INSTITUTO SAN ISIDRO

Curso de Java a distancia

Clase 25: Colecciones: Queue y PriorityQueue

Queue

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 utilizar la interfaz Queue y mediante la clase LinkedList administre la lista tipo cola.

Programa:

```
import java.util.LinkedList;
import java.util.Queue;

public class PruebaQueue {

   public static void main(String[] args) {
      Queue<String> cola1 = new LinkedList<String>();
      System.out.println("Insertamos tres elementos en la cola: juan, ana y luis");
      cola1.add("juan");
      cola1.add("ana");
      cola1.add("luis");
      System.out.println("Cantidad de elementos en la cola:" + cola1.size());
      System.out.println("Extraemos un elemento de la cola:" + cola1.poll());
      System.out.println("Cantidad de elementos en la cola:" + cola1.size());
      System.out.println("Consultamos el primer elemento de la cola sin extraerlo:" + cola1.peek());
      System.out.println("Cantidad de elementos en la cola:" + cola1.size());
      System.out.println("Extraemos uno a un cada elemento de la cola mientras no este vacía:");
      while (!cola1.isEmpty())
         System.out.print(cola1.poll() + "-");
      System.out.println();

      Queue<Integer> cola2 = new LinkedList<Integer>();
      cola2.add(70);
      cola2.add(120);
      cola2.add(6);
      System.out.println("Imprimimos la cola de enteros");
      for (Integer elemento : cola2)
         System.out.print(elemento + "-");
      System.out.println();
```

```
        System.out.println("Borramos toda la cola");
        cola2.clear();
        System.out.println("Cantidad de elementos en la cola de enteros:" + cola2.size());
    }

}
```

La diferencia con respecto a la administración de pilas en Java es que para trabajar con colas debemos crear un objeto de la clase LinkedList e implementar la interfaz Queue:

```
        Queue<String> cola1 = new LinkedList<String>();
```

La clase LinkedList implementa la interfaz Queue que es la que declara los método principales para trabajar una cola.

Añadimos elementos al final de la cola mediante el método 'add':

```
        cola1.add("juan");
        cola1.add("ana");
        cola1.add("luis");
```

Para conocer la cantidad disponemos del método size():

```
        System.out.println("Cantidad de elementos en la cola:" + cola1.size());
```

Para extraer el nodo de principio de la lista tipo cola lo hacemos mediante el método 'poll':

```
        System.out.println("Extraemos un elemento de la cola:" + cola1.poll());
```

Si queremos conocer el objeto primero de la cola sin extraerlo lo hacemos mediante el método 'peek':

```
    System.out.println("Consultamos el primer elemento de la cola sin extraerlo:" + cola1.peek());
```

Para conocer si la cola se encuentra vacía llamamos al método 'isEmpty':

```
        while (!cola1.isEmpty())
            System.out.print(cola1.poll() + "-");
```

Podemos utilizar un for para recorrer todos los elementos de la colección de tipo Queue con la siguiente sintaxis (no se eliminan los elementos de la cola):

```
        for (Integer elemento : cola2)
            System.out.print(elemento + "-");
```

Para eliminar todos los elementos de una pila empleamos el método clear:

```
        cola2.clear();
```

Más datos podemos conseguir visitando la documentación oficial de la interfaz Queue.

PriorityQueue

Una variante de una cola clásica la implementa la clase PriorityQueue. Cuando se agregan elementos a la cola se organiza según su valor, por ejemplo si es un número se ingresan de menor a mayor.

Veamos un ejemplo como se organizan los valores en la cola con prioridad:

Programa:

```
import java.util.PriorityQueue;

public class PruebaPriorityQueue {
```

```
   public static void main(String[] args) {
      PriorityQueue<Integer> cola1 = new PriorityQueue<Integer>();
      cola1.add(70);
      cola1.add(120);
      cola1.add(6);
      System.out.println("Imprimimos la cola con prioridades de enteros");
      while (!cola1.isEmpty())
         System.out.print(cola1.poll() + "-");
   }

}
```

Creamos un objeto de la clase PriorityQueue que almacene objetos de la clase Integer:

```
      PriorityQueue<Integer> cola1 = new PriorityQueue<Integer>();
```

Cargamos tres objetos en la cola de prioridad:

```
      cola1.add(70);
      cola1.add(120);
      cola1.add(6);
```

Mediante un while comenzamos a recuperar los elementos de la cola con prioridad y podemos comprobar que el primero de la cola es el que tiene el valor 6, luego el 70 y finalmente el 120:

```
      while (!cola1.isEmpty())
         System.out.print(cola1.poll() + "-");
```

Problema de aplicación de una cola

Cuando implementamos desde cero el concepto de una cola con el lenguaje Java desarrollamos una serie de ejercicios de aplicación. Los volveremos a codificar pero ahora utilizando la interfaz Queue y la clase LinkedList.

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:

```
import javax.swing.*;
import java.awt.event.*;
import java.util.LinkedList;
import java.util.Queue;
```

```
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;
      Queue<Integer> cola = new LinkedList<Integer>();
      for (int minuto = 0; minuto < 600; minuto++) {
         if (llegada == minuto) {
            if (estado == 0) {
               estado = 1;
               salida = minuto + 2 + (int) (Math.random() * 3);
            } else {
               cola.add(minuto);
            }
            llegada = minuto + 2 + (int) (Math.random() * 2);
         }
         if (salida == minuto) {
            estado = 0;
            cantAtendidas++;
            if (!cola.isEmpty()) {
               cola.poll();
               estado = 1;
               salida = minuto + 2 + (int) (Math.random() * 3);
            }
         }
      }
      l1.setText("Atendidos:" + String.valueOf(cantAtendidas));
      l2.setText("En cola" + String.valueOf(cola.size()));
      if (!cola.isEmpty())
         l3.setText("Minuto llegada:" + String.valueOf(cola.peek()));
   }

   public static void main(String[] ar) {
      Cajero cajero1 = new Cajero();
```

```
        cajero1.setBounds(0, 0, 340, 250);
        cajero1.setDefaultCloseOperation(EXIT_ON_CLOSE);
        cajero1.setVisible(true);
    }
}
```

Problema propuesto

1. Un supermercado tiene tres cajas para la atención de los clientes.
   Las cajeras 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

```
import javax.swing.*;

import java.awt.event.*;
import java.util.LinkedList;
import java.util.Queue;

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;
      Queue<Integer> cola1 = new LinkedList<Integer>();
      Queue<Integer> cola2 = new LinkedList<Integer>();
      Queue<Integer> cola3 = new LinkedList<Integer>();
      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.size() == 6 && cola2.size() == 6 && cola3.size() == 6) {
                        marchan++;
                    } else {
                        if (cola1.size() <= cola2.size() && cola1.size() <= cola3.size()) {
                            cola1.add(minuto);
                        } else {
                            if (cola2.size() <= cola3.size()) {
                                cola2.add(minuto);
                            } else {
                                cola3.add(minuto);
                            }
                        }
                    }
                }
            }

        }
        llegada = minuto + 2 + (int) (Math.random() * 2);
    }
    if (salida1 == minuto) {
        cantAte1++;
        estado1 = 0;
        if (!cola1.isEmpty()) {
            estado1 = 1;
            int m = cola1.poll();
            salida1 = minuto + 7 + (int) (Math.random() * 5);
            tiempoEnCola = tiempoEnCola + (minuto - m);
            cantidadEnCola++;
        }
    }
    if (salida2 == minuto) {
        cantAte2++;
        estado2 = 0;
        if (!cola2.isEmpty()) {
            estado2 = 1;
            int m = cola2.poll();
            salida2 = minuto + 7 + (int) (Math.random() * 5);
            tiempoEnCola = tiempoEnCola + (minuto - m);
            cantidadEnCola++;
        }
    }
    if (salida3 == minuto) {
        cantAte3++;
        estado3 = 0;
        if (!cola3.isEmpty()) {
            estado3 = 1;
            int m = cola3.poll();
            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.setDefaultCloseOperation(EXIT_ON_CLOSE);
        super1.setVisible(true);
    }
}
```

Colecciones: LinkedList

La clase LinkedList implementa la lógica para trabajar con listas genéricas, es decir podemos insertar y extraer elementos de cualquier parte de la lista.

Confeccionaremos un programa para llamar los principales métodos de esta clase.

Programa:

```
import java.util.LinkedList;

public class PruebaLinkedList {

    public static void imprimir(LinkedList<String> lista) {
        for (String elemento : lista)
            System.out.print(elemento + "-");
        System.out.println();
    }

    public static void main(String[] args) {
        LinkedList<String> lista1 = new LinkedList<String>();
        lista1.add("juan");
        lista1.add("Luis");
        lista1.add("Carlos");
        imprimir(lista1);
        lista1.add(1, "ana");
        imprimir(lista1);
        lista1.remove(0);
        imprimir(lista1);
        lista1.remove("Carlos");
        imprimir(lista1);
        System.out.println("Cantidad de elementos en la lista:" + lista1.size());
        if (lista1.contains("ana"))
            System.out.println("El nombre 'ana' si esta almacenado en la lista");
        else
            System.out.println("El nombre 'ana' no esta almacenado en la lista");
        System.out.println("El segundo elemento de la lista es:" + lista1.get(1));
        lista1.clear();
        if (lista1.isEmpty())
            System.out.println("La lista se encuentra vacía");
    }
}
```

Para trabajar con listas genéricas debemos importar la clase LinkedList:

```
import java.util.LinkedList;
```

Creamos un objeto de la clase LinkedList:

```
LinkedList<String> lista1 = new LinkedList<String>();
```

La lista administra objetos de la clase String, luego mediante el método 'add' añadimos al final nodos:

```
lista1.add("juan");
lista1.add("Luis");
lista1.add("Carlos");
```

Llamamos al método estático 'imprimir' para mostrar todos los elementos de la lista:

```
imprimir(lista1);
```

El método estático recibe la lista y mediante un for recorre la colección mostrando sus elementos:

```
public static void imprimir(LinkedList<String> lista) {
   for (String elemento : lista)
      System.out.print(elemento + "-");
   System.out.println();
}
```

El método 'add' de la clase LinkedList se encuentra sobrecargado, hay un segundo método 'add' con dos parámetros que recibe en el primer parámetro la posición donde se debe insertar el nodo y como segundo parámetro el dato a almacenar:

```
lista1.add(1, "ana");
```

Para eliminar un nodo de la lista debemos llamar al método 'remove' y pasar la posición del nodo a eliminar:

```
lista1.remove(0);
```

También podemos eliminar los nodos que coinciden con cierta información:

```
lista1.remove("Carlos");
```

La cantidad de nodos nos lo suministra el método 'size':

```
System.out.println("Cantidad de elementos en la lista:" + lista1.size());
```

Para conocer si la lista almacena cierto valor disponemos del método 'contains':

```
if (lista1.contains("ana"))
   System.out.println("El nombre 'ana' si esta almacenado en la lista");
else
   System.out.println("El nombre 'ana' no esta almacenado en la lista");
```

Para recuperar el dato de un nodo sin eliminarlo podemos hace uso del método 'get':

```
System.out.println("El segundo elemento de la lista es:" + lista1.get(1));
```

Eliminamos todos los nodos de la lista mediante el método 'clear':

```
lista1.clear();
```

Podemos consultar si la lista se encuentra vacía mediante el método 'isEmpty':

```
if (lista1.isEmpty())
   System.out.println("La lista se encuentra vacía");
```

Más datos podemos conseguir visitando la documentación oficial de la clase LinkedList.

Problema

Confeccionar el juego de la serpiente (snake) utilizando un objeto de la clase LinkedList para representar cada trozo de la misma.

Programa:

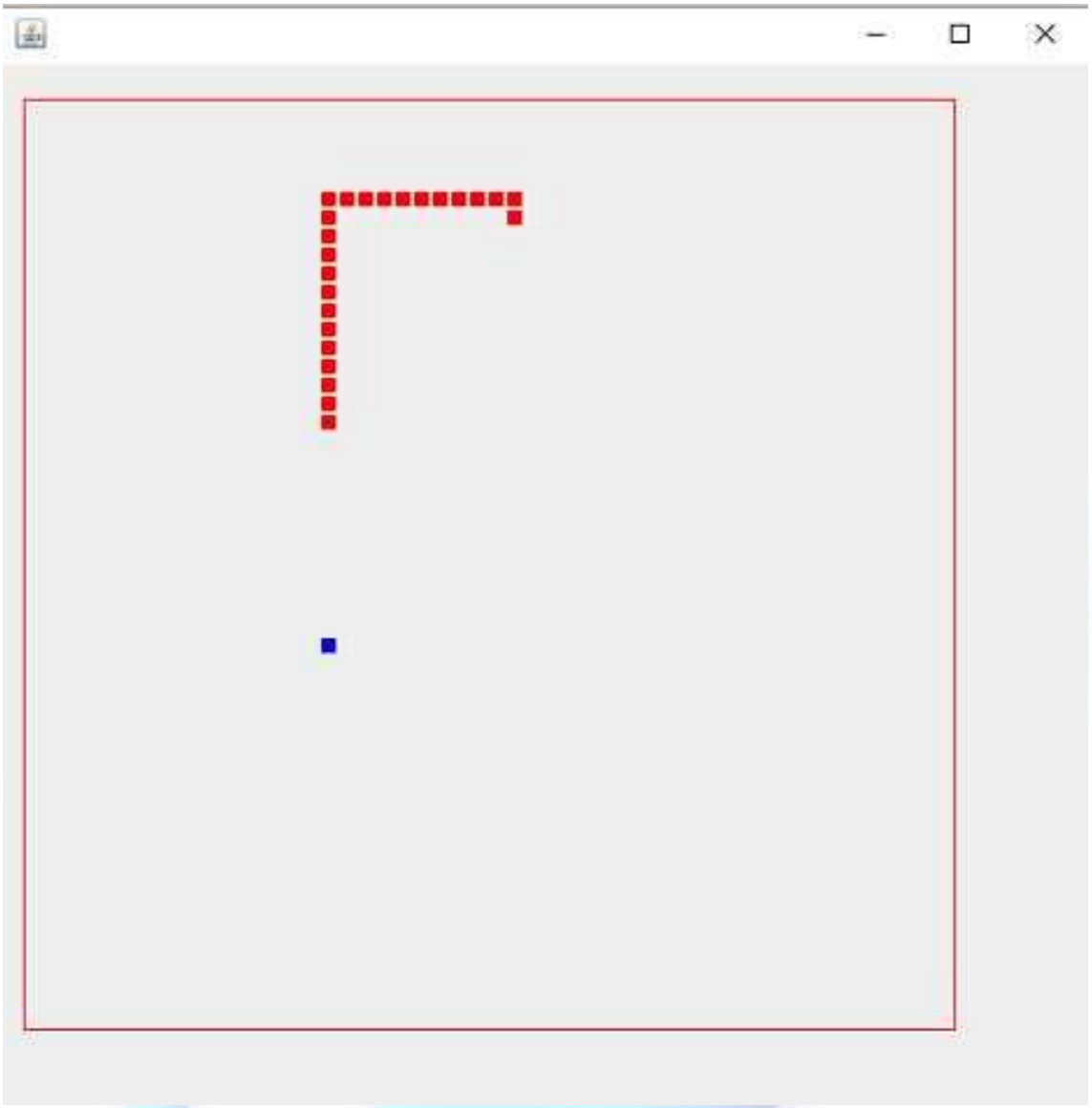
```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.util.LinkedList;

import javax.swing.JFrame;

public class Vibora extends JFrame implements Runnable, KeyListener {

   private LinkedList<Punto> lista = new LinkedList<Punto>();
   private int columna, fila; // columna y fila donde se encuentra la cabeza de la vibora
   private int colfruta, filfruta; // columna y fila donde se encuentra la fruta
   private boolean activo = true; // disponemos en false cuando finaliza el juego
   private Direccion direccion = Direccion.DERECHA;
   private Thread hilo; // Hilo de nuestro programa
   private int crecimiento = 0; // indica la cantidad de cuadraditos que debe crecer la vibora
   private Image imagen; // Para evitar el parpadeo del repaint()
   private Graphics bufferGraphics;// Se dibuja en memoria para evitar parpadeo

   private enum Direccion {
      IZQUIERDA, DERECHA, SUBE, BAJA
   };

   class Punto {
      int x, y;

      public Punto(int x, int y) {
         this.x = x;
         this.y = y;
      }
   }

   public Vibora() {
      // escuchamos los eventos de teclado para identificar cuando se presionan las
      // teclas de flechas
      this.addKeyListener(this);
      // la vibora comienza con cuatro cuadraditos
      lista.add(new Punto(4, 25));
      lista.add(new Punto(3, 25));
      lista.add(new Punto(2, 25));
      lista.add(new Punto(1, 25));
      // indicamos la ubicacion de la cabeza de la vibora
      columna = 4;
      fila = 25;
      // generamos la coordenada de la fruta
      colfruta = (int) (Math.random() * 50);
      filfruta = (int) (Math.random() * 50);
      // creamos el hilo y lo arrancamos (con esto se ejecuta el metodo run()
      hilo = new Thread(this);
      hilo.start();
   }
```

```
@Override
public void run() {
   while (activo) {
      try {
         // dormimos el hilo durante una décima de segundo para que no se mueva tan
         // rapidamente la vibora
         Thread.sleep(100);
         // segun el valor de la variable direccion generamos la nueva posicion de la
         // cabeza de la vibora
         switch (direccion) {
         case DERECHA:
            columna++;
            break;
         case IZQUIERDA:
            columna--;
            break;
         case SUBE:
            fila--;
            break;
         case BAJA:
            fila++;
            break;
         }

         repaint();
         sePisa();
         // insertamos la coordenada de la cabeza de la vibora en la lista
         lista.addFirst(new Punto(columna, fila));

         if (this.verificarComeFruta() == false && this.crecimiento == 0) {
            // si no estamos en la coordenada de la fruta y no debe crecer la vibora
            // borramos el ultimo nodo de la lista
            // esto hace que la lista siga teniendo la misma cantidad de nodos
            // ls.borrarUltimo();
            lista.remove(lista.size() - 1);
         } else {
            // Si creciento es mayor a cero es que debemos hacer crecer la vibora
            if (this.crecimiento > 0)
               this.crecimiento--;
         }
         verificarFin();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}

// controlamos si la cabeza de la vibora se encuentra dentro de su cuerpo
private void sePisa() {
   for (Punto p : lista) {
      if (p.x == columna && p.y == fila) {
         activo = false;
         setTitle("Perdiste");
      }
   }
}

// controlamos si estamos fuera de la region del tablero
private void verificarFin() {
   if (columna < 0 || columna >= 50 || fila < 0 || fila >= 50) {
      activo = false;
```

```
            setTitle("Perdiste");
        }
    }

    private boolean verificarComeFruta() {
        if (columna == colfruta && fila == filfruta) {
            colfruta = (int) (Math.random() * 50);
            filfruta = (int) (Math.random() * 50);
            this.crecimiento = 10;
            return true;
        } else
            return false;
    }

    public void paint(Graphics g) {
        super.paint(g);
        if (!lista.isEmpty()) {
            if (imagen == null) {
                imagen = createImage(this.getSize().width, this.getSize().height);
                bufferGraphics = imagen.getGraphics();
            }
            // borramos la imagen de memoria
            bufferGraphics.clearRect(0, 0, getSize().width, getSize().width);
            // dibujar recuadro
            bufferGraphics.setColor(Color.red);
            bufferGraphics.drawRect(20, 50, 500, 500);
            // dibujar vibora
            for (Punto punto : lista) {
                bufferGraphics.fillRect(punto.x * 10 + 20, 50 + punto.y * 10, 8, 8);
            }
            // dibujar fruta
            bufferGraphics.setColor(Color.blue);
            bufferGraphics.fillRect(colfruta * 10 + 20, filfruta * 10 + 50, 8, 8);
            g.drawImage(imagen, 0, 0, this);
        }
    }

    public void keyPressed(KeyEvent arg0) {
        if (arg0.getKeyCode() == KeyEvent.VK_RIGHT) {
            direccion = Direccion.DERECHA;
        }
        if (arg0.getKeyCode() == KeyEvent.VK_LEFT) {
            direccion = Direccion.IZQUIERDA;
        }
        if (arg0.getKeyCode() == KeyEvent.VK_UP) {
            direccion = Direccion.SUBE;
        }
        if (arg0.getKeyCode() == KeyEvent.VK_DOWN) {
            direccion = Direccion.BAJA;
        }
    }

    public void keyReleased(KeyEvent arg0) {
    }

    public void keyTyped(KeyEvent arg0) {
    }

    public static void main(String[] args) {
        Vibora f = new Vibora();
        f.setSize(600, 600);
```

```
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

}
```

Cuando lo ejecutemos tendremos un resultado similar a esto:



Colecciones: ArrayList

La clase ArrayList implementa la lógica para trabajar con listas genéricas, es decir podemos insertar y extraer elementos de cualquier parte de la lista. La diferencia del ArrayList con la clase LinkedList es la implementación interna de los algoritmos. La clase LinkedList emplea una lista doblemente encadenada y la clase ArrayList utiliza un arreglo que se redimensiona en forma automática según se efectúan inserciones y extracciones de datos.

La principal ventaja de emplear la clase ArrayList es que el acceso a un elemento de la lista es inmediato mediante el método 'get', en cambio la implementación del método 'get' en la clase LinkedList requiere recorrer en forma secuencial la lista hasta llegar a la posición a buscar.

Si la lista no va ha tener grandes cambios en inserciones y extracciones durante la ejecución del programa es más común utilizar la clase ArrayList en lugar de LinkedList.

Confeccionaremos el mismo programa que hicimos con la clase LinkedList pero empleando ahora la clase ArrayList.

Programa:

```
import java.util.ArrayList;

public class PruebaArrayList {
   public static void imprimir(ArrayList<String> lista) {
      for (String elemento : lista)
         System.out.print(elemento + "-");
      System.out.println();
   }

   public static void main(String[] args) {
      ArrayList<String> lista1 = new ArrayList<String>();

      lista1.add("juan");
      lista1.add("Luis");
      lista1.add("Carlos");

      imprimir(lista1);
      lista1.add(1, "ana");
      imprimir(lista1);
      lista1.remove(0);
      imprimir(lista1);
      lista1.remove("Carlos");
      imprimir(lista1);
      System.out.println("Cantidad de elementos en la lista:" + lista1.size());
      if (lista1.contains("ana"))
         System.out.println("El nombre 'ana' si esta almacenado en la lista");
      else
         System.out.println("El nombre 'ana' no esta almacenado en la lista");
      System.out.println("El segundo elemento de la lista es:" + lista1.get(1));
      lista1.clear();
      if (lista1.isEmpty())
         System.out.println("La lista se encuentra vacía");
   }

}
```

Para trabajar con una lista implementada en forma interna con un arreglo debemos importar la clase ArrayList:

```
import java.util.ArrayList;
```

Creamos un objeto de la clase ArrayList:

```
      ArrayList<String> lista1 = new ArrayList<String>();
```

La lista administra objetos de la clase String, luego mediante el método 'add' añadimos componentes al final:

```
      lista1.add("juan");
      lista1.add("Luis");
      lista1.add("Carlos");
```

Llamamos al método estático 'imprimir' para mostrar todos los elementos del ArrayList:

```
imprimir(lista1);
```

El método estático recibe la lista y mediante un for recorre la colección mostrando sus elementos:

```
public static void imprimir(ArrayList<String> lista) {
   for (String elemento : lista)
      System.out.print(elemento + "-");
   System.out.println();
}
```

El método 'add' de la clase ArrayList se encuentra sobrecargado, hay un segundo método 'add' con dos parámetros que recibe en el primer parámetro la posición donde se debe insertar el nodo y como segundo parámetro el dato a almacenar (tener en cuenta que internamente el ArrayList debe desplazar todos los elementos a derecha una posición):

```
lista1.add(1, "ana");
```

Para eliminar un nodo de la lista debemos llamar al método 'remove' y pasar la posición del nodo a eliminar:

```
lista1.remove(0);
```

También podemos eliminar los elementos que coinciden con cierta información:

```
lista1.remove("Carlos");
```

La cantidad de elementos nos lo suministra el método 'size':

```
System.out.println("Cantidad de elementos en la lista:" + lista1.size());
```

Para conocer si la lista almacena cierto valor disponemos del método 'contains':

```
if (lista1.contains("ana"))
   System.out.println("El nombre 'ana' si esta almacenado en la lista");
else
   System.out.println("El nombre 'ana' no esta almacenado en la lista");
```

Para recuperar el dato de un nodo sin eliminarlo podemos hace uso del método 'get' (en un ArrayList este método es muy rápido y no depende de la cantidad de elementos):

```
System.out.println("El segundo elemento de la lista es:" + lista1.get(1));
```

Eliminamos todos los nodos de la lista mediante el método 'clear':

```
lista1.clear();
```

Podemos consultar si la lista se encuentra vacía mediante el método 'isEmpty':

```
if (lista1.isEmpty())
   System.out.println("La lista se encuentra vacía");
```

Más datos podemos conseguir visitando la documentación oficial de la clase ArrayList.

Problema

Crear un proyecto y dentro del mismo crear dos clases. La primer clase se debe llamar 'Carta', con dos atributos el palo y el número de carta. Por otro lado declarar una clase llamada 'Mazo' que contenga un ArrayList de tipo 'Carta'. Imprimir las cartas en forma ordenadas según como se insertaron y luego mezclar y volver a imprimir.

Programa:

```
public class Carta {
   public enum Palo {
      TREBOL, DIAMANTE, CORAZON, PICA
   };

   private int numero;
   private Palo palo;

   Carta(int numero, Palo palo) {
      this.numero = numero;
      this.palo = palo;
   }

   public void imprimir() {
      System.out.println(numero + " - " + palo.toString().toLowerCase());
   }

   public Palo retornarPalo() {
      return palo;
   }

}

import java.util.ArrayList;
import java.util.Collections;

public class Mazo {
   private ArrayList<Carta> cartas;

   Mazo() {
      cartas = new ArrayList<Carta>(8);
      cartas.add(new Carta(1, Carta.Palo.TREBOL));
      cartas.add(new Carta(2, Carta.Palo.TREBOL));
      cartas.add(new Carta(1, Carta.Palo.DIAMANTE));
      cartas.add(new Carta(2, Carta.Palo.DIAMANTE));
      cartas.add(new Carta(1, Carta.Palo.PICA));
      cartas.add(new Carta(2, Carta.Palo.PICA));
      cartas.add(new Carta(1, Carta.Palo.CORAZON));
      cartas.add(new Carta(2, Carta.Palo.CORAZON));
   }

   public void imprimir() {
      for (Carta carta : cartas)
         carta.imprimir();
   }

   public void barajar() {
      Collections.shuffle(cartas);
   }

   public static void main(String[] ar) {
      Mazo mazo = new Mazo();
      System.out.println("Mazo de cartas ordenado");
      mazo.imprimir();
      mazo.barajar();
      System.out.println("Mazo de cartas despues de barajar");
      mazo.imprimir();
   }
```

}

La clase Carta ya la analizamos en conceptos anteriores, veamos lo principal de la clase Mazo.

Definimos un ArrayList llamado cartas:

```
   private ArrayList<Carta> cartas;
```

En el constructor creamos el ArrayList pasando el valor 8 ya que ese será el número de cartas de nuestro mazo (si no pasamos el 8 el ArrayList se redimensiona automáticamente, el pasar un valor es conveniente para que sea más eficiente el programa):

```
Mazo() { cartas = new ArrayList<Carta>(8);
```

Agregamos al ArrayList los 8 elementos:

```
      cartas.add(new Carta(1, Carta.Palo.TREBOL));
      cartas.add(new Carta(2, Carta.Palo.TREBOL));
      cartas.add(new Carta(1, Carta.Palo.DIAMANTE));
      cartas.add(new Carta(2, Carta.Palo.DIAMANTE));
      cartas.add(new Carta(1, Carta.Palo.PICA));
      cartas.add(new Carta(2, Carta.Palo.PICA));
      cartas.add(new Carta(1, Carta.Palo.CORAZON));
      cartas.add(new Carta(2, Carta.Palo.CORAZON));
```

Para imprimir todas las carta recorremos la colección mediante un for:

```
   public void imprimir() {
      for (Carta carta : cartas)
          carta.imprimir();
   }
```

Para mezclar las cartas la forma más sencilla es emplear el método estático 'shuffle' que pertenece a la clase 'Collections':

```
   public void barajar() {
      Collections.shuffle(cartas);
   }
```

Es importante hacer el import tanto de la clase ArrayList como de Collections:

```
import java.util.ArrayList;
import java.util.Collections;
```

En el método main solo nos queda crear un objeto de la clase 'Mazo' proceder a imprimir todo el mazo, seguidamente mezclar las cartas y volver a imprimir:

```
   public static void main(String[] ar) {
      Mazo mazo = new Mazo();
      System.out.println("Mazo de cartas ordenado");
      mazo.imprimir();
      mazo.barajar();
      System.out.println("Mazo de cartas despues de barajar");
      mazo.imprimir();
   }
```

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