

## Curso de Java a distancia

### Clase 15: Estructuras dinámicas: Listas tipo Pila - Problema de aplicación

Hasta ahora hemos visto como desarrollar los algoritmos para administrar una lista tipo Pila, hemos visto que hay bastante complejidad en el manejo de punteros pero todo esto acarrea ventajas en la solución de problemas que requieren una estructura de tipo Pila.

#### Planteo del problema:

Este práctico tiene por objetivo mostrar la importancia de las pilas en las Ciencias de la Computación y más precisamente en la programación de software de bajo nivel.

Todo compilador o intérprete de un lenguaje tiene un módulo dedicado a analizar si una expresión está correctamente codificada, es decir que los paréntesis estén abiertos y cerrados en un orden lógico y bien balanceados.

Se debe desarrollar una clase que tenga las siguientes responsabilidades (clase Formula):

- Ingresar una fórmula que contenga paréntesis, corchetes y llaves.
- Validar que los ( ) [] y {} estén correctamente balanceados.

Para la solución de este problema la clase formula tendrá un atributo de la clase Pila.

Veamos como nos puede ayudar el empleo de una pila para solucionar este problema. Primero cargaremos la fórmula en un JTextField.

Ejemplo de fórmula:  $(2+[3-12]*\{8/3\})$

El algoritmo de validación es el siguiente:

Analizamos caracter a caracter la presencia de los paréntesis, corchetes y llaves.

Si vienen símbolos de apertura los almacenamos en la pila.

Si vienen símbolos de cerrado extraemos de la pila y verificamos si está el mismo símbolo pero de apertura: en caso negativo podemos inferir que la fórmula no está correctamente balanceada.

Si al finalizar el análisis del último caracter de la fórmula la pila está vacía podemos concluir que está correctamente balanceada.

Ejemplos de fórmulas no balanceadas:

$\}(2+[3-12]*\{8/3\})$

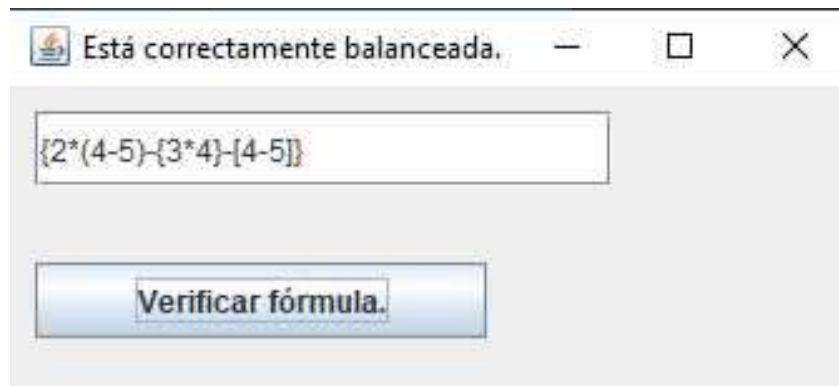
Incorrecta: llega una } de cerrado y la pila está vacía.

{[2+4]}

Incorrecta: llega una llave } y en el tope de la pila hay un corchete [.

{[2+4]

Incorrecta: al finalizar el análisis del último caracter en la pila queda pendiente una llave {.



**Programa:**

```
public class Pila {  
  
    class Nodo {  
        char simbolo;  
        Nodo sig;  
    }  
  
    private Nodo raiz;  
  
    Pila () {  
        raiz=null;  
    }  
  
    public void insertar(char x) {  
        Nodo nuevo;  
        nuevo = new Nodo();  
        nuevo.simbolo = x;  
        if (raiz==null)  
        {  
            nuevo.sig = null;  
            raiz = nuevo;  
        }  
        else  
        {  
            nuevo.sig = raiz;  
            raiz = nuevo;  
        }  
    }  
  
    public char extraer ()  
    {
```

```

    if (raiz!=null)
    {
        char informacion = raiz.simbolo;
        raiz = raiz.sig;
        return informacion;
    }
    else
    {
        return Character.MAX_VALUE;
    }
}

public boolean vacia() {
    if (raiz==null) {
        return true;
    } else {
        return false;
    }
}
}

import javax.swing.*.*;
import java.awt.event.*;
public class Formula extends JFrame implements ActionListener {
    private JTextField tf1;
    private JButton boton1;
    public Formula() {
        setLayout(null);
        tf1=new JTextField("{2*(4-5)-{3*4}-[4-5]}");
        tf1.setBounds(10,10,230,30);
        add(tf1);
        boton1=new JButton("Verificar fórmula.");
        boton1.setBounds(10,70,180,30);
        add(boton1);
        boton1.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==boton1) {
            if (balanceada()) {
                setTitle("Está correctamente balanceada.");
            } else {
                setTitle("No está correctamente balanceada.");
            }
        }
    }
}

```

```

    }
}

public boolean balanceada() {
    Pila pila1;
    pila1 = new Pila ();
    String cadena=tf1.getText();
    for (int f = 0 ; f < cadena.length() ; f++)
    {
        if (cadena.charAt(f) == '(' || cadena.charAt(f) == '[' || cadena.charAt(f) == '{')
        {
            pila1.insertar(cadena.charAt(f));
        } else {
            if (cadena.charAt(f)==')') {
                if (pila1.extraer()!='(') {
                    return false;
                }
            } else {
                if (cadena.charAt(f)==']') {
                    if (pila1.extraer()!='[') {
                        return false;
                    }
                } else {
                    if (cadena.charAt(f)=='}') {
                        if (pila1.extraer()!='{') {
                            return false;
                        }
                    }
                }
            }
        }
    }
    if (pila1.vacia()) {
        return true;
    } else {
        return false;
    }
}

public static void main(String[] ar) {
    Formula formula1=new Formula();
    formula1.setBounds(0,0,360,160);
    formula1.setVisible(true);
}
}

```

Primero declaramos y definimos la clase Pila. Almacenamos en cada nodo un caracter y llamamos al campo de información símbolo.

No es necesario implementar los métodos imprimir, cantidad, etc. Porque no se requieren para este problema.

La clase Formula tiene como atributos:

```
private JTextField tf1;  
private JButton boton1;
```

En el constructor creamos los dos objetos y los ubicamos:

```
setLayout(null);  
tf1=new JTextField("{2*(4-5)-{3*4}-[4-5]}");  
tf1.setBounds(10,10,230,30);  
add(tf1);  
boton1=new JButton("Verificar fórmula.");  
boton1.setBounds(10,70,180,30);  
add(boton1);  
boton1.addActionListener(this);
```

En el método actionPerformed llamamos al método balanceada que debe retornar si la fórmula están correctos los parentesis, corchetes y llaves:

```
if (e.getSource()==boton1) {  
    if (balanceada()) {  
        setTitle("Está correctamente balanceada.");  
    } else {  
        setTitle("No está correctamente balanceada.");  
    }  
}
```

El método más importante es el balanceada.

En este analizamos la fórmula para verificar si está correctamente balanceada.

En este método es donde está gran parte del algoritmo de este problema. Retorna true en caso de ser correcta y false en caso contrario.

Definimos una pila y extraemos el contenido del JTextField:

```
Pila pila1;  
pila1 = new Pila ();  
String cadena=tf1.getText();
```

El for se repite tantas veces como caracteres tenga el JTextField.

Se deben procesar sólo los símbolos ( [ { y ] } ).

Si el símbolo es un ( [ { de apertura procedemos a cargarlo en la pila:

```
if (cadena.charAt(f) == '(' || cadena.charAt(f) == '[' || cadena.charAt(f) == '{') {  
    pila1.insertar(cadena.charAt(f));
```

En caso de ser un ) cerrado debemos extraer un carácter de la pila y verificar si no coincide con el paréntesis de apertura ( la fórmula está incorrecta:

```

        if (cadena.charAt(f)=='') {
            if (pila1.extraer()!='(') {
                return false;
            }
        }
    }

```

El mismo proceso es para los símbolos `]`.

Al finalizar el análisis de toda la cadena si la pila está vacía podemos afirmar que la fórmula está correctamente balanceada, en caso contrario quiere decir que faltan símbolos de cerrado y es incorrecta:

```

    if (pila1.vacia()) {
        return true;
    } else {
        return false;
    }

```

Es importante entender que la clase `Formula` utiliza un objeto de la clase `Pila` para resolver el algoritmo de verificar el balanceo de la fórmula, pero no accede directamente a los nodos de la lista.

## Estructuras dinámicas: Listas tipo Cola

Una lista se comporta como una cola si las inserciones las hacemos al final y las extracciones las hacemos por el frente de la lista. También se las llama listas FIFO (First In First Out - primero en entrar primero en salir)

Confeccionaremos un programa que permita administrar una lista tipo cola. Desarrollaremos los métodos de insertar, extraer, vacía e imprimir.

### Programa:

```

public class Cola {

    class Nodo {
        int info;
        Nodo sig;
    }

    private Nodo raiz,fondo;

    Cola() {
        raiz=null;
        fondo=null;
    }

    boolean vacia (){
        if (raiz == null)
            return true;
    }

```

```

        else
            return false;
    }

    void insertar (int info)
    {
        Nodo nuevo;
        nuevo = new Nodo ();
        nuevo.info = info;
        nuevo.sig = null;
        if (vacía ()) {
            raiz = nuevo;
            fondo = nuevo;
        } else {
            fondo.sig = nuevo;
            fondo = nuevo;
        }
    }

    int extraer ()
    {
        if (!vacía ())
        {
            int informacion = raiz.info;
            if (raiz == fondo){
                raiz = null;
                fondo = null;
            } else {
                raiz = raiz.sig;
            }
            return informacion;
        } else
            return Integer.MAX_VALUE;
    }

    public void imprimir() {
        Nodo reco=raiz;
        System.out.println("Listado de todos los elementos de la cola.");
        while (reco!=null) {
            System.out.print(reco.info+"-");
            reco=reco.sig;
        }
        System.out.println();
    }

    public static void main(String[] ar) {
        Cola cola1=new Cola();
        cola1.insertar(5);
    }

```

```

        cola1.insertar(10);
        cola1.insertar(50);
        cola1.imprimir();
        System.out.println("Extraemos uno de la cola:"+cola1.extraer());
        cola1.imprimir();
    }
}

```

La declaración del nodo es igual a la clase Pila. Luego definimos dos punteros externos:

```
private Nodo raiz,fondo;
```

raíz apunta al principio de la lista y fondo al final de la lista. Utilizar dos punteros tiene como ventaja que cada vez que tengamos que insertar un nodo al final de la lista no tengamos que recorrerla. Por supuesto que es perfectamente válido implementar una cola con un único puntero externo a la lista.

En el constructor inicializamos a los dos punteros en null (Realmente esto es opcional ya que los atributos de una clase en java se inicializan automáticamente con null):

```

Cola() {
    raiz=null;
    fondo=null;
}

```

El método vacía retorna true si la lista no tiene nodos y false en caso contrario:

```

boolean vacia (){
    if (raiz == null)
        return true;
    else
        return false;
}

```

En la inserción luego de crear el nodo tenemos dos posibilidades: que la cola esté vacía, en cuyo caso los dos punteros externos a la lista deben apuntar al nodo creado, o que haya nodos en la lista.

```

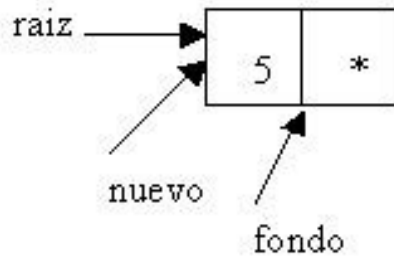
Nodo nuevo;
nuevo = new Nodo ();
nuevo.info = info;
nuevo.sig = null;
if (vacía ()) {
    raiz = nuevo;
    fondo = nuevo;
} else {
    fondo.sig = nuevo;
    fondo = nuevo;
}

```

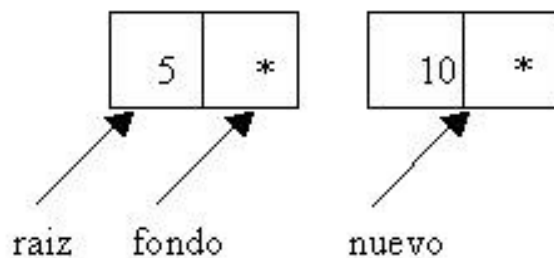
Recordemos que definimos un puntero llamado nuevo, luego creamos el nodo con el operador new y cargamos los dos campos, el de información con lo que llega en el parámetro y el puntero con null ya que se insertará al final de la lista, es decir no hay otro después de este.



Si la lista está vacía:

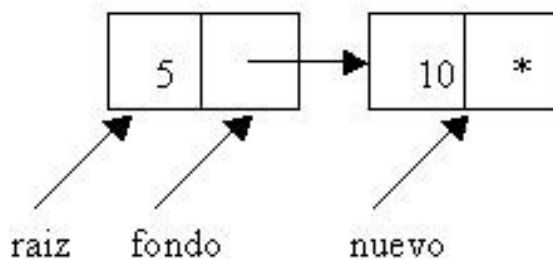


En caso de no estar vacía:



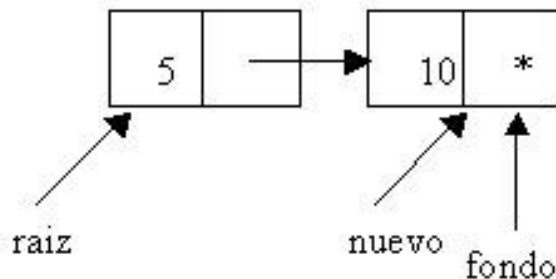
Debemos enlazar el puntero sig del último nodo con el nodo recién creado:

`fondo.sig = nuevo;`



Y por último el puntero externo fondo debe apuntar al nodo apuntado por nuevo:

`fondo = nuevo;`



Con esto ya tenemos correctamente enlazados los nodos en la lista tipo cola. Recordar que el puntero nuevo desaparece cuando se sale del método insertar, pero el nodo creado no se pierde porque queda enlazado en la lista.

El funcionamiento del método extraer es similar al de la pila:

```

int extraer ()
{
    if (!vacía ())
    {
        int informacion = raiz.info;
        if (raiz == fondo){
            raiz = null;
            fondo = null;
        } else {
            raiz = raiz.sig;
        }
        return informacion;
    } else
        return Integer.MAX_VALUE;
}

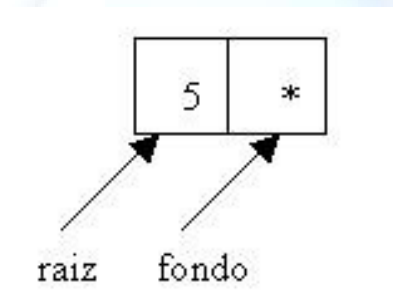
```

Si la lista no está vacía guardamos en una variable local la información del primer nodo:

```
int informacion = raiz.info;
```

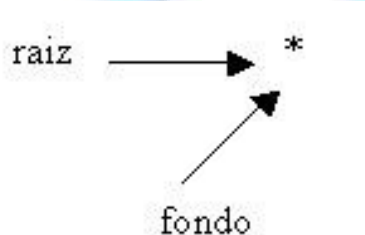
Para saber si hay un solo nodo verificamos si los dos punteros raiz y fondo apuntan a la misma dirección de memoria:

```
if (raiz == fondo){
```

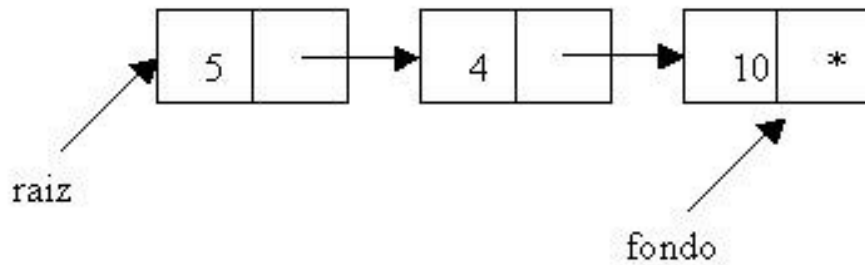


Luego hacemos:

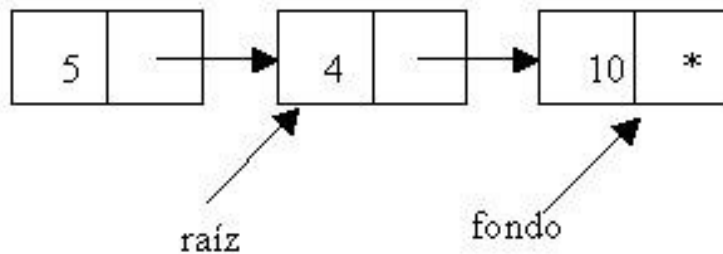
```
raiz = null;
fondo = null;
```



En caso de haber 2 o más nodos debemos avanzar el puntero raiz al siguiente nodo:



raiz = raiz.sig;



Ya tenemos la lista correctamente enlazada (raiz apunta al primer nodo y fondo continúa apuntando al último nodo)

## Estructuras dinámicas: Listas tipo Cola - Problemas de aplicación

### Planteo del problema:

Este práctico tiene por objetivo mostrar la importancia de las colas en las Ciencias de la Computación y más precisamente en las simulaciones.

Las simulaciones permiten analizar situaciones de la realidad sin la necesidad de ejecutarlas realmente. Tiene el beneficio que su costo es muy inferior a hacer pruebas en la realidad.

Desarrollar un programa para la simulación de un cajero automático.

Se cuenta con la siguiente información:

- Llegan clientes a la puerta del cajero cada 2 ó 3 minutos.
- Cada cliente tarda entre 2 y 4 minutos para ser atendido.

Obtener la siguiente información:

- 1 - Cantidad de clientes que se atienden en 10 horas.
- 2 - Cantidad de clientes que hay en cola después de 10 horas.
- 3 - Hora de llegada del primer cliente que no es atendido luego de 10 horas (es decir la persona que está primera en la cola cuando se cumplen 10 horas)

### Programa:

```
public class Cola {
```

```
class Nodo {  
    int info;  
    Nodo sig;  
}
```

```
Nodo raiz,fondo;
```

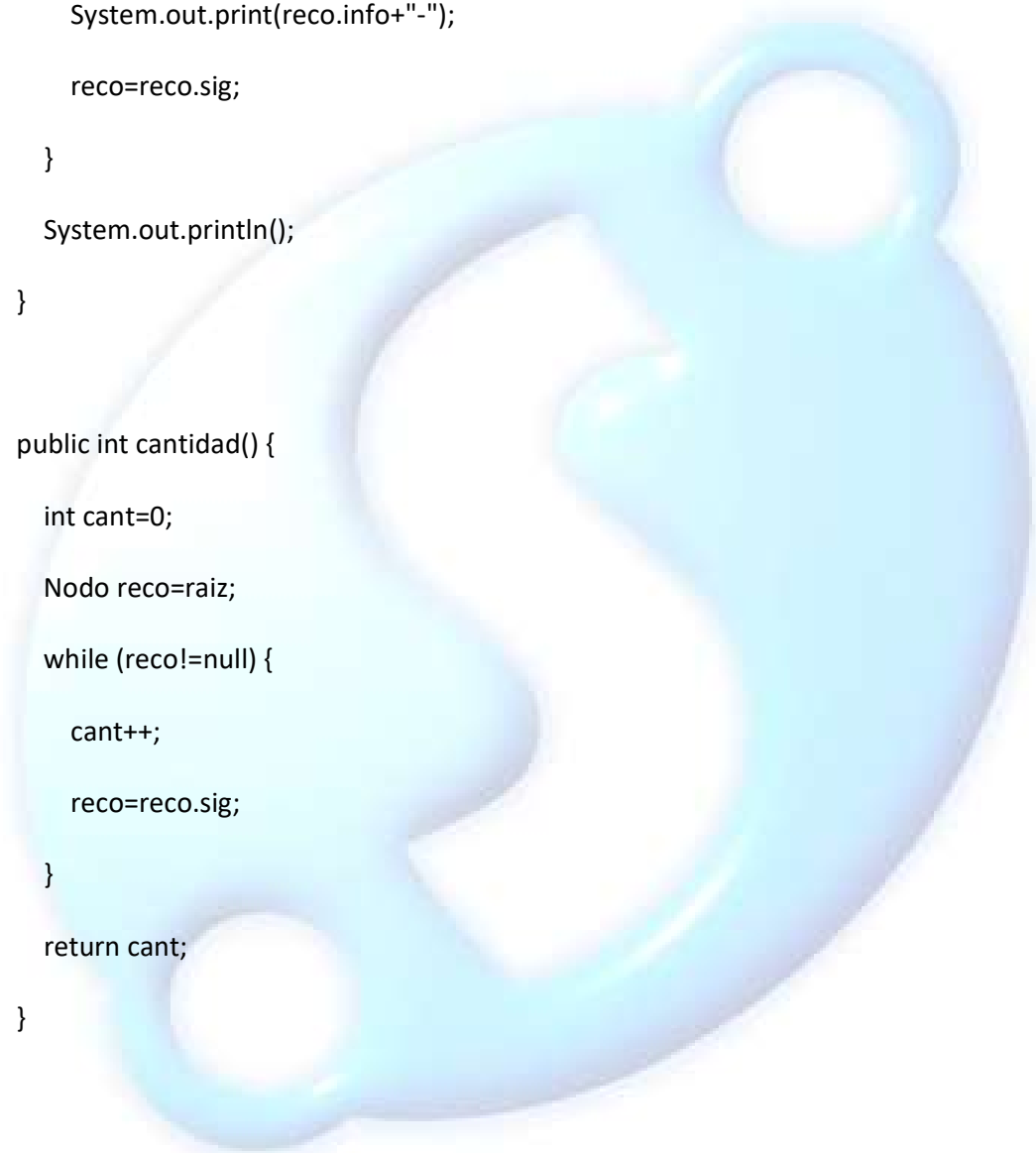
```
Cola() {  
    raiz=null;  
    fondo=null;  
}
```

```
boolean vacia (){  
    if (raiz == null)  
        return true;  
    else  
        return false;  
}
```

```
void insertar (int info)  
{  
    Nodo nuevo;  
    nuevo = new Nodo ();  
    nuevo.info = info;  
    nuevo.sig = null;
```

```
if (vacía ()) {  
    raíz = nuevo;  
    fondo = nuevo;  
} else {  
    fondo.sig = nuevo;  
    fondo = nuevo;  
}  
}  
  
int extraer ()  
{  
    if (!vacía ())  
    {  
        int informacion = raíz.info;  
        if (raíz == fondo){  
            raíz = null;  
            fondo = null;  
        } else {  
            raíz = raíz.sig;  
        }  
        return informacion;  
    } else  
        return Integer.MAX_VALUE;  
}
```

```
public void imprimir() {  
    Nodo reco=raiz;  
    System.out.println("Listado de todos los elementos de la cola.");  
    while (reco!=null) {  
        System.out.print(reco.info+"-");  
        reco=reco.sig;  
    }  
    System.out.println();  
}  
  
public int cantidad() {  
    int cant=0;  
    Nodo reco=raiz;  
    while (reco!=null) {  
        cant++;  
        reco=reco.sig;  
    }  
    return cant;  
}  
}
```



```
import javax.swing.*;

import java.awt.event.*;

public class Cajero extends JFrame implements ActionListener{

    private JLabel l1,l2,l3;

    private JButton boton1;

    public Cajero() {

        setLayout(null);

        boton1=new JButton("Activar Simulación");

        boton1.setBounds(10,10,180,30);

        add(boton1);

        boton1.addActionListener(this);

        l1=new JLabel("Atendidos:");

        l1.setBounds(10,50,300,30);

        add(l1);

        l2=new JLabel("En cola:");

        l2.setBounds(10,90,300,30);

        add(l2);

        l3=new JLabel("Minuto de llegada:");

        l3.setBounds(10,130,400,30);

        add(l3);

    }

    public void actionPerformed(ActionEvent e) {

        if (e.getSource()==boton1) {

            simulacion();

        }

    }

}
```

```
}  
}
```

```
public void simulacion () {  
  
    int estado = 0;  
  
    int llegada = 2 + (int) (Math.random () * 2);  
  
    int salida = -1;  
  
    int cantAtendidas = 0;  
  
    Cola cola = new Cola ();  
  
    for (int minuto = 0 ; minuto < 600 ; minuto++) {  
  
        if (llegada == minuto)  
        {  
  
            if (estado==0) {  
  
                estado=1;  
  
                salida=minuto+2+(int)(Math.random()*3);  
  
            } else {  
  
                cola.insertar(minuto);  
  
            }  
  
            llegada=minuto+2+(int)(Math.random()*2);  
  
        }  
  
        if (salida == minuto)  
  
        {  
  
            estado=0;  
  
            cantAtendidas++;  
  
            if (!cola.vacia()) {
```



```

        cola.extraer();

        estado=1;

        salida=minuto+2+(int)(Math.random()*3);

    }

}

l1.setText("Atendidos:"+String.valueOf(cantAtendidas));

l2.setText("En cola"+String.valueOf(cola.cantidad()));

l3.setText("Minuto llegada:"+String.valueOf(cola.extraer()));

}

public static void main(String[] ar) {

    Cajero cajero1=new Cajero();

    cajero1.setBounds(0,0,340,250);

    cajero1.setVisible(true);

}

}

```

La clase Cola colabora con la clase Cajero. En la clase cola debemos definir como mínimo los métodos de insertar, extraer, vacía y cantidad.

La clase Cajero define tres objetos de la clase JLabel para mostrar los resultados de la simulación.

El método más importante es el de simulacion, veamos las distintas partes de dicho método:

```

int estado = 0;
int llegada = 2 + (int) (Math.random () * 2);
int salida = -1;
int cantAtendidas = 0;
Cola cola = new Cola ();

```

La variable estado almacena un cero si el cajero está libre y un uno cuando está ocupado.  
La variable llegada almacena en que minuto llegará el próximo cliente (debemos generar un valor entre 2 y 3)

La variable salida almacenará en que minuto terminará el cliente de ser atendido (como al principio el cajero está vacío inicializamos esta variable con -1).

Llevamos un contador para saber la cantidad de personas atendidas (cantAtendidas)

Luego definimos un objeto de la clase Cola para poder almacenar las personas que llegan al cajero y se lo encuentran ocupado.

Disponemos un for que se repita 600 veces (600 minutos o lo que es lo mismo 10 horas)

```
for (int minuto = 0 ; minuto < 600 ; minuto++) {
```

Dentro del for hay dos if fundamentales que verifican que sucede cuando llega una persona o cuando una persona se retira:

```
    if (llegada == minuto)
    {
        .....
    }
    if (salida == minuto)
    {
        .....
    }
```

Cuando llega una persona al cajero primero verificamos si el cajero está desocupado:

```
    if (llegada == minuto)
    {
        if (estado==0) {
```

Si está desocupado lo ocupamos cambiando el valor de la variable estado y generando en que minuto esta persona dejará el cajero (un valor aleatorio entre 2 y 4 minutos):

```
        estado=1;
        salida=minuto+2+(int)(Math.random()*3);
```

Si el cajero está ocupado procedemos a cargar dicha persona en la cola (insertamos el minuto que llega):

```
        } else {
            cola.insertar(minuto);
        }
```

Luego generamos el próximo minuto que llegará otra persona:

```
        llegada=minuto+2+(int)(Math.random()*2);
```

El otro if importante es ver que sucede cuando sale la persona del cajero:

```
    if (salida == minuto) {
```

Si sale una persona del cajero cambiamos el valor de la variable estado, incrementamos en uno el contador cantAtendidas y si la cola no está vacía extraemos una persona, cambiamos a uno la variable estado y generamos en que minuto dejará esta persona el cajero:

```
        estado=0;
        cantAtendidas++;
        if (!cola.vacia()) {
            cola.extraer();
            estado=1;
            salida=minuto+2+(int)(Math.random()*3);
        }
    }
}
```

Fuera del for actualizamos las tres JLabel:

```
l1.setText("Atendidos:"+String.valueOf(cantAtendidas));
l2.setText("En cola"+String.valueOf(cola.cantidad()));
l3.setText("Minuto llegada:"+String.valueOf(cola.extraer()));
```

## Problemas propuestos

1. Un supermercado tiene tres cajas para la atención de los clientes.  
Las cajas tardan entre 7 y 11 minutos para la atención de cada cliente.  
Los clientes llegan a la zona de cajas cada 2 ó 3 minutos. (Cuando el cliente llega, si todas las cajas tienen 6 personas, el cliente se marcha del supermercado)  
Cuando el cliente llega a la zona de cajas elige la caja con una cola menor.

Realizar una simulación durante 8 horas y obtener la siguiente información:

- a - Cantidad de clientes atendidos por cada caja.
- b - Cantidad de clientes que se marcharon sin hacer compras.
- c - Tiempo promedio en cola.

## Solución

```
public class Cola {  
  
    class Nodo {  
        int info;  
        Nodo sig;  
    }  
  
    Nodo raiz,fondo;  
  
    Cola() {  
        raiz=null;  
        fondo=null;  
    }  
  
    boolean vacia (){  
        if (raiz == null)  
            return true;  
        else  
            return false;  
    }  
  
    void insertar (int info)  
    {  
        Nodo nuevo;  
        nuevo = new Nodo ();  
        nuevo.info = info;  
        nuevo.sig = null;  
        if (vacía ()) {  
            raiz = nuevo;  
            fondo = nuevo;  
        } else {  
            fondo.sig = nuevo;  
            fondo = nuevo;  
        }  
    }  
  
    int extraer ()  
    {  
        if (!vacía ())  
        {  
            int informacion = raiz.info;  
            if (raiz == fondo){  
                raiz = null;  
                fondo = null;  
            } else {  
                raiz = raiz.sig;  
            }  
        }  
    }  
}
```

```

        }
        return informacion;
    } else
        return Integer.MAX_VALUE;
    }

    public void imprimir() {
        Nodo reco=raiz;
        System.out.println("Listado de todos los elementos de la cola.");
        while (reco!=null) {
            System.out.print(reco.info+"-");
            reco=reco.sig;
        }
        System.out.println();
    }

    public int cantidad() {
        int cant=0;
        Nodo reco=raiz;
        while (reco!=null) {
            cant++;
            reco=reco.sig;
        }
        return cant;
    }
}

import javax.swing.*;

import java.awt.event.*;

public class Supermercado extends JFrame implements ActionListener {
    private JButton boton1;
    private JLabel l1,l2,l3;
    public Supermercado() {
        setLayout(null);
        boton1=new JButton("Activar Simulación");
        boton1.setBounds(10,10,180,30);
        add(boton1);
        boton1.addActionListener(this);
        l1=new JLabel("Clientes atendidos por caja:");
        l1.setBounds(10,50,400,30);
        add(l1);
        l2=new JLabel("Se marchan sin hacer compras:");
        l2.setBounds(10,90,400,30);
    }
}

```

```

add(l2);
l3=new JLabel("Tiempo promedio en cola:");
l3.setBounds(10,130,400,30);
add(l3);
}
public void actionPerformed(ActionEvent e) {
    if (e.getSource()==boton1) {
        simulacion();
    }
}

public void simulacion () {
    int estado1=0,estado2=0,estado3=0;
    int marchan=0;
    int llegada = 2 + (int) (Math.random () * 2);
    int salida1=-1,salida2=-1,salida3=-1;
    int cantAte1=0,cantAte2=0,cantAte3=0;
    int tiempoEnCola=0;
    int cantidadEnCola=0;
    Cola cola1 = new Cola ();
    Cola cola2 = new Cola ();
    Cola cola3 = new Cola ();
    for (int minuto = 0 ; minuto < 600 ; minuto++) {
        if (llegada == minuto) {
            if (estado1==0) {
                estado1=1;
                salida1=minuto+7+(int)(Math.random()*5);
            } else {
                if (estado2==0) {
                    estado2=1;
                    salida2=minuto+7+(int)(Math.random()*5);
                } else {
                    if (estado3==0) {
                        estado3=1;
                        salida3=minuto+7+(int)(Math.random()*5);
                    } else {
                        if (cola1.cantidad()==6 && cola2.cantidad()==6 &&
cola3.cantidad()==6) {
                            marchan++;
                        } else {
                            if (cola1.cantidad()<=cola2.cantidad() &&
cola1.cantidad()<=cola3.cantidad()) {
                                cola1.insertar(minuto);
                            } else {
                                if (cola2.cantidad()<=cola3.cantidad()) {
                                    cola2.insertar(minuto);
                                } else {
                                    cola3.insertar(minuto);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

    }
    }
    }
    }

    }
    llegada=minuto+ 2+ (int) (Math.random () * 2);
}
if (salida1 == minuto) {
    cantAte1++;
    estado1=0;
    if(!cola1.vacia()) {
        estado1=1;
        int m=cola1.extraer();
        salida1=minuto+7+(int)(Math.random()*5);
        tiempoEnCola=tiempoEnCola+(minuto-m);
        cantidadEnCola++;
    }
}
if (salida2 == minuto) {
    cantAte2++;
    estado2=0;
    if(!cola2.vacia()) {
        estado2=1;
        int m=cola2.extraer();
        salida2=minuto+7+(int)(Math.random()*5);
        tiempoEnCola=tiempoEnCola+(minuto-m);
        cantidadEnCola++;
    }
}
if (salida3 == minuto) {
    cantAte3++;
    estado3=0;
    if(!cola3.vacia()) {
        estado3=1;
        int m=cola3.extraer();
        salida3=minuto+7+(int)(Math.random()*5);
        tiempoEnCola=tiempoEnCola+(minuto-m);
        cantidadEnCola++;
    }
}
}

l1.setText("Clientes atendidos por caja: caja1="+cantAte1+" caja2="+cantAte2+"
caja3="+cantAte3);
l2.setText("Se marchan sin hacer compras:"+marchan);
if (cantidadEnCola>0) {
    int tiempoPromedio=tiempoEnCola/cantidadEnCola;

```

```
        l3.setText("Tiempo promedio en cola:"+tiempoPromedio);  
    }  
}  
  
public static void main(String[] ar) {  
    Supermercado super1=new Supermercado();  
    super1.setBounds(0,0,390,250);  
    super1.setVisible(true);  
}  
}
```





Muchas gracias hasta la próxima clase.

Alsina 16 [B1642FNB] San Isidro | Pcia. De Buenos Aires |Argentina |

TEL.: [011] 4742-1532 o [011] 4742-1665 |

www.institutosanisidro.com.ar [info@institutosanisidro.com.ar](mailto:info@institutosanisidro.com.ar)