {
        int duracion = Integer.parseInt(tiempoSimulacion.getText());
        int delay = Integer.parseInt(numeroIteraciones.getText());
        if (textAreaA == null) {
            throw new IllegalArgumentException("El JTextArea no puede ser null");
        }
        Ejecucion2 hilo = new Ejecucion2(duracion, delay, textAreaA, textAreaB);
        hilo.start();
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Por favor, introduce valores numéricos vál
    }
}
```

En la parte de inicialización del programa se recogen los datos de la interfaz como el tiempo de delay y el tiempo o cantidad de objetos a simular, luego se crea una instancia para ejecutar el hilo para poder ver la simulación en simultaneo a la ejecución.

```
package com.mycompany.promodel;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import org.apache.commons.math3.distribution.ExponentialDistribution;
import org.apache.commons.math3.distribution.UniformRealDistribution;

public class Ejecucion2 extends Thread {
    private int piezas;
    private int delay;
    private Random random = new Random();
    private SimulacionPanel simPanel;
    private JTextArea textAreaA;
    private JTextArea textAreaB;

    public Ejecucion2(int piezas, int delay, JTextArea textAreaA, JTextArea textAreaB) {
        this.piezas = piezas;
        this.delay = delay;
        this.textAreaA = textAreaA;
        this.textAreaB = textAreaB;
        initGrafico();
    }

    private void initGrafico() {
        JFrame frame = new JFrame("Simulación de Piezas");
```

El método de ejecución de la simulación primero exportamos las herramientas necesarias para realizar la simulación y creamos el constructor que recibe el número de piezas el delay y las áreas de texto donde se mostraran los resultados y donde se inicializa el grafico.

```
@Override
public void run() {
    int tiempo = 0;
    int aux = 0;
    List<Integer> tiempoPro1 = new ArrayList<>();
    List<Integer> tiempoPro2 = new ArrayList<>();
    List<Integer> tiempoPro3 = new ArrayList<>();
    List<Integer> tiempoSis1 = new ArrayList<>();
    List<Integer> tiempoSis2 = new ArrayList<>();
    List<Integer> tiempoSis3 = new ArrayList<>();
    List<Entidad> cola1 = new ArrayList<>();
    List<Entidad> cola2 = new ArrayList<>();
    List<Entidad> cola3 = new ArrayList<>();
    List<Entidad> cola4 = new ArrayList<>();
    List<Entidad> basico = new ArrayList<>();
    List<Entidad> especialista = new ArrayList<>();
    List<Entidad> investigacion = new ArrayList<>();
    List<Entidad> informacion = new ArrayList<>();
    int contador1 = 0;
    int contador2 = 0;
    int contador3 = 0;
    int contador4 = 0;
```

En el hilo primero iniciamos los datos y variables necesarias para realizar la simulación como las listas y otros.

```
(int pieza = 1; pieza <= piezas; pieza++) {
 textAreaA.append("Pieza " + pieza + "\n");
 ExponentialDistribution expDistribution = new ExponentialDistribution(20);
 int tiempoLlegada = (int) expDistribution.sample();
 tiempo += tiempoLlegada;

 Entidad entidad = new Entidad("pieza1", pieza, tiempoLlegada);
 cola1.add(entidad);
 textAreaA.append("Intervalo de llegada: " + tiempoLlegada + " minutos" + "\n");
 textAreaA.append("Llego en minuto: " + tiempo + " pieza id: " + entidad.getId() + " : " + entidad.getNombre() + "\n");

 // Actualizar la gráfica para el ingreso a cola1
 actualizarGrafico(cola1, cola2, cola3, cola4, contador1, contador2, contador3, contador4);
 aux = proceso1("basico", tiempo, basico, entidad, textAreaA);
 tiempo += aux;
 tiempoPro1.add(aux);
 textAreaA.append("Sale en: " + tiempo + " pieza id: " + entidad.getId() + " : " + entidad.getNombre()+ "\n");

 double numeroAleatorio = random.nextDouble();
 contador1++;
```

Primero con la distribución poison para realizar la simulación de las llegadas y vamos aumentando el tiempo de las llegadas para ir controlando el tiempo, luego creamos la entidad e imprimimos algunos datos y mandamos a actualizar el grafico y a procesar el tiempo de las piezas en sus locaciones.

```
if (numeroAleatorio < 0.4) {
    contador2++;
    entidad.cambiarNombre("pieza2");
    cola1.remove(entidad);
    cola2.add(entidad);

    // Actualizar la gráfica para el ingreso a cola2
    actualizarGrafico(cola1, cola2, cola3, cola4, contador1, contador2, contador3, contador4);
    aux = procesol("especialista", tiempo, especialista, entidad, textAreaA);
    tiempo += aux;
    tiempoPro2.add(aux);
    textAreaA.append("Llego en minuto: " + tiempo + " pieza id: " + entidad.getId() + " : " + entidad.getNombre()+ "\n");

    double numeroAleatorio1 = random.nextDouble();
    if (numeroAleatorio1 < 0.2) {
        entidad.cambiarNombre("pieza3");
        contador3++;
        cola2.remove(entidad);
        cola3.add(entidad);

        // Actualizar la gráfica para el ingreso a cola3
        actualizarGrafico(cola1, cola2, cola3, cola4, contador1, contador2, contador3, contador4);
        aux = procesol("investigacion", tiempo, investigacion, entidad, textAreaA);
        tiempo += aux;
        tiempoPro3.add(aux);
        textAreaA.append("Llego en minuto: " + tiempo + " pieza id: " + entidad.getId() + " : " + entidad.getNombre()+ "\n");
        cola3.remove(entidad);

        contador4++;
        cola4.add(entidad);
        cola4.remove(entidad);
        aux = procesol("Informacion", tiempo, investigacion, entidad, textAreaA);
        tiempo += aux;
```

Luego simulamos la probabilidad de que una pieza pase a una locación en especifico y realizamos la misma técnica primero eliminamos la pieza de la fila 1 y luego la ponemos en la siguiente locación actualizamos el grafico y así sucesivamente has llegar a la última locación.

```
lse {
 contador4++;
 cola1.remove(entidad);
 cola4.add(entidad);

 // Actualizar la gráfica para el ingreso a cola4
 actualizarGrafico(cola1, cola2, cola3, cola4, contador1, contador2, contador3, contador4);
 aux = procesol("informacion", tiempo, informacion, entidad, textAreaA);
 tiempo += aux;
 if (entidad.getNombre() != null) {
     String nombre = entidad.getNombre();
     switch (nombre) {
         case "pieza1":
             agregarTiempo(tiempoPro1, tiempoSis1, aux, tiempo);
             break;
         case "pieza2":
             agregarTiempo(tiempoPro2, tiempoSis2, aux, tiempo);
             break;
         case "pieza3":
             agregarTiempo(tiempoPro3, tiempoSis3, aux, tiempo);
             break;
         default:
             break;
     }
 }
```

Luego en la última locación mandamos los datos necesarios para que se producen y den los resultados esperados.

```
        Thread.sleep(delay); // 2000 ms = 2 segundos
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

    // Imprimir resultados finales
    textAreaB.append("Locaciones: "+ "\n");
    textAreaB.append("Piezas que pasaron por basico: " + contador1+ "\n");
    textAreaB.append("Piezas que pasaron por especialista: " + contador2+ "\n");
    textAreaB.append("Piezas que pasaron por investigacion: " + contador3+ "\n");
    textAreaB.append("Piezas que pasaron por informacion: " + contador4+ "\n");
    textAreaB.append("Capacidad de cola1: 99999.999"+ "\n");
    textAreaB.append("Capacidad de cola2: 99999.999"+ "\n");
    textAreaB.append("Capacidad de cola3: 99999.999"+ "\n");
    textAreaB.append("Capacidad de cola4: 99999.999"+ "\n");
```

Luego damos el delay para poder ver la simulación en tiempo real y procedemos a imprimir los datos necesarios para seguir la simulación.

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    int anchoFila = 100;
    int altoFila = 50;

    for (int i = 0; i < 4; i++) {
        // Cambia el color de la fila basado en la cantidad de piezas
        if (filaCounts[i] > 0) {
            g.setColor(Color.CYAN); // Celeste si hay piezas en la fila
        } else {
            g.setColor(Color.YELLOW); // Amarillo si está vacía
        }
        // Dibuja cada cuadrado en su posición individual
        g.fillRect(xPositions[i], yPositions[i], anchoFila, altoFila);
        g.setColor(Color.BLACK);
        g.drawRect(xPositions[i], yPositions[i], anchoFila, altoFila); // Dibuja el borde
        g.drawString("Fila " + (i + 1) + ": " + filaCounts[i], xPositions[i] + 10, yPositions[i] + 30);
        g.drawString("Atendidos: " + contadorEmpleado[i], xPositions[i] + 10, yPositions[i] + 15); // M
    }
}
```

Luego procedemos a crear los componentes s necesarios para poder realizar la simulación grafica y se vea todo el proceso.

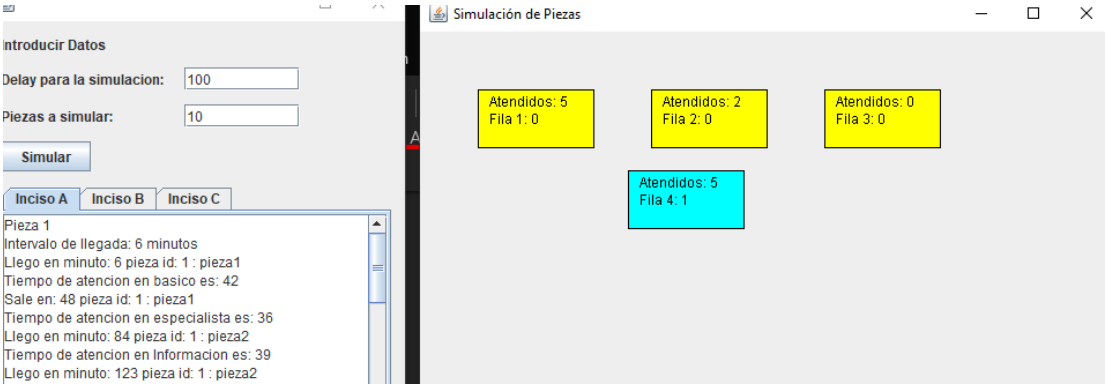
```
public int proceso1(String locacion, int tiempoSis, List<Entidad> cola, Entidad pieza, JTextArea textAreaA) {
    cola.add(pieza);
    UniformRealDistribution uniforme = new UniformRealDistribution(10, 50);
    int demora = (int) uniforme.sample();
    textAreaA.append("Tiempo de atencion en " + locacion + " es: " + demora +"\n");
    cola.remove(0);
    return demora;
}

private void actualizarGrafico(List<Entidad> cola1, List<Entidad> cola2, List<Entidad> cola3, List<Entidad> cola4,
                               int contador1, int contador2, int contador3, int contador4) {
    int[] counts = {cola1.size(), cola2.size(), cola3.size(), cola4.size()};
    int[] contadorEmp = {contador1, contador2, contador3, contador4};
    simPanel.setFilaCounts(counts);
    simPanel.setContadorEmpleado(contadorEmp);
    simPanel.repaint();
    try {
        Thread.sleep(500); // Pausa para ver el cambio
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
```

Primero creamos el proceso donde la pieza calculara el tiempo que la pieza tardara al hacer su proceso utilizando la distribución uniforme, y el siguiente método recibe datos para actualizar los gráficos de forma dinámica.

```java
public static double calcularMedia(List<Integer> numeros) {
    if (numeros == null || numeros.isEmpty()) {
        return 0;
    }
    int suma = 0;
    for (int numero : numeros) {
        suma += numero;
    }
    return (double) suma / numeros.size();
}

private void agregarTiempo(List<Integer> tiempoPro, List<Integer> tiempoSis, int aux, int tiempo) {
    tiempoPro.add(aux);
    if (tiempoSis.isEmpty()) {
        tiempoSis.add(tiempo);
    } else {
        tiempoSis.add(tiempo - tiempoSis.get(tiempoSis.size() - 1));
    }
}
```

Luego tenemos un método que calcula la media de una lista que le pasaremos y un método que se encarga de agregar datos como los tiempos a las listas correspondientes.

Ejecución:



Respuestas y comparación:

Capacidad de investigacion: 1
Capacidad de informacion: 1
Porcentaje de utilizacion de Basico: 100.0%
Porcentaje de utilizacion de especialista: 41.0%
Porcentaje de utilizacion de investigacion: 10.0%
Porcentaje de utilizacion de informacion: 100.0%
Entidades:
Piezas1: 59
Piezas2: 31
Piezas3: 10
Tiempo promedio en operacion pieza1: 30.59748427672956
Tiempo promedio en operacion pieza2: 31.083333333333332
Tiempo promedio en operacion pieza3: 29.6
Tiempo promedio en sistema pieza1: 2399.5254237288136
Tiempo promedio en sistema pieza2: 2808.1935483870966
Tiempo promedio en sistema pieza3: 3088.9

Contenido Promedio	Contenido Máximo	Contenido Actual	% Utilización
0,44	1,00	0,00	44,42
0,43	1,00	0,00	43,50
0,24	1,00	0,00	23,81
0,37	1,00	0,00	36,52

En la ejecución podemos ver que el porcentaje de utilización de cada sector es de 100% para el básico, para el especialista 41%, para el de investigación de 10% y el de informa solución de 100%. Como se puede ver hay variaciones con las de

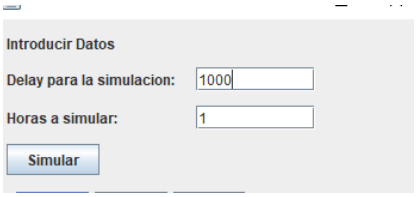
pro model debido a que las distribuciones generan números aleatorios en las diferentes simulaciones.

Ejercicio 12:

Descripción:

Un edificio 1 llega la materia prima donde al hacer uso de un vehículo este es transportado a un edificio 2.

Código e interfaz:



Para la interfaz este igualmente recibe un numero en milisegundos para el delay y el número de piezas que se desea simular.

```
private void simularEjercicioA(java.awt.event.ActionEvent evt) {
    try {
        int duracion = Integer.parseInt(tiempoSimulacion.getText());
        int delay = Integer.parseInt(numeroIteraciones.getText());
        Ejecucion3 hilo = new Ejecucion3(duracion,delay,textAreaA, textAreaB);
        hilo.start();
    } catch (NumberFormatException e) {
        JOptionPane.showMessageDialog(this, "Por favor, introduce valores numéricos válidos.", "Error de entrada", JOptionPane.ERROR_M
    }
}
```

En la parte de inicialización del programa se recogen los datos de la interfaz como el tiempo de delay y el tiempo o cantidad de objetos a simular, luego se crea una instancia para ejecutar el hilo para poder ver la simulación en simultaneo a la ejecución.

```java
package com.mycompany.promodel;

import java.awt.Color;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextArea;

public class Ejecucion3 extends Thread {
    private int tiempo;
    private int delay;
    private SimulacionPanel simPanel;
    private JTextArea textAreaA;
    private JTextArea textAreaB;

    public Ejecucion3(int tiempo, int delay, JTextArea textAreaA, JTextArea textAreaB) {
        this.tiempo = tiempo;
        this.delay = delay;
        this.textAreaA = textAreaA;
        this.textAreaB = textAreaB;
        initGrafico();
    }

    private void initGrafico() {
        JFrame frame = new JFrame("Simulación de Ejecución 3");
        simPanel = new SimulacionPanel();
        frame.setSize(500, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(simPanel);
        frame.setVisible(true);
    }
```

El método de ejecución de la simulación primero exportamos las herramientas necesarias para realizar la simulación y creamos el constructor que recibe el número de piezas el delay y las áreas de texto donde se mostraran los resultados y donde se inicializa el grafico.

```java
@Override
public void run() {
    List<Entidad> edificio1 = new ArrayList<>();
    List<Entidad> edificio2 = new ArrayList<>();
    List<Entidad> carro = new ArrayList<>();

    int contadorEdificio1 = 0; // Contador para edificio1
    int contadorEdificio2 = 0; // Contador para edificio2
    int contadorCarro = 0; // Contador para carro

    for (int minuto = 1; minuto <= tiempo * 60; minuto++) {
        Entidad entidad = new Entidad("paquete", minuto, tiempo);
        edificio1.add(entidad);
        contadorEdificio1++;
        textAreaA.append("Paquete " + minuto + " llega a edificio 1" + "\n");

        // Actualiza y dibuja el gráfico
        simPanel.setFilaCounts(new int[]{edificio1.size(), carro.size(), edificio2.size()});
        simPanel.setContadores(contadorEdificio1, contadorCarro, contadorEdificio2);
        simPanel.setCarroPosition(50); // Posición debajo del edificio 1
        simPanel.repaint();

        try {
            Thread.sleep(delay); // Espera 1 segundo
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        edificio1.remove(entidad);
        carro.add(entidad);
        contadorCarro++; // Incrementa el contador del carro
        textAreaA.append("Paquete está en carro y va a edificio 2" + "\n");
```

En el método run se ejecuta todas las operaciones necesarias para hacer la simulación, primero creamos la entidad y la agragamos a la primera locación e

imprimimos un mensaje luego actualizamos el grafico dinámico que tendrá el camión que se ira moviendo.

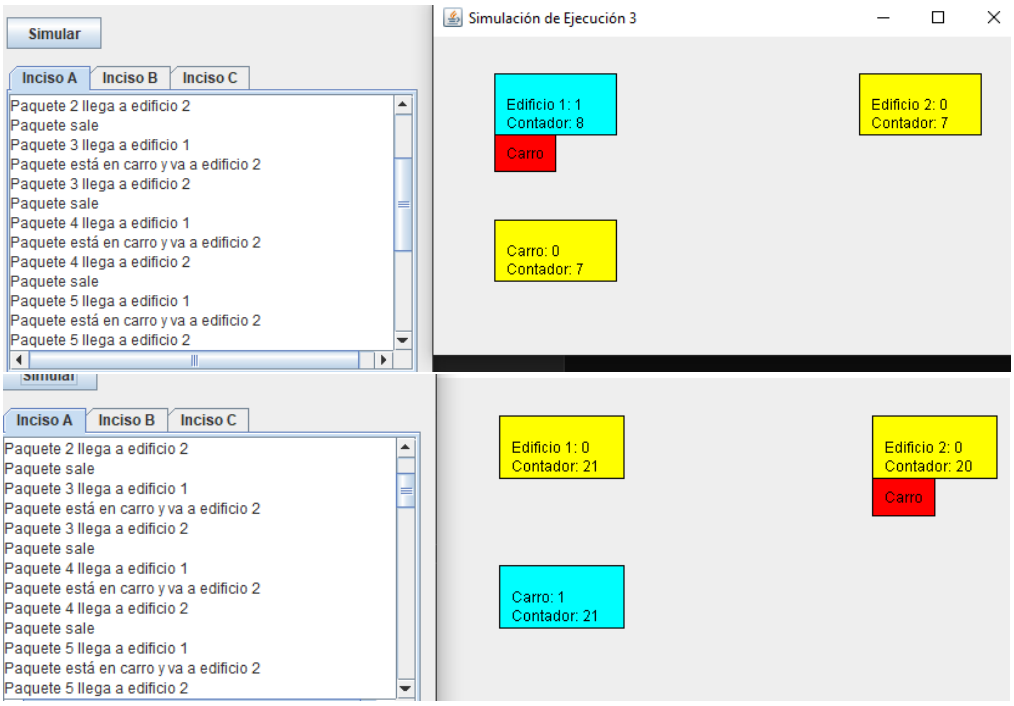
```
    // Actualiza y dibuja el gráfico
    simPanel.setFilaCounts(new int[]{edificio1.size(), carro.size(), edificio2.size()});
    simPanel.setContadores(contadorEdificio1, contadorCarro, contadorEdificio2);
    simPanel.repaint();

    try {
        Thread.sleep(delay); // Espera 1 segundo
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    edificio2.remove(entidad);
    textAreaA.append("Paquete sale" + "\n");
    // En este punto, los contadores no necesitan cambiar ya que el paquete ya ha salido
    simPanel.setFilaCounts(new int[]{edificio1.size(), carro.size(), edificio2.size()});
    simPanel.setContadores(contadorEdificio1, contadorCarro, contadorEdificio2);
    simPanel.setCarroPosition(50); // Regresa a la posición debajo del edificio 1
    simPanel.repaint();
}

textAreaB.append("Locaciones: "+ "\n");
textAreaB.append("Paquetes en edificio 1: " + contadorEdificio1+ "\n");
textAreaB.append("Paquetes en edificio 2: " + contadorEdificio2+ "\n");
textAreaB.append("Capacidad de edificio1: 1"+ "\n");
textAreaB.append("Capacidad de edificio2: 1"+ "\n");
textAreaB.append("Porcentaje de utilizacion de edificio 1: 100%"+ "\n");
textAreaB.append("Porcentaje de utilizacion de edificio 2: 100%"+ "\n");
```

Luego las piezas pasan al camión y del camión al edifico 2 donde finalmente saldrán y se agrega delay después de cada acción para poder ver cada paso del proceso y luego procedemos a imprimir los datos.

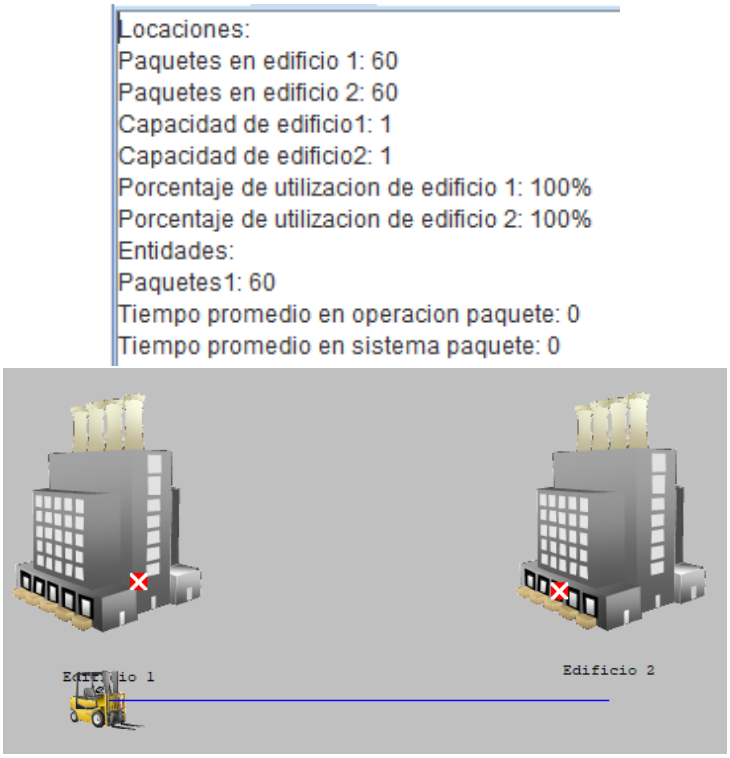
```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
    int anchoFila = 100;
    int altoFila = 50;

    // Dibuja las filas
    for (int i = 0; i < 3; i++) {
        // Cambia el color de la fila basado en la cantidad de piezas
        if (filaCounts[i] > 0) {
            g.setColor(Color.CYAN);
        } else {
            g.setColor(Color.YELLOW);
        }
        g.fillRect(xPositions[i], yPositions[i], anchoFila, altoFila);
        g.setColor(Color.BLACK);
        g.drawRect(xPositions[i], yPositions[i], anchoFila, altoFila);

        // Dibuja el nombre y el contador
        if (i == 0) {
            g.drawString("Edificio 1: " + filaCounts[i], xPositions[i] + 10, yPositions[i] + 30);
            g.drawString("Contador: " + contadorEdificio1, xPositions[i] + 10, yPositions[i] + 45);
        } else if (i == 1) {
            g.drawString("Carro: " + filaCounts[i], xPositions[i] + 10, yPositions[i] + 30);
            g.drawString("Contador: " + contadorCarro, xPositions[i] + 10, yPositions[i] + 45);
        } else {
            g.drawString("Edificio 2: " + filaCounts[i], xPositions[i] + 10, yPositions[i] + 30);
            g.drawString("Contador: " + contadorEdificio2, xPositions[i] + 10, yPositions[i] + 45);
        }
    }

    // Dibuja el carro
    g.setColor(Color.RED);
    g.fillRect(carroPosition, 80, 50, 30);
    g.setColor(Color.BLACK);
    g.drawRect(carroPosition, 80, 50, 30);
```

Luego por fin podemos construir los gráficos para la simulación y también construiremos los gráficos del carro para que se baja moviendo en el gráfico.

Ejecución:



Resultados y comparación:



Al ser un ejercicio basado en la creación de recursos que se muevan en la simulación entonces las locaciones no tienen operaciones por lo que la mayoría es 0 y solo se puede contar datos como cantidad de paquetes y porcentaje de utilización de los edificios.

Videos de los ejercicios:

Video del ejercicio 10:

https://youtu.be/2zIstk798LM

Video del ejercicio 11:

https://youtu.be/dGIoh8vRH7A

Video del ejercicio 12:

https://youtu.be/BNCtOUtXG1M