

Actividad 8

Simulación de problemas de Crystal Ball en java

Nombre: Abel Alejandro Pacheco Quispe

Materia: Taller de Simulación de Sistemas

Docente: Ayoroa Cardozo Jose Richard

Ejercicio 1:

Mezcla de productos: optimización de la fabricación

Descripción:

El problema planteado es un clásico caso de optimización en el que una empresa, Gourmet Meats, necesita determinar cuántas libras de cada uno de sus cinco tipos de salchichas debe producir para maximizar el beneficio operativo, mientras enfrenta restricciones en la cantidad de ingredientes disponibles. Los ingredientes limitados son ternera, cerdo, carne de res y tripas, y la demanda de los productos es incierta. Además, la empresa debe decidir si es rentable comprar ingredientes adicionales o vender los excedentes, considerando los costos asociados a estos procesos.

El modelo utiliza Crystal Ball para manejar la incertidumbre en la cantidad de ingredientes utilizados, los costos y la demanda, a través de distribuciones de probabilidad. Luego, OptQuest se encarga de la optimización, utilizando las cantidades de producción de cada tipo de salchicha y las decisiones de compra o venta de ingredientes como variables controlables. El objetivo final es encontrar la combinación óptima de productos que maximice el beneficio operativo, mientras se respetan las limitaciones de inventario y otras restricciones.

```
import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.math3.distribution.NormalDistribution;
import org.apache.commons.math3.distribution.LogNormalDistribution;
import org.apache.commons.math3.distribution.TriangularDistribution;
import java.text.DecimalFormat;
```

Primero importamos las dependencias necesarias que son la distribución normal, triangular y logarítmica normal.

```
{
// Obtener valor del campo de texto
int iteraciones = Integer.parseInt(numeroIteraciones.getText());
textAreaA.append("Número de iteraciones: " + iteraciones + "\n");
List<Double> totalGP = new ArrayList<>();
List<Double> totalBE = new ArrayList<>();
int i=0;
while(i < iteraciones){
    double markup = 0;
    double costoAlmacenarLbs = 0;
    double procesamientoPedidos = 0;
    List<Double> veal = new ArrayList<>();
    List<Double> pork = new ArrayList<>();
    List<Double> beef = new ArrayList<>();
    List<Double> casing = new ArrayList<>();
    List<Double> summer = new ArrayList<>();
    List<Double> bratwurst = new ArrayList<>();
    List<Double> italian = new ArrayList<>();
    List<Double> peperoni = new ArrayList<>();
    List<Double> polish = new ArrayList<>();
    List<Double> precios = new ArrayList<>();
    List<Double> demandaMedia = new ArrayList<>();
    List<Double> demandaMensual = new ArrayList<>();
    List<Double> unidadesProdu = new ArrayList<>();
    textAreaA.append("Iteración " + (i+1) + "\n");
    if(i == 0){
```

Luego inicializamos todas las variables necesarias para realizar el ejercicio en estecaso las importantes son totalGP (Ganancia potencial) y totalBE (beneficioTotal) que son los datos que se quieren analizar.

```

if(i == 0){
    //dist
    markup = 0.3;
    costoAlmacenarLbs = 0.75;
    procesamientoPedidos = 0.05;
    //horizontal
    veal.addAll(List.of(0.0, 4.0, 1.0, 0.0, 0.0, 8.0));
    pork.addAll(List.of(2.5, 1.0, 3.0, 4.0, 1.0, 3.25));
    beef.addAll(List.of(1.0, 0.0, 1.5, 0.0, 3.0, 4.5));
    casing.addAll(List.of(1.0, 1.5, 1.0, 2.0, 1.5, 0.5));//dis
    //vertical
    summer.addAll(List.of(0.0, 2.5, 1.0, 1.0));
    bratwurst.addAll(List.of(4.0, 1.0, 0.0, 1.5));
    italian.addAll(List.of(1.0, 3.0, 1.5, 1.0));
    peperoni.addAll(List.of(0.0, 4.0, 0.0, 2.0));
    polish.addAll(List.of(0.0, 1.0, 3.0, 1.5));
    precios.addAll(List.of(8.0, 3.25, 4.5, 0.5));//dis
    //Demanda
    demandaMedia.addAll(List.of(3600.0, 7000.0, 4225.0, 5500.0, 8500.0));
    demandaMensual.addAll(List.of(3600.0, 7000.0, 4225.0, 5500.0, 8500.0));//dis
    unidadesProdu.addAll(List.of(1000.0, 1100.0, 750.0, 900.0, 1500.0, 0.0));
}

```

Con un if vemos si es la primera iteración y si es así procedemos a introducir los datos de forma manual desde las probabilidades, las cantidades de ternera, cerdo, vaca y viseras, los respectivos precios y la demanda mensual y las unidades máximas producidas.

```

}else{
    //dist
    NormalDistribution normalDistribution = new NormalDistribution(0.3, 0.03);
    markup = normalDistribution.sample();
    NormalDistribution normalDistributionC = new NormalDistribution(0.75, 0.05);
    costoAlmacenarLbs = normalDistributionC.sample();
    NormalDistribution normalDistributionP = new NormalDistribution(0.1, 0.08);
    procesamientoPedidos = normalDistributionP.sample();
    //horizontal
    TriangularDistribution triangularDistribution11 = new TriangularDistribution(6.5, 8.0, 10);
    double costo1 = triangularDistribution11.sample();
    TriangularDistribution triangularDistribution12 = new TriangularDistribution(2.75, 3.25, 4.75);
    double costo2 = triangularDistribution12.sample();
    TriangularDistribution triangularDistribution13 = new TriangularDistribution(3.0, 4.5, 5.0);
    double costo3 = triangularDistribution13.sample();
    TriangularDistribution triangularDistribution14 = new TriangularDistribution(0.45, 0.5, 0.7);
    double costo4 = triangularDistribution14.sample();

    veal.addAll(List.of(0.0, 4.0, 1.0, 0.0, 0.0, costo1));
    pork.addAll(List.of(2.5, 1.0, 3.0, 4.0, 1.0, costo2));
    beef.addAll(List.of(1.0, 0.0, 1.5, 0.0, 3.0, costo3));
    TriangularDistribution triangularDistribution1 = new TriangularDistribution(1.0, 1.0, 1.25);
    double dato1 = triangularDistribution1.sample();
    TriangularDistribution triangularDistribution2 = new TriangularDistribution(1.5, 1.5, 1.65);
    double dato2 = triangularDistribution2.sample();
    TriangularDistribution triangularDistribution3 = new TriangularDistribution(1.0, 1.0, 1.25);
    double dato3 = triangularDistribution3.sample();
    TriangularDistribution triangularDistribution4 = new TriangularDistribution(2.0, 2.0, 2.2);
    double dato4 = triangularDistribution4.sample();
    TriangularDistribution triangularDistribution5 = new TriangularDistribution(1.5, 1.5, 1.65);
    double dato5 = triangularDistribution5.sample();
    casing.addAll(List.of(dato1, dato2, dato3, dato4, dato5, costo4));//dis
    System.out.println("1" + casing);
}

```

Si no es la primera iteración procedemos a usar las diferentes distribuciones para calcular los datos de entrada como los porcentajes y precios con la distribución normal,

los costos de los diferentes ingredientes utilizan la distribución triangular, la cantidad de viseras utilizada utiliza una distribución triangular y la demanda mensual utiliza la distribución logarítmica normal.

```
//Inventario
List<Double> invVe = new ArrayList<>(List.of(12520.0, 0.0, MYS(veal,unidadesProdu)));
invVe.add(invVe.get(0)+ invVe.get(1) - invVe.get(2));
invVe.add(Math.max(invVe.get(3)* costoAlmacenarLbs, 0));
textAreaA.append("Inventario 1: " + invVe + "\n");

List<Double> invPo = new ArrayList<>(List.of(14100.0, 0.0, MYS(pork,unidadesProdu)));
invPo.add(invPo.get(0)+ invPo.get(1) - invPo.get(2));
invPo.add(Math.max(invPo.get(3)* costoAlmacenarLbs, 0));
textAreaA.append("Inventario 2: " + invPo + "\n");

List<Double> invBe = new ArrayList<>(List.of(6480.0, 200.0, MYS(beef,unidadesProdu)));
invBe.add(invBe.get(0)+ invBe.get(1) - invBe.get(2));
invBe.add(Math.max(invBe.get(3)* costoAlmacenarLbs, 0));
textAreaA.append("Inventario 3: " + invBe + "\n");

List<Double> invCa = new ArrayList<>(List.of(10800.0, 0.0, MYS(casing,unidadesProdu)));
invCa.add(invCa.get(0)+ invCa.get(1) - invCa.get(2));
invCa.add(Math.max(invCa.get(3)* costoAlmacenarLbs, 0));
textAreaA.append("Inventario 4: " + invCa + "\n");
textAreaA.append("SummerSausage-Bratwurst-ItalianSausage-Pepperoni-PolishSausage\");
```

Luego procedemos a calcular los datos de los inventarios que en este caso nos encargamos en pasa los primeros 3 datos y vamos añadiendo los demás haciendo cálculos de los datos ya introducidos.

```
DecimalFormat df = new DecimalFormat("#.##");
//Costos
List<Double> costoUnitario = new ArrayList<>(List.of(MYS(summer,precios), MYS(bratwurst,precios), MYS(italian,precios),
MYS(peperoni,precios), MYS(polish,precios)));
textAreaA.append("Costos: " + String.join(" ", costoUnitario.stream().map(d -> df.format(d)).toList());

List<Double> peso = new ArrayList<>(List.of(sumar(summer),sumar(bratwurst),sumar(italian),sumar(peperoni),sumar(polish)));
textAreaA.append("Peso: " + String.join(" ", peso.stream().map(d -> df.format(d)).toList()) + "\n");

List<Double> salchichasPaquete = new ArrayList<>(List.of(peso.get(0)*4,peso.get(1)*4,peso.get(2)*4,peso.get(3)*4,peso.get(4)*4));
textAreaA.append("# Salchichas: " + String.join(" ", salchichasPaquete.stream().map(d -> df.format(d)).toList());

List<Double> costo20 = new ArrayList<>(List.of((costoUnitario.get(0)/salchichasPaquete.get(0))*20,(costoUnitario.get(1)/salchichasPaquete.get(1))*20,(costoUnitario.get(2)/salchichasPaquete.get(2))*20,(costoUnitario.get(3)/salchichasPaquete.get(3))*20,(costoUnitario.get(4)/salchichasPaquete.get(4))*20));
textAreaA.append("Costo20: " + String.join(" ", costo20.stream().map(d -> df.format(d)).toList()) + "\n");

List<Double> precioVent = new ArrayList<>(List.of(costo20.get(0)*(1+markup), costo20.get(1)*(1+markup), costo20.get(2)*(1+markup), costo20.get(3)*(1+markup), costo20.get(4)*(1+markup)));
textAreaA.append("PrecioVenta: " + String.join(" ", precioVent.stream().map(d -> df.format(d)).toList()) + "\n");

List<Double> ganancia = new ArrayList<>(List.of(precioVent.get(0)-costo20.get(0), precioVent.get(1)-costo20.get(1), precioVent.get(2)-costo20.get(2), precioVent.get(3)-costo20.get(3), precioVent.get(4)-costo20.get(4)));
textAreaA.append("Ganancia: " + String.join(" ", ganancia.stream().map(d -> df.format(d)).toList()) + "\n");

List<Double> gananciaLbs = new ArrayList<>(List.of(ganancia.get(0)/peso.get(0), ganancia.get(1)/peso.get(1), ganancia.get(2)/peso.get(2), ganancia.get(3)/peso.get(3), ganancia.get(4)/peso.get(4)));
textAreaA.append("GananciaLbs: " + String.join(" ", gananciaLbs.stream().map(d -> df.format(d)).toList()) + "\n");

// Tabla min y max
textAreaA.append("Tabla minima y maxima: \n");
textAreaA.append("UnidadesProducir: " + String.join(" ", unidadesProdu.stream().map(d -> df.format(d)).toList()) + "\n");
```

Luego procedemos a calcular los datos de los costos con sus respectivas formulas donde cada uno utiliza datos previamente introducidos y posteriormente son mandados a ser impresos.

```

// Resultados
List<Double> gal = new ArrayList<>(List.of(costoA.get(0), costoInv.get(0), invAdj.get(0)));
List<Double> ga2 = new ArrayList<>(List.of(costoA.get(1), costoInv.get(1), invAdj.get(1)));
List<Double> ga3 = new ArrayList<>(List.of(costoA.get(2), costoInv.get(2), invAdj.get(2)));
List<Double> ga4 = new ArrayList<>(List.of(costoA.get(3), costoInv.get(3), invAdj.get(3)));
List<Double> ga5 = new ArrayList<>(List.of(costoA.get(4), costoInv.get(4), invAdj.get(4)));

List<Double> gananciaP = new ArrayList<>(List.of(sumar(gal), sumar(ga2), sumar(ga3), sumar(ga4), sumar(ga5)));
textAreaA.append("Ganancia potencial: " + String.join(" ", gananciaP.stream().map(d -> df.format(
List<Double> beneficio = new ArrayList<>(List.of((gananciaLbs.get(0)*demandaMensual.get(0))+sumar(gal), (gananciaLbs.get(1)*demandaMensual.get(1))+sumar(ga2), (gananciaLbs.get(2)*demandaMensual.get(2))+sumar(ga3), (gananciaLbs.get(3)*demandaMensual.get(3))+sumar(ga4), (gananciaLbs.get(4)*demandaMensual.get(4))+sumar(ga5)));
textAreaA.append("Beneficio proyectado: " + String.join(" ", beneficio.stream().map(d -> df.format(

totalGP.add(sumar(gananciaP));
totalBE.add(sumar(beneficio));
textAreaA.append("-----\n");
i++;
}
System.out.println(totalBE);
SwingUtilities.invokeLater(() -> {
    BeneficioOperativoChart chart = new BeneficioOperativoChart("Gráfica de Beneficio Operativo", totalBE, totalGP);
    chart.setSize(800, 600);
    chart.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    chart.setVisible(true);
});
}
}

```

Una vez llegamos al final de los cálculos procedemos a guardar los datos que nos interesan en las listas totalGP y totalBE para posteriormente enviarlos a ser graficados.

Ejecución:

Para la ejecución primero introducimos el numero de iteraciones que deseamos simular.

Luego procedemos a apretar el botón simular y esperamos a que nos arrojen los datos.

Tenemos la iteración 1 que coincide con los datos arrojados por cristal ball en Excel por lo que podemos decir que el programa si simula el modelo deseado.

Demanda media	3600 Lbs	7000 Lbs	4225 Lbs	5500 Lbs	8500 Lbs	
Demanda mensual en Lbs	3600 Lbs	7000 Lbs	4225 Lbs	5500 Lbs	8500 Lbs	28825 Lbs
Units to produce?	1.000	1.100	750	900	1.500	5250,00
Min	560	754	455	642	1.082	
Max	1.200	1.615	975	1.375	2.318	
Production in Lbs	4500 Lbs	7150 Lbs	4875 Lbs	5400 Lbs	8250 Lbs	30175 Lbs
Superávit / Déficit Lbs	900 Lbs	150 Lbs	650 Lbs	-100 Lbs	-250 Lbs	-250,00
costo venta perdido	\$0	\$0	\$0	(\$58)	(\$217)	(\$275)
costo de almacenar produc	(\$675)	(\$113)	(\$488)	\$0	\$0	(\$1.275)
Costo de inventario	(\$1.304)	(\$2.536)	(\$1.531)	(\$1.993)	(\$3.080)	(\$10.444)
Inventory Adj. (ajuste)	(\$107)	(\$208)	(\$125)	(\$163)	(\$252)	(\$855)
Mejora ganancia potenci	(\$2.086)	(\$2.856)	(\$2.144)	(\$2.214)	(\$3.549)	(\$12.849)
Beneficio opertivo	\$1.413,88	\$6.090,40	\$1.606,40	\$1.052,46	\$4.044,22	\$14.207,36

Iteración 1

Inventario 1: [12520.0, 0.0, 5150.0, 7370.0, 5527.5]

Inventario 2: [14100.0, 0.0, 10950.0, 3150.0, 2362.5]

Inventario 3: [6480.0, 200.0, 6625.0, 55.0, 41.25]

Inventario 4: [10800.0, 0.0, 7450.0, 3350.0, 2512.5]

SummerSausage-Bratwurst-ItalianSausage-Pepperoni-PolishSausage

Costos:	13,12	36	25	14	17,5	
Peso:	4,5	6,5	6,5	6	5,5	
# Salchicas:	18	26	26	24	22	
Costo20:	14,58	27,69	19,23	11,67	15,91	
PreciVenta:	18,96	36	25	15,17	20,68	
Ganancia:	4,38	8,31	5,77	3,5	4,77	
GananciaLbs:	0,97	1,28	0,89	0,58	0,87	
Tabla mínima y máxima:						
UnidadesProcurar:	1000	1100	750	900	1500	0
minimi:	560	753,85	455	641,67	1081,82	
Maximo:	1200	1615,38	975	1375	2318,18	
Produccion/Lbs:	4500	7150	4875	5400	8250	
Superhabit/Deficit:	900	150	650	-100	-250	
CostoVentaP:	0	0	0	-58,33	-216,94	
CostoAlmacen:	-675	-112,5	-487,5	0	0	
Costo de Inventario:	-1304,34	-2536,21	-1530,78	-1992,74	-3079,68	
Inventario Ajustado:	-106,78	-207,63	-125,32	-163,14	-252,12	
Ganancia potencial:	-2086,12	-2856,34	-2143,6	-2155,88	-3331,81	
Beneficio proyectado:	1413,88	6090,4	1606,4	1052,46	4044,22	

Y también procedemos a ver el resultado después de 50 iteraciones y vemos que si bien no es igual por lo menos es similar.

Iteración 50

Inventario 1: [12520.0, 0.0, 5150.0, 7370.0, 5669.158854180797]

Inventario 2: [14100.0, 0.0, 10950.0, 3150.0, 2423.0461859795805]

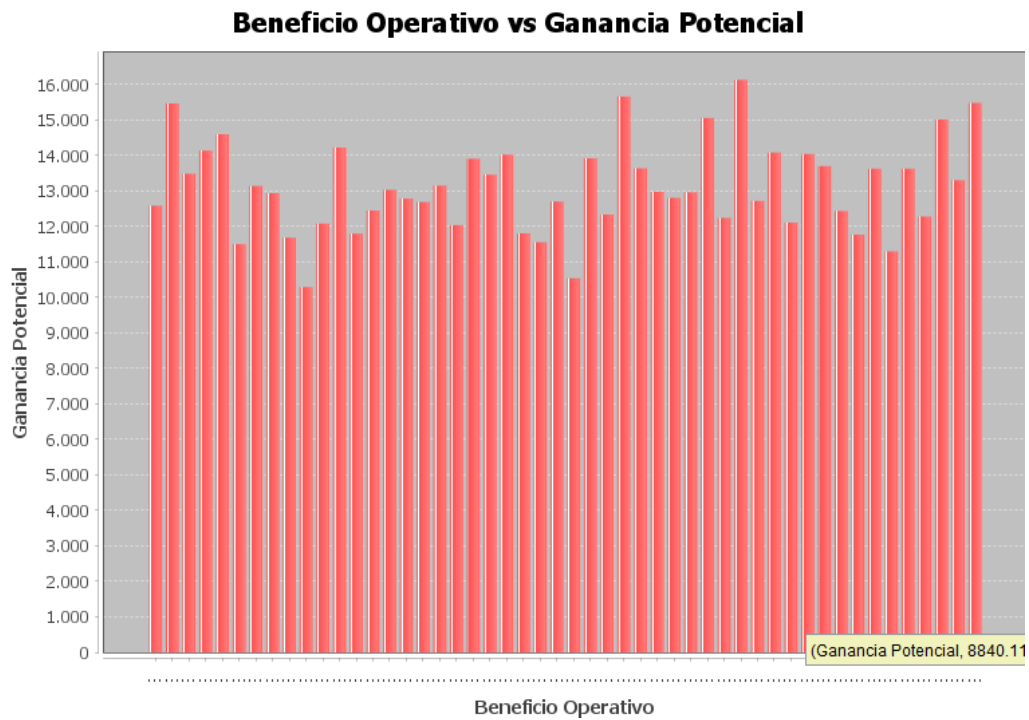
Inventario 3: [6480.0, 200.0, 6625.0, 55.0, 42.3071556282149]

Inventario 4: [10800.0, 0.0, 7706.319261630784, 3093.6807383692158, 2379.7242265854034]

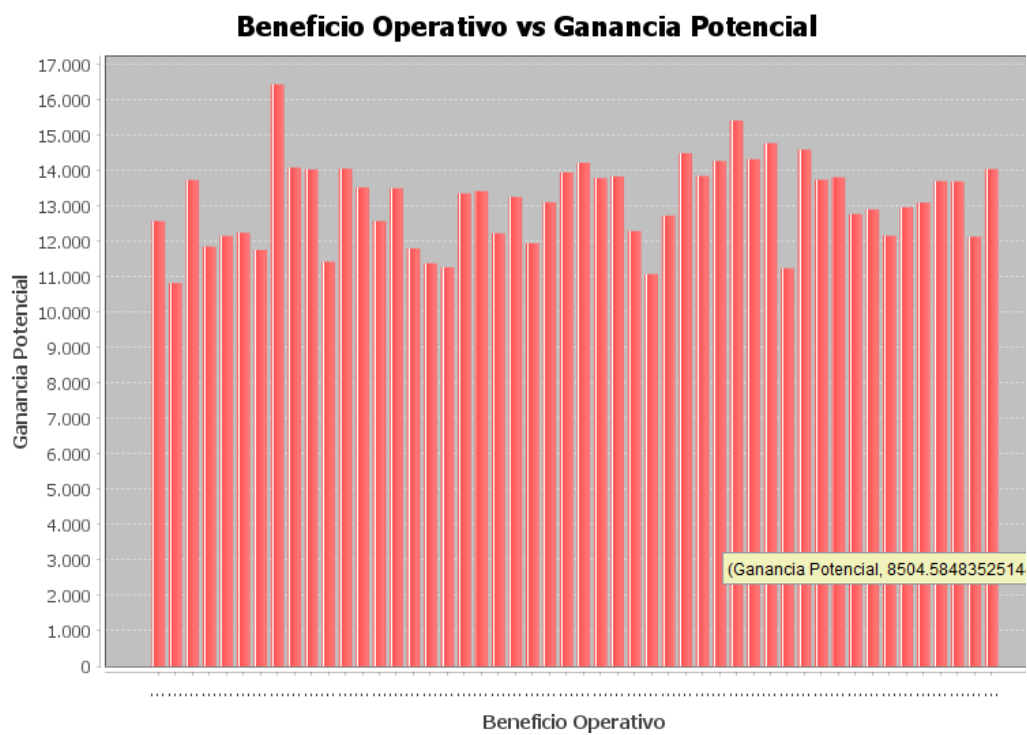
SummerSausage-Bratwurst-ItalianSausage-Pepperoni-PolishSausage

Costos:	14,71	40,02	27,69	16,63	17,97	
Peso:	4,59	6,5	6,6	6,05	5,53	
# Salchicas:	18,34	26	26,41	24,18	22,14	
Costo20:	16,04	30,78	20,97	13,75	16,23	
PreciVenta:	21,27	40,83	27,82	18,24	21,54	
Ganancia:	5,24	10,06	6,85	4,49	5,3	
GananciaLbs:	1,14	1,55	1,04	0,74	0,96	
Tabla mínima y máxima:						
UnidadesProcurar:	1000	1100	750	900	1500	0
minimi:	549,57	753,79	447,9	636,79	1075,06	
Maximo:	1177,66	1615,27	959,79	1364,55	2303,71	
Produccion/Lbs:	4585,37	7150,52	4952,24	5441,35	8301,84	
Superhabit/Deficit:	1308,49	844,77	1418,85	1275,6	599,09	
CostoVentaP:	0	0	0	0		
CostoAlmacen:	-1006,52	-649,82	-1091,41	-981,22	-460,84	
Costo de Inventario:	-1379,01	-2653,65	-1486,96	-1753,07	-3241,55	
Inventario Ajustado:	-100,37	-193,15	-108,23	-127,6	-235,94	
Ganancia potencial:	-2485,9	-3496,62	-2686,6	-2861,89	-3938,32	
Beneficio proyectado:	1257,82	6257,74	979,55	233,48	3442,19	

Y procedemos a ver el grafico proporcionado que indica el beneficio y el margen de ganancia que este puede tener.



Probando para otra cantidad de 50 tenemos que.



Conclusión:

En conclusión, podemos decir que seguir utilizando los valores actuales o semejantes según la simulación puede traer una probabilidad de beneficio operativo mayor mente positiva y una mejora en la ganancia potencial muy positiva.

Ejercicio 2:

Ruta crítica modelo de programación estocástica

Descripción:

Este modelo analiza el proceso de la programación del proyecto o el tiempo que llevaría llevar un producto al mercado, con el objetivo de comprender cómo la **incertidumbre afecta la finalización del proyecto**. Al final de la hoja de cálculo del modelo, un diagrama muestra el patrón de flujo de las tareas. En términos de Six Sigma, el **defecto puede definirse como la diferencia entre el tiempo real de finalización del proyecto y el tiempo mínimo de finalización del proyecto**. Los resultados de este modelo indican la probabilidad de que una tarea en particular esté en el camino crítico, y el modelo puede usarse para evaluar qué tareas clave deben abordarse para mejorar los resultados de todo el proyecto.

```
package com.mycompany.cristallball;
import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.math3.distribution.NormalDistribution;
import org.apache.commons.math3.distribution.LogNormalDistribution;
```

Importamos las distribuciones normal y logarítmica que son las que utilizaremos.

```
// Obtener valor del campo de texto
double costoTenencia = Double.parseDouble(costoTenenciaInventario.getText());
int iteraciones = Integer.parseInt(numeroIteraciones.getText());
textAreaA.append("Costo de tenencia: " + costoTenencia + "\n");
textAreaA.append("Número de iteraciones: " + iteraciones + "\n");
//Ejercicio 2
double leadLag = 0;
double duracionOp = 0;
double duracionEsp = 0;
double duracionPe = 0;
double valorSimular = 0;
double EST = 0;
double EFT = 0;
double LST = 0;
double LFT = 0;
double slack = 0;
boolean rutaCritica = false;
double probabilidad = 0;
List<Integer> inicioMasTardio = new ArrayList<>();
List<Double> probabilidades = new ArrayList<>();
List<Integer> c1 = new ArrayList<>();
List<Integer> c2 = new ArrayList<>();
List<Integer> c3 = new ArrayList<>();
... ..
```

Inicializamos los datos necesarios para realizar el ejercicio como los datos fijos de lag, duración optimista, esperada y pesimista, el valor a simular y los datos a calcular EST, EFT, LST, LFT y si es o no una ruta crítica.


```

List<Integer> c20 = new ArrayList<>();
int i = 0;
while(i < iteraciones){
    textAreaA.append("Iteración: " + (i+1) + "\n");
    if(i == 0){
        textAreaA.append(String.format("%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s\n",
            "Tarea", "Lead/Lag", "Dur.Opt.", "Dur.Esp.", "Dur.Pes.", "ValorSim.",
            "EST", "EFT", "LST", "LFT", "Slack", "Ruta Crítica", "Probabilidad"));
        List<Integer> predecesores0 = new ArrayList<>();
        Tarea tarea0 = new Tarea("Task 0", predecesores0, 0, 0, 0, 0, 0);
        List<Integer> EFTant0 = new ArrayList<>();
        tarea0.calcularDatosIni(EFTant0);

        List<Integer> predecesores1 = new ArrayList<>(List.of(0));
        Tarea tarea1 = new Tarea("Task 1", predecesores1, 0, 10, 15, 20, 15);
        List<Integer> EFTant1 = new ArrayList<>(List.of(tarea0.getEFT()));
        tarea1.calcularDatosIni(EFTant1);
        List<Integer> predecesores2 = new ArrayList<>(List.of(1));
        Tarea tarea2 = new Tarea("Task 2", predecesores2, -5, 15, 20, 22, 20);
        List<Integer> EFTant2 = new ArrayList<>(List.of(tarea1.getEFT()));
        tarea2.calcularDatosIni(EFTant2);

        List<Integer> predecesores3 = new ArrayList<>(List.of(2));
        Tarea tarea3 = new Tarea("Task 3", predecesores3, 0, 21, 26, 30, 26);
        List<Integer> EFTant3 = new ArrayList<>(List.of(tarea2.getEFT()));
        tarea3.calcularDatosIni(EFTant3);
    }
}

```

Luego iniciamos un bucle con un if para ver si es la primera iteración y si es así procedemos a pasar los datos de forma manual que son el constructor new tarea y el método de calculo es manual que es el método calcular datos iniciales.

```

}else{
    //List<Tarea> listaDeTareas = new ArrayList<>();
    textAreaA.append(String.format("%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s%-20s\n",
        "Tarea", "Lead/Lag", "Dur.Opt.", "Dur.Esp.", "Dur.Pes.", "ValorSim.",
        "EST", "EFT", "LST", "LFT", "Slack", "Ruta Crítica", "Probabilidad"));

    List<Integer> predecesores0 = new ArrayList<>();
    Tarea tarea0 = new Tarea("Task 0", predecesores0, 0, 0, 0, 0, 0);
    List<Integer> EFTant0 = new ArrayList<>();
    tarea0.calcularDatos(EFTant0, 0);

    List<Integer> predecesores1 = new ArrayList<>(List.of(0));
    Tarea tarea1 = new Tarea("Task 1", predecesores1, 0, 10, 15, 20);
    List<Integer> EFTant1 = new ArrayList<>(List.of(tarea0.getEFT()));
    tarea1.calcularDatos(EFTant1, 0);

    List<Integer> predecesores2 = new ArrayList<>(List.of(1));
    Tarea tarea2 = new Tarea("Task 2", predecesores2, -5, 15, 20, 22);
    List<Integer> EFTant2 = new ArrayList<>(List.of(tarea1.getEFT()));
    tarea2.calcularDatos(EFTant2, tarea1.getLFT());
}

```

Si no es la primera iteración se pasan solo los datos necesarios con el constructor new tarea y el método de calcular datos.

```

textAreaA.append(tarea14.obtenerResumen());
textAreaA.append(tarea15.obtenerResumen());
textAreaA.append(tarea16.obtenerResumen());
textAreaA.append(tarea17.obtenerResumen());
textAreaA.append(tarea18.obtenerResumen());
textAreaA.append(tarea19.obtenerResumen());
textAreaA.append(tarea20.obtenerResumen());
textAreaA.append("-----");
inicioMasTardio.add(tarea20.getLFT());
if (tarea1.getCritica()) {
    c1.add(1);
} else {
    c1.add(0);
}
if (tarea2.getCritica()) {
    c2.add(1);
} else {
    c2.add(0);
}

```

Una vez realizados los cálculos de cada una de las 20 tareas procedemos a imprimir los resultados y introducimos las tareas critica a una lista para analizarlas posteriormente.

```
766 probabilidades.add(calcularProbabilidadDeUno(c13));
767 probabilidades.add(calcularProbabilidadDeUno(c14));
768 probabilidades.add(calcularProbabilidadDeUno(c15));
769 probabilidades.add(calcularProbabilidadDeUno(c16));
770 probabilidades.add(calcularProbabilidadDeUno(c17));
771 probabilidades.add(calcularProbabilidadDeUno(c18));
772 probabilidades.add(calcularProbabilidadDeUno(c19));
773 probabilidades.add(calcularProbabilidadDeUno(c20));
774 SwingUtilities.invokeLater(() -> {
775     GraficoProbabilidad chart = new GraficoProbabilidad(probabilidades);
776     chart.setSize(800, 600);
777     chart.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
778     chart.setVisible(true);
779 });
780 SwingUtilities.invokeLater(() -> {
781     GraficoMasTardio chart = new GraficoMasTardio(inicioMasTardio);
782     chart.setSize(800, 600);
783     chart.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
784     chart.setVisible(true);
785 });
```

Procedemos a guardar los datos de las tareas criticas en una lista individual y lo mandamos a imprimir.

```
// Constructor
public Tarea(String nombre, List<Integer> predecesores, int leadLag, int duracionOptimista, int duracionEsperada,
    , int duracioPesimista) {
    this.nombre = nombre;
    this.predecesores = predecesores;
    this.leadLag = leadLag;
    this.duracionOptimista = duracionOptimista;
    this.duracionEsperada = duracionEsperada;
    this.duracioPesimista = duracioPesimista;
}

public Tarea(String nombre, List<Integer> predecesores, int leadLag, int duracionOptimista, int duracionEsperada,
    , int duracioPesimista, int valorSimular) {
    this.nombre = nombre;
    this.predecesores = predecesores;
    this.leadLag = leadLag;
    this.duracionOptimista = duracionOptimista;
    this.duracionEsperada = duracionEsperada;
    this.duracioPesimista = duracioPesimista;
    this.valorSimular = valorSimular;
}
```

El objeto tarea tiene dos constructores uno que pide de forma manual el valor simulado y otro que no pide el valor simulado para calcularlo con la distribución.

```
public void calcularValorSimulado() {
    if(predecesores.isEmpty()){
        valorSimular = 0;
    }else{
        TriangularDistribution triangularDistribution = new TriangularDistribution(
            duracionOptimista, duracionEsperada, duracioPesimista);
        valorSimular = (int) triangularDistribution.sample();
    }
}
```

El método para calcular el valor simulado con la distribución triangular cuando no se Introduzca manualmente.

```

public void calcularESTyEFT(List<Integer> predecesoresl) { //lista de EFTant
    if(predecesoresl.isEmpty()){
        est = 0;
        eft = 0;
    }else{
        int tamaño = predecesoresl.size();
        if(tamaño == 1){
            est = predecesoresl.get(0) + leadLag; // Tiempo de inicio temprano
            eft = est + valorSimular; // Tiempo de finalización temprano
        }else{
            int maximo = predecesoresl.get(0);
            for (int i = 1; i < predecesoresl.size(); i++) {
                if (predecesoresl.get(i) > maximo) {
                    maximo = predecesoresl.get(i);
                }
            }
            est = maximo + leadLag;
            eft = est + valorSimular;
        }
    }
}

```

Tenemos el método para calcular el EST y EFT que recibe una lista de predecesores y luego pregunta si esta vacía la lista si no pregunta el tamaño y si es igual a 1 se calcula de una forma y si es más de 1 entonces se calcula en conjuntos.

```

// Método para calcular el LST y LFT
public void calcularLSTyLFT(List<Integer> antecesorl) {
    if(antecesorl.isEmpty()){
        lst = est;
        lft = lst + valorSimular;
    }else{
        int tamaño = antecesorl.size();
        if(tamaño == 1){
            lst = antecesorl.get(0) - valorSimular;
            lft = lst + valorSimular;
        }else{
            int minimo = antecesorl.get(0);
            for (int i = 1; i < antecesorl.size(); i++) {
                if (antecesorl.get(i) < minimo) {
                    minimo = antecesorl.get(i);
                }
            }
            lst = minimo - valorSimular;
            lft = lst + valorSimular;
        }
    }
}

```

De igual forma para calcular el LST y LFT que sigue una lógica similar a la anterior.

```

//Caso especial 1
public void calcularDatosV1(List<Integer> LFTant, int lag){
    calcularLSTyLFT1(LFTant, lag);
    calcularSlack();
    critica();
}

public void calcularLSTyLFT1(List<Integer> antecesorl, int lag) {
    lst = antecesorl.get(0) - lag;
    lft = lst + valorSimular;
}

//Caso especial 2
public void calcularDatosV2(List<Integer> LFTant, int lag){
    calcularLSTyLFT2(LFTant, lag);
    calcularSlack();
    critica();
}

public void calcularLSTyLFT2(List<Integer> antecesorl, int lag) {
    lst = antecesorl.get(0) - valorSimular - lag;
    lft = lst + valorSimular;
}

//Caso especial 3
public void calcularDatosV3(int min){
    calcularLSTyLFT3(min);
    calcularSlack();
    critica();
}

public void calcularLSTyLFT3(int lag) {
    lst = lag;
    lft = lst + valorSimular;
}

```

Este caso tiene casos especiales para su calculo y se calculan de la siguiente manera.

Ejecución:

Para la ejecución de igual manera nos pide el dato de numero de iteraciones que requiere para simular y luego se presiona el botón simular.

Numero de iteraciones: 20

Iteración: 1

Tarea	Lead/Lag	Dur.Opt.	Dur.Esp.	Dur.Pes.	ValorSim.	EST	EFT	LST	LFT	Slack	Ruta Critica	Probabilidad
Task 0	0	0	0	0	0	0	0	0	0	0	Sí	0,00
Task 1	0	10	15	20	15	0	15	3	18	3	No	0,00
Task 2	-5	15	20	22	20	10	30	13	33	3	No	0,00
Task 3	0	21	26	30	26	30	56	33	59	3	No	53,00
Task 4	10	15	18	23	18	20	38	26	44	6	No	58,00
Task 5	0	13	15	17	15	38	53	44	59	6	No	35,00
Task 6	0	30	38	45	38	56	94	59	97	3	No	0,00
Task 7	-5	20	25	30	25	89	114	92	117	3	No	50,00
Task 8	5	10	15	20	15	99	114	102	117	3	No	49,00
Task 9	15	11	18	22	18	71	89	99	117	28	No	31,00
Task 10	0	23	30	45	30	114	144	117	147	3	No	0,00
Task 11	5	22	28	39	28	149	177	152	180	3	No	15,00
Task 12	0	120	140	180	140	0	140	0	140	0	Sí	0,00
Task 13	-5	13	18	22	18	135	153	135	153	0	Sí	0,00
Task 14	10	15	20	25	20	145	165	145	165	0	Sí	68,00
Task 15	0	10	15	20	15	165	180	165	180	0	Sí	47,00
Task 16	0	30	33	60	33	180	213	180	213	0	Sí	56,00
Task 17	0	5	8	11	8	213	221	213	221	0	Sí	75,00
Task 18	0	10	15	25	15	221	236	223	238	2	No	85,00
Task 19	0	13	17	19	17	221	238	221	238	0	Sí	15,00
Task 20	0	20	25	45	25	238	263	238	263	0	Sí	15,00

Si comparamos los datos de la primera iteración podemos ver que coinciden con los datos proporcionado por cristal ball por lo que podemos decir que el modelo es valido para realizar la simulación.

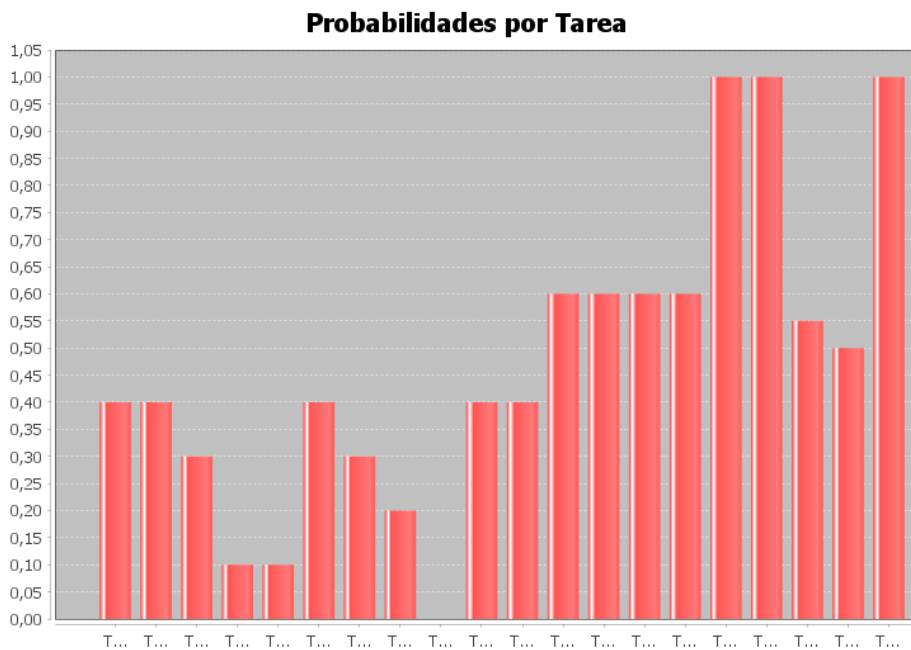
Numero tarea	Predecesor	Lead/ Lag	Duración optimista	Duración esperada	Duración pesimista	Valor Simula	EST	EFT	LST	LFT	Slack	Ruta critica	Prob. on CP
Start						0	0	0	0	0	0		
Task 1	Start FS		10	15	20	15	0	15	3	18	3	0	39%
Task 2	1 FS	-5	15	20	22	20	10	30	13	33	3	0	39%
Task 3	2 FS		21	26	30	26	30	56	33	59	3	0	26%
Task 4	2 SS	10	15	18	23	18	20	38	26	44	6	0	13%
Task 5	4 FS		13	15	17	15	38	53	44	59	6	0	13%
Task 6	3,5 FS		30	38	45	38	56	94	59	97	3	0	39%
Task 7	6 FS	-5	20	25	30	25	89	114	92	117	3	0	19%
Task 8	6 FS	5	10	15	20	15	99	114	102	117	3	0	20%
Task 9	6 SS	15	11	18	22	18	71	89	99	117	28	0	0%
Task 10	7,8,9 FS		23	30	45	30	114	144	117	147	3	0	39%
Task 11	10 FS	5	22	28	39	28	149	177	152	180	3	0	39%
Task 12	Start FS		120	140	180	140	0	140	0	140	0	1	61%
Task 13	12 FS	-5	13	18	22	18	135	153	135	153	0	1	61%
Task 14	13 SS	10	15	20	25	20	145	165	145	165	0	1	61%
Task 15	14 FS		10	15	20	15	165	180	165	180	0	1	61%
Task 16	11, 15 FS		30	33	60	33	180	213	180	213	0	1	100%
Task 17	16 FS		5	8	11	8	213	221	213	221	0	1	100%
Task 18	17 FS		10	15	25	15	221	236	223	238	2	0	52%
Task 19	17 FS		13	17	19	17	221	238	221	238	0	1	48%
Task 20	18, 19 FS		20	25	45	25	238	263	238	263	0	1	100%

Luego vemos la ultima iteración y vemos que tienen datos que coinciden de una manera en la simialcion que queremos realizar.

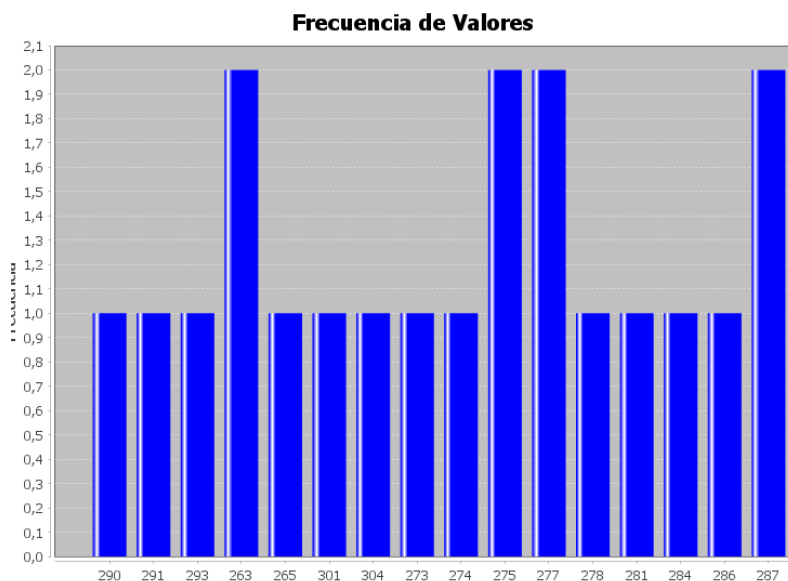
Iteración: 20

Tarea	Lead/Lag	Dur.Opt.	Dur.Esp.	Dur.Pes.	ValorSim.	EST	EFT	LST	LFT	Slack	Ruta Critica	Probabilidad
Task 0	0	0	0	0	0	0	0	3	3	3	No	0,00
Task 1	0	10	15	20	18	0	18	0	18	0	Sí	0,00
Task 2	-5	15	20	22	17	13	30	13	30	0	Sí	0,00
Task 3	0	21	26	30	27	30	57	30	57	0	Sí	35,00
Task 4	10	15	18	23	17	23	40	25	42	2	No	75,00
Task 5	0	13	15	17	15	40	55	42	57	2	No	71,00
Task 6	0	30	38	45	39	57	96	57	96	0	Sí	0,00
Task 7	-5	20	25	30	26	91	117	91	117	0	Sí	92,00
Task 8	5	10	15	20	11	101	112	106	117	5	No	55,00
Task 9	15	11	18	22	15	72	87	102	117	30	No	80,00
Task 10	0	23	30	45	33	117	150	117	150	0	Sí	0,00
Task 11	5	22	28	39	36	155	191	155	191	0	Sí	52,00
Task 12	0	120	140	180	147	0	147	3	150	3	No	0,00
Task 13	-5	13	18	22	19	142	161	145	164	3	No	0,00
Task 14	10	15	20	25	23	152	175	155	178	3	No	8,00
Task 15	0	10	15	20	13	175	188	178	191	3	No	8,00
Task 16	0	30	33	60	50	191	241	191	241	0	Sí	5,00
Task 17	0	5	8	11	7	241	248	241	248	0	Sí	29,00
Task 18	0	10	15	25	20	248	268	248	268	0	Sí	61,00
Task 19	0	13	17	19	18	248	266	250	268	2	No	44,00
Task 20	0	20	25	45	33	268	301	268	301	0	Sí	47,00

Una vez teniendo los datos tenemos que analizar las graficas que muestran lo siguiente.



Esta grafica muestra la probabilidad de que la tarea se convierta en una tarea critica en in proceltaje del 1 al 100.



La grafica siguiente muestra como el tiempo de inicio mas tardio se repite a lo largo de las simulaciones para ver cual es mas probable que salga.

Conclusión:

El análisis de la **ruta crítica** revela que las tareas con **probabilidad alta de estar en la ruta crítica** son aquellas que tienen un mayor impacto en la fecha de finalización del proyecto. Estas tareas tienen **holgura cero**, lo que significa que cualquier retraso en ellas afectará directamente la finalización del proyecto. Además, el uso de herramientas como **Crystal Ball** permite evaluar la incertidumbre y determinar con precisión qué tareas son críticas y cuáles tienen más flexibilidad.

Decisión:

Dado que las tareas en la **ruta crítica** son las que más impactan el éxito del proyecto, se debe priorizar la asignación de recursos (como personal, presupuesto y tiempo) hacia estas tareas para minimizar retrasos.

Ejercicio 3:

Cadena de suministro de gasolina

Descripción:

En este ejemplo, determinamos la cantidad óptima de gasolina que se debe transportar entre diferentes niveles de una cadena de suministro de gasolina con el objetivo de **minimizar el costo total**, que incluye los costos de transporte y los costos de mantenimiento de inventario en varios puntos de la cadena de suministro. Además, buscamos **minimizar la falta de existencias** en varios puntos de venta al por menor. La complejidad del problema surge debido a la **producción estocástica** en la refinería y la **demanda estocástica** en los puntos de venta al por menor.

```
import javax.swing.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;
import org.apache.commons.math3.distribution.NormalDistribution;
import org.apache.commons.math3.distribution.LogNormalDistribution;
```

Primero exportamos las distribuciones que se utilizarán que en este caso es la distribución normal y la normal logarítmica.

Refinación de gas:

```
//Refineria
private double refinaria(double demandaSD1, double demandaSD2, double costoTenencia, int i){
    double invInicial = 200;
    //Distribución normal
    double produccion = 0;
    if(i == 0){
        produccion = 2000;
    }else{
        NormalDistribution normalDistribution = new NormalDistribution(2000, 450);
        produccion = normalDistribution.sample();
    }
    double demandaCumplida = demandaSD1 + demandaSD2;
    double invFinal = invInicial + produccion - demandaCumplida;
    double costoInv = invFinal * costoTenencia;
    return costoInv;
}
```

En la parte de refinería se encarga de recibir la demanda del depósito 1 y 2 el costo de tener en inventario y la interacción actual, luego procede a calcular la producción total y si es la primera iteración será 2000 y si no se calculará con la distribución normal y se procede a calcular los datos siguientes.

Almacenamiento de gas:

```
//Deposito
private List<Double> deposito(List<Double> SD1, List<Double> SD2, double costoTenencia, double costo1, double costo2) {
    double invInicialSD1 = 50;
    double invInicialSD2 = 100;
    double suministroRecibidoSD1 = 165;
    double suministroRecibidoSD2 = 1965;
    double costoRefineria = refineria(suministroRecibidoSD1, suministroRecibidoSD2, costoTenencia, i);
    double demandaCumplidaSD1 = 0;
    for (Double num : SD1) {
        demandaCumplidaSD1 += num;
    }
    double demandaCumplidaSD2 = 0;
    for (Double num : SD2) {
        demandaCumplidaSD2 += num;
    }
    double invFinalSD1 = invInicialSD1 + suministroRecibidoSD1 - demandaCumplidaSD1;
    double invFinalSD2 = invInicialSD2 + suministroRecibidoSD2 - demandaCumplidaSD2;
    double costoInvSD1 = 0;
    double costoInvSD2 = 0;
    if(invFinalSD1 > 0){
        costoInvSD1 = invFinalSD1 * costoTenencia;
    }
    if (invFinalSD2 > 0){
        costoInvSD2 = invFinalSD2 * costoTenencia;
    }
    double costoTransporteSD1 = suministroRecibidoSD1 * costo1;
    double costoTransporteSD2 = suministroRecibidoSD2 * costo2;
    double costoInv = costoInvSD1 + costoInvSD2 + costoRefineria;
    double costoTransporte = costoTransporteSD1 + costoTransporteSD2;
    List<Double> totales = new ArrayList<>(List.of(costoInv, costoTransporte));
    return totales;
}
```

En la parte de almacenamiento se encarga de recibir la demanda del deposito 1 y 2 el costo de tenencia y el costo de transporte y también la iteración, luego procedemos a calcular los datos necesarios del almacenamiento.

Puntos de venta:

```
private void simularEjercicioA(java.awt.event.ActionEvent evt) {
    try {
        // Obtener valor del campo de texto
        double costoTenencia = Double.parseDouble(costoTenenciaInventario.getText());
        int iteraciones = Integer.parseInt(numeroIteraciones.getText());
        textAreaA.append("Costo de tenencia: " + costoTenencia + "\n");
        textAreaA.append("Número de iteraciones: " + iteraciones + "\n");
        //Costo de transporte a:
        double costoTransporteDeposito1 = 15;
        double costoTransporteDeposito2 = 12.5;
        //Costo de transporte a:
        double costoTransporteVentaSD11 = 6.5;
        double costoTransporteVentaSD12 = 7.5;
        double costoTransporteVentaSD13 = 9;
        double costoTransporteVentaSD21 = 9;
        double costoTransporteVentaSD22 = 8;
        double costoTransporteVentaSD23 = 7;
        List<Double> costosTotales1 = new ArrayList<>();
        List<Double> faltanteEnPeorCasos1 = new ArrayList<>();
    }
}
```

En el punto de venta se inicializan todos los datos necesarios para realizar el ejercicio y se pasan los costos de transporte de cada destino y también listas donde se almacenarán los datos a analizar.


```

int i = 0;
while(i < iteraciones){
    textAreaA.append("Iteración: " + (i+1) + "\n");
    //Punto de venta
    double invInicialR01 = 120;
    double invInicialR02 = 180;
    double invInicialR03 = 80;
    List<Double> SD1 = new ArrayList<>(List.of(80.0, 100.0, 45.0));
    List<Double> SD2 = new ArrayList<>(List.of(200.0, 310.0, 985.0));
    double suministroRecividoR01 = SD1.get(0) + SD2.get(0);
    double suministroRecividoR02 = SD1.get(1) + SD2.get(1);
    double suministroRecividoR03 = SD1.get(2) + SD2.get(2);
    //Distribucion logaritmica normal
    List<Double> demandaCumplida = new ArrayList<>();
    if(i == 0){
        //List<Double> demandaCumplida = new ArrayList<>(List.of(400.0, 500.0, 650.0));
        demandaCumplida.add(400.0);
        demandaCumplida.add(500.0);
        demandaCumplida.add(650.0);
    }else{
        LogNormalDistribution logNormalDistribution1 = new LogNormalDistribution(Math.log(400), 50);
        double demanda1 = logNormalDistribution1.sample();
        LogNormalDistribution logNormalDistribution2 = new LogNormalDistribution(Math.log(500), 75);
        double demanda2 = logNormalDistribution2.sample();
        LogNormalDistribution logNormalDistribution3 = new LogNormalDistribution(Math.log(650), 100);
        double demanda3 = logNormalDistribution3.sample();
        //List<Double> demandaCumplida = new ArrayList<>(List.of(demanda1, demanda2, demanda3));
        demandaCumplida.add(demanda1);
        demandaCumplida.add(demanda2);
        demandaCumplida.add(demanda3);
    }
}

```

Luego se inicia un bucle while primero procedemos a calcular el suministro recibido y el inventario inicial y si es la primera iteración entonces los datos se pasan manualmente y si no los datos se calculan utilizando la distribución logarítmica normal.

```

double costoTransporteR01 = (SD1.get(0) * costoTransporteVentaSD11) + (SD2.get(0) * costoTransporteVen
double costoTransporteR02 = (SD1.get(1) * costoTransporteVentaSD12) + (SD2.get(1) * costoTransporteVen
double costoTransporteR03 = (SD1.get(2) * costoTransporteVentaSD13) + (SD2.get(2) * costoTransporteVen

double desavastecimientoR01 = demandaCumplida.get(0) - invInicialR01 - suministroRecividoR01;
double desavastecimientoR02 = demandaCumplida.get(1) - invInicialR02 - suministroRecividoR02;
double desavastecimientoR03 = demandaCumplida.get(2) - invInicialR03 - suministroRecividoR03;
List<Double> totales = deposito(SD1, SD2, costoTenencia, costoTransporteDeposito1, costoTransporteDepo
double costoTrans = costoTransporteR01 + costoTransporteR02 + costoTransporteR03;
double costoInv = costoInvR01 + costoInvR02 + costoInvR03;
double desavastecimiento = Math.max(desavastecimientoR01, Math.max(desavastecimientoR02, desavastecimi
double costoInventarios = costoInv + totales.get(0);
double costoTransporter = costoTrans + totales.get(1);
double costoTotal = costoInventarios + costoTransporter;
textAreaA.append("Costo de inventarios: " + costoInventarios + "\n");
textAreaA.append("Costo de transporte: " + costoTransporter + "\n");
textAreaA.append("Costo total: " + costoTotal + "\n");
textAreaA.append("Desavastecimiento en el peor de los casos: " + desavastecimiento + "\n");
textAreaA.append("-----\n");
costosTotales1.add(costoTotal);
faltanteEnPeorCasos1.add(desavastecimiento);
i++;
}
SwingUtilities.invokeLater(() -> {
    Grafico chart = new Grafico(costosTotales1, calcularMedia(costosTotales1));
    chart.setSize(800, 600);
    chart.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    chart.setVisible(true);
});
SwingUtilities.invokeLater(() -> {
    GraficoGalonesFaltantes grafico = new GraficoGalonesFaltantes(faltanteEnPeorCasos1);
    grafico.setVisible(true);
});

```

Luego procedemos a calcular los demás datos y mandamos a llamar a los otros métodos para realizar los cálculos faltantes luego de eso procedemos a imprimir los datos que nos interesan que son el costo total y el desabastecimiento en el peor de los casos.

Luego los guardamos en las listas respectiva para mandarlos a imprimir.

Ejecución:

Introducir Datos

Costo de tenencia de inventario: 0.2

Número de iteraciones: 20

Simular

Inciso A

Inciso B

Inciso C

Costo de tenencia: 0.2

Número de iteraciones: 20

Iteración: 1

Costo de inventarios: 238.0

Costo de transporte: 39887.5

Costo total: 40125.5

Desavastecimiento en el peor de los casos: 0.0

Iteración: 2

Costo de inventarios: 279.56410996441355

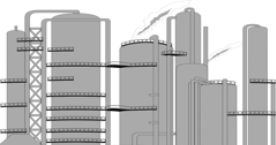
Costo de transporte: 39887.5

Costo total: 40167.06410996441

Desavastecimiento en el peor de los casos: 4.985443804229814E29

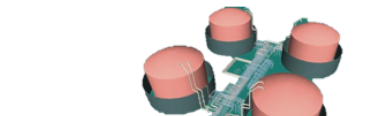
Vemos que la simulación si arroja los datos que pasa cristal ball por lo que podemos decir que el modelo es valido para realiza la simulación.

ENTRADAS DEL MODELO	
Costo de tenencia inv.	\$0,20



Refineria	
Inv inicial	200
Producción	2000
Demanda cumplida	2130
Inv final	70
costo de Inv	\$14,00


Costos de transporte a:	
Depósito de suministros 1	\$15,00
Depósito de suministros 2	\$12,50
(dollars/gallon)	



Depósitos suministros	SD1	SD2
Inv inicial	50	100
Suministro recibido	165	1965
Demanda cumplida	380	1495
Inv final	-165	570
Costo de Inv	\$0,00	\$114,00
Costo transporte	\$2.475,00	\$24.562,50

Costos de transporte a:	
Al punto de venta 1	\$6,50 \$9,00
Al punto de venta 2	\$7,50 \$8,00

Resultados del modelo	
Costos de inventario:	\$238,00
Costos de transporte:	\$39.887,50
Costos totales:	\$40.125,50
Desavastecimiento en el peor de los casos:	0



Puntos de venta	RO1	RO2	RO3
Inv inicial	120	180	80
Suministro de SD1	80	100	45
Suministro de SD2	200	310	985
Suministro recibido	280	410	1030
Demanda cumplida	400	500	650
Inv final	0	90	460
Costo Inv	\$0,00	\$18,00	\$92,00
Costo transporte	\$2.320,00	\$3.230,00	\$7.300,00
Desavastecimiento	0	-90	-460

Luego viendo las demás iteraciones podemos ver que si son datos parecido a los que deseamos analizar por lo que la simulación cumple con lo que se quiere simular.

Iteración: 18

Costo de inventarios: 573.0126268037991

Costo de transporte: 39887.5

Costo total: 40460.5126268038

Desavastecimiento en el peor de los casos: -399.999999501479

Iteración: 19

Costo de inventarios: 103.65446813370163

Costo de transporte: 39887.5

Costo total: 39991.1544681337

Desavastecimiento en el peor de los casos: 1.9893507001110544E37

Iteración: 20

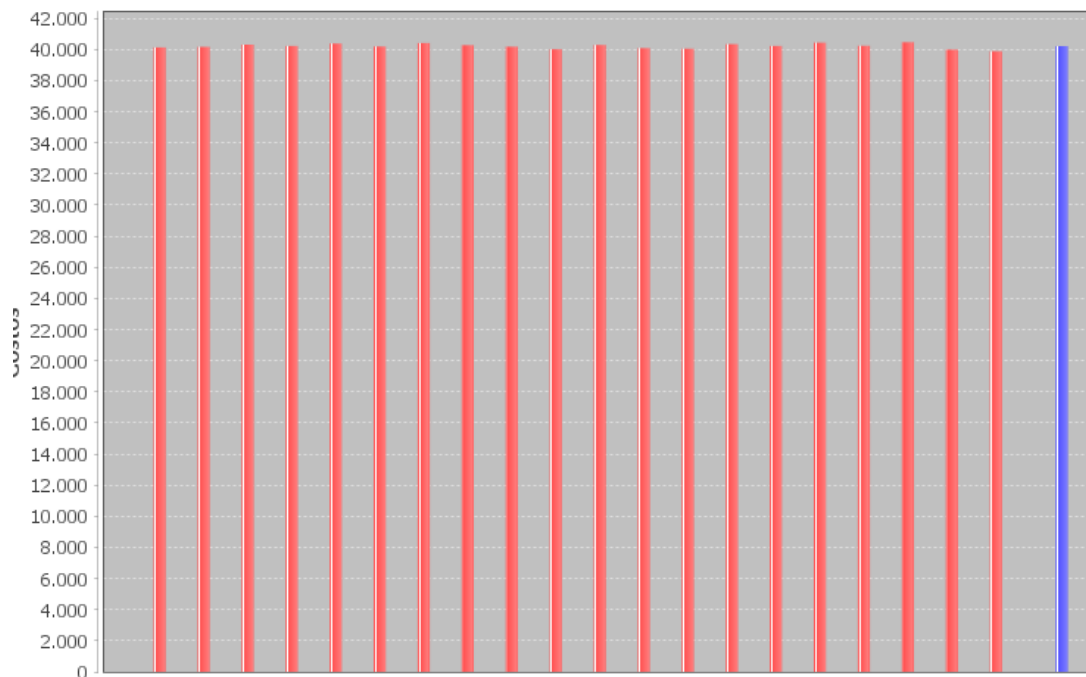
Costo de inventarios: -5.689906313818852

Costo de transporte: 39887.5

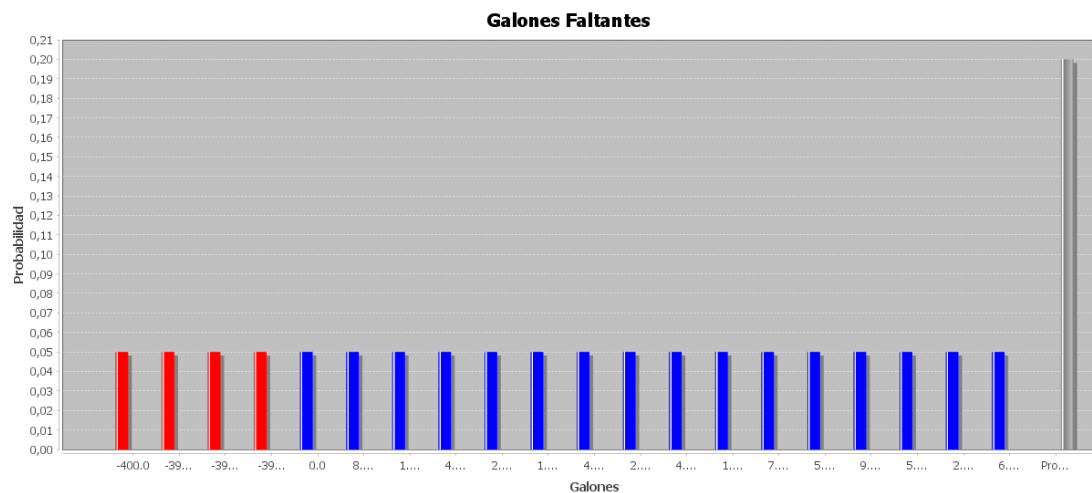
Costo total: 39881.81009368618

Desavastecimiento en el peor de los casos: 5.709847634922664E55

En las respectivas graficas podemos ver qué.



Los diferentes costos y la media necesaria de dichos costos pintada de un color diferente que en este caso es el color azul.



El siguiente grafico muestra los calones faltantes y la probabilidad de que haya un desabastecimiento en el peor de los casos.

Conclusión:

El análisis de la cadena de suministro de gasolina utilizando simulaciones estocásticas y optimización con OptQuest permitió encontrar una estrategia eficiente para minimizar los **costos totales** de operación, que incluyen tanto los costos de transporte como los de mantenimiento de inventario. Además, se logró reducir el riesgo de **agotamiento de existencias** en los puntos de venta al por menor (RO), lo cual es crucial para evitar pérdidas de ventas y mantener la satisfacción del cliente.

Decisión:

La empresa debe implementar la estrategia de abastecimiento óptima sugerida por OptQuest, transportando cantidades específicas de gasolina a los depósitos de suministro (SD) y los puntos de venta (RO) para **minimizar los costos** y al mismo tiempo mantener el riesgo de agotamientos de stock bajo control.