Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas
Introducción a la Programación de Computadoras 1
Primer Semestre 2023

Manual de programador

Nombre: Jose Javier Bonilla Salazar
Carnet: 202200035

índice

1. Librerías

```
import java.util.LinkedList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JLabel;
```

```
import java.awt.Color;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JButton;
```

A continuación, se muestran las clases que darán funcionamiento a nuestra aplicación, la primera servirá para el uso de arreglos dinámicos.
La segunda y tercera servirán para el manejo de expresiones regulares.
La cuarta y quinta servirán como opciones adicionales de diseño de la librería “swing”.

1.1. Java swing

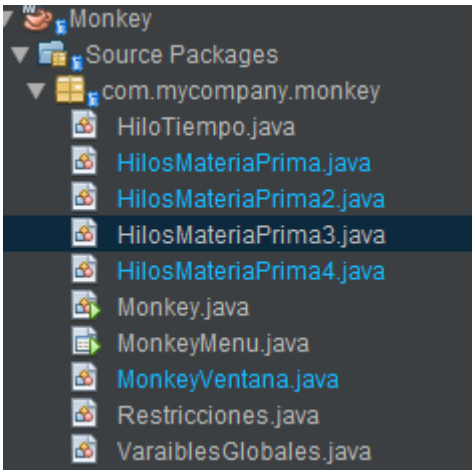
Esta librería tiene la función de crear interfaces gráficas, para un uso mas amigable con el usuario.
En este caso se trabajó manualmente la implementación de la interfaz gráfica, quiere decir que se codifico y no se utiliza la herramienta de “drag and drop”.
Al momento de utilizar esta opción se puede apreciar que se agregan muchas librerías relacionadas a “swing”

1.2. LinkedList

Esta librería se utilizó para el manejo de arreglos.

2. División de la aplicación

Esta aplicación se dividió en tres carpetas con el propósito de tener un código mas ordenado y fácil de leer.



2.1. Clase arranque

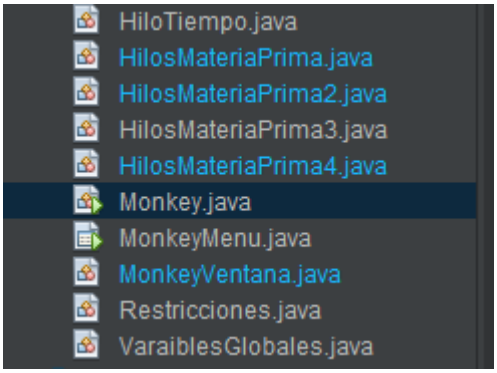
Se encuentra en la carpeta com.mycompany.monkey, como indica el nombre de la sección esta clase sirve para el arranque de la aplicación, esta clase recibe el nombre de “Moneky” en dicha clase se inicia la aplicación además de agregar la vista de la simulación.

```
public class Monkey extends JFrame{

    public Monkey() {
        setTitle("MonkeySimulation");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        MonkeyVentana ventana = new MonkeyVentana();
        this.getContentPane().add(ventana);
        this.pack();
        this.setLocationRelativeTo(null);
        this.setVisible(true);

    }

    public static void main(String[] args) {
        MonkeyMenu ventana = new MonkeyMenu();
        ventana.setVisible(true);
    }
}
```

Dentro de la clase se puede observar el método “Monkey” el cual tiene la función de devolver el panel de la simulación, esto es debido a que se codifica en vez de utilizar la herramienta de “drag and drop” por ultimo se puede ver un método estático vacío, que su única función es inicializar la vista del menú.

2.2. Clases

Se puede apreciar una gran variedad de clases, pero cuatro de ellas cumplen con la misma función, la cual es el manejo de hilos.



2.2.1. Instanciar clases

```
if(Restricciones.isNullOrEmpty(strTiempoInventario)) {
    JOptionPane.showMessageDialog(null, "El campo tiempo de inventario no puede ir vacio");
}
```

Se puede apreciar que dentro del condicional “if” se manda a llamar a la clase seguido de su método, y a su vez se manda el parámetro que se necesita. En este caso la forma de instanciar clases se realiza así, ya que todos los archivos están en una misma carpeta y es mas practico hacerlo así.

2.2.2. Variables globales

```
public class VaraiblesGlobales {

    public static int tiempoInventario;
    public static int costoInventario;
    public static int tiempoProduccion;
    public static int costoProduccion;
    public static int tiempoEmpaquetado;
    public static int costoEmpaquetado;
    public static int tiempoSalida;
    public static int costoSalida;

    public static String strTiempoInventario;
    public static String strCostoInventario;
    public static String strTiempoProduccion;
    public static String strCostoProduccion;
    public static String strTiempoEmpaquetado;
    public static String strCostoEmpaquetado;
    public static String strTiempoSalida;
    public static String strCostoSalida;

    public static boolean iniciar;

    public static int inicio;
    public static int inventario;
    public static int produccion;
    public static int empaquetado;
    public static int salida;
    public static int fin;

    public static int materiaPrima = 30;

    public static LinkedList<String> hilosIntermedio1 = new LinkedList<String>();
    public static LinkedList<String> hilosIntermedio2 = new LinkedList<String>();
    public static LinkedList<String> hilosIntermedio3 = new LinkedList<String>();



}
```

Como su nombre lo indica esta es una clase la cual guardara de forma global las variables necesarias para el manejo de la aplicación, desde enteros, flotantes, booleanos, cadenas y listas. De esta forma se puede pasar información de vista a vista de forma mas practica y sencilla. Solo se llama a la clase y la variable que se desee.

2.2.3. Restricciones

```java
public class Restricciones {

    public static boolean soloNumeros(String str) {
        for (char c : str.toCharArray()) {
            if (!Character.isDigit(c)) {
                return false;
            }
        }
        return true;
    }

    public static boolean isNullOrEmpty(String str) {
        return str == null || str.isEmpty();
    }

    public static String generatetablaHtml(String[] encbezados, String[][] filas) {
        StringBuilder sb = new StringBuilder();

        sb.append("<h2>Reporte de costo total de la simulación</h2>\n");
        // Se abre etiqueta de inciio de la tabla
        sb.append("<table style=\"width: 25%; border: 1px solid black;\">\n");

        // Se agregan los encabezados de la tabla
        sb.append("<tr>");
        for (String encabezado : encbezados) {
            sb.append("<th style=\"border-bottom: 1px solid black;\"><strong>").append(encabezado).append("</strong></th>");
        }
        sb.append("</tr>\n");

        // Se agregan las fialas a la tabla
        for (String[] fila : filas) {
            sb.append("<tr>");
            for (String casilla : fila) {
                sb.append("<td style=\"padding: 10px; text-align: left;\">").append(casilla).append("</td>");
            }
            sb.append("</tr>\n");
        }

        // Se cierra la etiqueta d ela tabla
        sb.append("</table>\n");

        return sb.toString();
    }
```

Como su nombre lo indica, dicha clase es para colocar métodos que servirán como restricciones para el usuario. En este caso temeos la validación para ver que ningún campo de texto venga vacío y también para validar que sea un entero. Como extra se agregó el método para generar el reporte en html. El cual es un archivo plano que se le concatena una tabla por medio de un bucle for, de esta manera se concatenan las etiquetas html y la información deseada.

2.3. Interfaz grafica

2.3.1. Obtener texto de un jTextFiel

```java
String strTiempoInventario = txtTiempoInventario.getText();
String strCostoInventario = txtCostoInventario.getText();
```

Con uso del método “getText()” se obtendrá el texto de este.

2.3.2. Mensajes de alerta

```java
if(Restricciones.isNullOrEmpty(strTiempoInventario)) {
    JOptionPane.showMessageDialog(null, "El campo tiempo de inventario no puede ir vacio");
}
```

Con uso de la clase “JOptionPane” se podrán mostrar mensajes de alerta al usuario.

2.3.3. Creación de elementos

```
private JLabel lbTiempo;
private JLabel lbTitulo;
private JLabel lbInventario, lbProduccion, lbEmpaquetado, lbSalida;
private JButton btnRegresar, btnReporte;
private JLabel lbTituloInicio,lbTituloInventario, lbTituloProduccion, lbTituloEmpaquetado, lbTituloSalida, lbTituloFinal;
private JLabel lbCirculo1,lbCirculo2,lbCirculo3,lbCirculo4,lbCirculo5,
      lbCirculo6,lbCirculo7,lbCirculo8,lbCirculo9,lbCirculo10;
```

Para crear manualmente una vista primero se deberán crear las variables del tipo deseado para generar dicho elemento.

```
//Declarando constructor para inicializar los componentes
public MonkeyVentana(){
    this.setLayout(null);
    this.setBackground(Color.white);
    this.setPreferredSize(new Dimension(800,600));
    this.setFocusable(true);

    lbTitulo = new JLabel("Tiempo Transcurrido");
    lbTitulo.setLocation(375, 5);
    lbTitulo.setSize(200,50);
    this.add(lbTitulo);

    lbTiempo = new JLabel("0:0");
    lbTiempo.setLocation(400, 15);
    lbTiempo.setSize(50,50);
    this.add(lbTiempo);
    //Pasar label a hilo tiempo
    HiloTiempo hiloTiempo = new HiloTiempo(lbTiempo);

    //Correr el hilo
    hiloTiempo.start();
```

Dentro del método principal de la clase se colocara los componente que se quieran iniciar. Se puede colocar color, dimensiones y posición a los elementos deseados.

Para agregar un elemento al panel, se deberá usar “this.add()” con esto indicamos que se quiere agregar aquí el elemento.

2.4. Reporte

```
btnReporte.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {

        int costoInventario = VaraiblesGlobales.tiempoInventario * VaraiblesGlobales.costoInventario * 30;
        String strCostoInventario = String.valueOf(costoInventario);
        int costoProduccion = VaraiblesGlobales.tiempoProduccion * VaraiblesGlobales.costoProduccion * 30;
        String strCostoProduccion = String.valueOf(costoProduccion);
        int costoEmpaquetado = VaraiblesGlobales.tiempoEmpaquetado * VaraiblesGlobales.costoEmpaquetado * 30;
        String strCostoEmpaquetado = String.valueOf(costoEmpaquetado);
        int costoSalida = VaraiblesGlobales.tiempoSalida * VaraiblesGlobales.costoSalida * 30;
        String strCostoSalida = String.valueOf(costoSalida);
        int total = costoInventario + costoProduccion + costoEmpaquetado + costoSalida;
        String strTotal = String.valueOf(total);

        String[][] filas = {{"Inventario",VaraiblesGlobales.strTiempoInventario , VaraiblesGlobales.strCostoInventario, strCostoInventario},
            {"Producción", VaraiblesGlobales.strTiempoProduccion, VaraiblesGlobales.strCostoProduccion, strCostoProduccion},
            {"Empaquetado", VaraiblesGlobales.strTiempoEmpaquetado, VaraiblesGlobales.strCostoEmpaquetado, strCostoEmpaquetado},
            {"Salida",VaraiblesGlobales.strTiempoSalida , VaraiblesGlobales.strCostoSalida, strCostoSalida},
            {"","" , "Total", strTotal}};

        String[] encbezados = {"Sector.", "Tiempo(s)", "Costo(Q/s)","Costo Total"};
        String tablaHtml = Restricciones.generatetablaHtml(encbezados, filas);

        // Crea la carpeta de reportes si no esiste
        File carpeta = new File("./reportes");
        if (!carpeta.exists()) {
            carpeta.mkdirs();
        }

        // Escribe el archivo .html dentro de la carpeta
        try {
            FileWriter fileWriter = new FileWriter("./reportes/reporteCosto.html");
            fileWriter.write(tablaHtml);
            fileWriter.close();
        } catch (IOException ex) {
            ex.printStackTrace();
        }


    }
});
```

Dicho método se colocara en el botón de reporte y se utilizara también el método de GeneraHtml guardado en “restricciones” aquí se trabajaran las formulas para calcular el costo total y se guardaran en arreglos para generar el reporte html de forma mas sencilla.

2.5. Hilos

```
public class HilosMateriaPrima extends Thread{

    private JLabel lbTituloInicio, lbTituloInventario;

    LinkedList<String> hilosInicio = new LinkedList<String>();
    LinkedList<String> hilosInventario = new LinkedList<String>();
    private String primeroInicio, primeroInventario;

    private boolean inicio = true;

    public HilosMateriaPrima(JLabel lbTituloInicio, JLabel lbTituloInventario){
        this.lbTituloInicio = lbTituloInicio;
        this.lbTituloInventario = lbTituloInventario;
    }
```

Para la creación del hilo, se deberá utilizar “extends Thread” de esta forma se indica que se incluye el componente de hilo. Primero se crearan todas las variables a utilizar y se creara un constructor que reciba los jLabel que se modificaran.

```java
@Override
public void run() {


    //Iniciara el hilo y llenara el arreglo
    for (int i = 0; i < 10; i++) {
        hilosInicio.add("hilo" + (i +1));
    }


    //Siempre se ejecutara
    while(inicio) {

        //Dara el intervalo de tiempo asignado en el menu
        try {
            sleep(VaraiblesGlobales.tiempoInventario * 1000);
        } catch (InterruptedException ex) {
            Logger.getLogger(HilosMateriaPrima.class.getName()).log(Level.SEVERE, null, ex);
        }

        //Liminatara a que el arreglo de inventario solo tenga 5 objetos y el de inicio tenga algo
        if(hilosInventario.size() <= 5 && !hilosInicio.isEmpty()){
            //validar que el arreglo de inicio tenga datos
            if(hilosInicio.get(0) != null) {
                //Variable auxiliar para guardar primera posicion y pasar al segundo arreglo
                primeroInicio = hilosInicio.get(0);
                hilosInicio.removeFirst();
                hilosInventario.add(primeroInicio);
                lbTituloInicio.setText("Inicio: " + hilosInicio.size());
                lbTituloInventario.setText("Inventario: " + hilosInventario.size());

                //Cuando haya llegado al limite el arreglo de inventario empezara a mover los datos
                if(hilosInventario.size() == 5 && !hilosInicio.isEmpty()) {
                    primeroInventario = hilosInventario.get(0);
                    hilosInventario.removeFirst();
                    VaraiblesGlobales.hilosIntermedio1.add(primeroInventario);
                    lbTituloInventario.setText("Inventario: " + hilosInventario.size());
                }
            }
        }
        //Cuando solo hayan datos en inventario los movera al siguiente arreglo
        else if (!hilosInventario.isEmpty() && hilosInicio.isEmpty()){
            primeroInventario = hilosInventario.get(0);
            hilosInventario.removeFirst();
            VaraiblesGlobales.hilosIntermedio1.add(primeroInventario);
            lbTituloInventario.setText("Inventario: " + hilosInventario.size());
        }
        //Cuando ya no hayan mas cosas el buclo se apagara
        else {
            inicio = false;
            break;
        }
```

A diferencia de los demás hilos, el primer hilo posee con un ciclo for que llena el arreglo con 30 datos, en este caso la palabra “hilo” sumado de su número, con el fin de identificarlos y tener un mejor manejo.

Posterior a eso tendremos un bucle while, el cual se ejecutará hasta que el arreglo de inicial y el de la sección que se esté trabajando ya no tengan datos.
Se tiene un “try” el cual sirve para manejar el tiempo de ejecución del hilo.
Luego se tienen condicionales que limitan el funcionamiento del arreglo inicial o intermediario y el arreglo de la sección que se esté trabajando.
Estos hilos se instanciarán en la vista de simulación y trabaran automáticamente.