



**INSTITUTO POLITECNICO NACIONAL**  
**ESCUELA SUPERIOR DE CÓMPUTO (ESCOM)**



**PROGRAMACIÓN ORIENTADA A OBJETOS**

---

NOMBRE DEL ALUMNO:

- SANTOS MÉNDEZ ULISES JESÚS

PRÁCTICA:

- HILOS

NÚMERO DE PRÁCTICA: 4

OPCIÓN:

- MARCO DIGITAL

FECHA DE ENTREGA:

- 18/06/2021

GRUPO:

- 2CM11

# Hilos

## Introducción

Se desarrollo un Marco Digital con 8 imágenes diferentes sobre diversos monstruos mitológicos y de historias de libros, a continuación, se verá el uso de los hilos para saber si existe o no la imagen haciendo uso del try y catch para el manejo de excepciones.

## Marco Teórico

### Hilos

Un programa multihilo contiene dos o más partes que pueden ejecutarse de forma concurrente. Cada parte de ese programa se llama hilo (Thread) y cada hilo establece un camino de ejecución independiente.

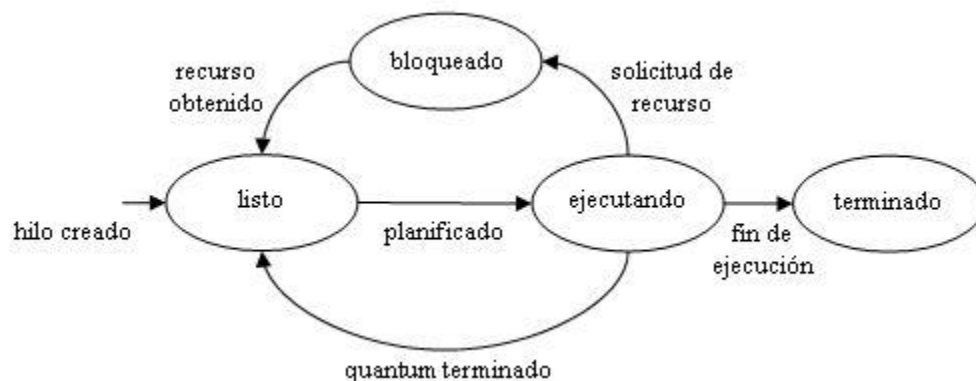
Es un tipo de multitarea distinta a la multitarea por procesos, la cual ejecuta varios programas a la vez. En Java, es un solo programa con varios hilos a la vez.

La programación multihilo permite escribir programas muy eficientes que utilizan al máximo la CPU, reduciendo al mínimo el tiempo que está libre. Esto es importante en los entornos interactivos y en red en los que se trabaja con Java.

Los hilos pueden estar en distintos estados. Un hilo puede estar ejecutándose. También puede estar preparado para ejecutarse tan pronto como disponga de tiempo de CPU. Un hilo que está ejecutándose para suspenderse, lo que equivale a detener temporalmente su actividad.

El sistema multihilo de Java está construido en torno a la clase **Thread**, sus métodos y su interfaz de apoyo **Runnable**. La clase Thread involucra la noción hilo de ejecución. Para crear un hilo, se debe transferir un objeto Runnable (objeto que instrumenta la interfaz Runnable al definir un método run ()) al constructor Thread para que defina su propio método run ().

El método run() del Thread o el objeto Runnable especificados en el cuerpo del hilo empieza a ejecutarse cuando se llama al método **start()** del objeto Thread. El hilo funciona hasta que regresa el método run() o hasta que se llama al método **stop()** del objeto Thread.



## Desarrollo

- 1) Se importaron las siguientes librerías de Java para el diseño del Marco Digital y para el uso del JFrame y del ComboBox, en ella están contenidas las clases y métodos que se utilizarán

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;
import java.lang.*;
```

- 2) Se declara la clase pública MarcoDig que implementa la interfaz Runnable y ActionListener

```
public class MarcoDig extends JFrame implements Runnable, ActionListener {
```

- 3) Se establecen las variables de instancia para general el ComboBox y las cadenas que se insertaran en el ComboBox, un panel, etiquetas y la variable que es el Hilo de control

```
//Declaramos variables de instancia
Vector <String> nombres=new Vector<String>();
String[]opcion = {"Default", "300", "500", "1000", "2000"};
Panel p;
Container ventana;
ImageIcon iconos[];
JLabel jling;
JComboBox cb;
Thread hilo; //Hilo de control
int t = 3000;
int i;
```

- 4) En el constructor del programa ponemos los parámetros para inicializar las variables de instancia, tanto del panel así como de las imágenes y su inserción al arreglo de las mismas, recordando que las dos últimas instrucciones del constructor siempre van al último, una reserva memoria para el hilo y la otra llama al constructor con new.

```

public MarcoDig(){
    super("Marco Digital");
    Arrimages();
    p = new Panel();
    iconos = new ImageIcon[nombres.size()];
    iconos[0] = new ImageIcon(nombres.elementAt(0));
    iconos[1] = new ImageIcon(nombres.elementAt(1));
    iconos[2] = new ImageIcon(nombres.elementAt(2));
    iconos[3] = new ImageIcon(nombres.elementAt(3));
    iconos[4] = new ImageIcon(nombres.elementAt(4));
    iconos[5] = new ImageIcon(nombres.elementAt(5));
    iconos[6] = new ImageIcon(nombres.elementAt(6));
    iconos[7] = new ImageIcon(nombres.elementAt(7));
    jling = new JLabel(iconos[0], JLabel.CENTER);
    p.add(jling);
    cb = new JComboBox(opcion);
    cb.setBounds(400, 20, 60, 25);
    cb.setSelectedIndex(0); // Seleccionamos el primer item del indice
    cb.addActionListener(this);
    p.add(cb);
    p.setBackground(new Color(0xCCFFCC));
    JFrame v = new JFrame("MARCO DIGITAL");
    v.add(p);
    v.setSize(1000, 900);
    v.setLocationRelativeTo(null);
    v.setVisible(true);
    v.setBackground(Color.BLACK);
    v.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    // Se reserva memoria y se llama al constructor
    hilo = new Thread(this);
    hilo.start();
}

```

5) Se manda a llamar al método Arrimages que es en donde se agregaron las imágenes

```

private void Arrimages(){
    nombres.addElement("Azathoth.jpg");
    nombres.addElement("Baku.jpg");
    nombres.addElement("Cthulhu.jpg");
    nombres.addElement("Hastur.jpg");
    nombres.addElement("Nyarlathotep.jpg");
    nombres.addElement("Ratatoskr.png");
    nombres.addElement("Shuma.png");
    nombres.addElement("Sleipner.jpg");
}

```

- 6) Se tiene el método actionPerformed que será el encargado de extraer datos debido a la interacción del usuario con el ComboBox

```
public void actionPerformed(ActionEvent e){
    if(e.getSource() == cb){
        JComboBox combocaja = (JComboBox)e.getSource();
        String op = (String)combocaja.getSelectedItem();
        if("Default".equals(op)){
            t = 3000;
        }
        if("300".equals(op)){
            t = 300;
        }
        if("500".equals(op)){
            t = 500;
        }
        if("1000".equals(op)){
            t = 1000;
        }
        if("2000".equals(op)){
            t = 2000;
        }
    }
}
```

- 7) Se establece el cuerpo del hilo con el método run en ella se hace uso del try y catch, se pidió que se hiciera uso del operador modulo.

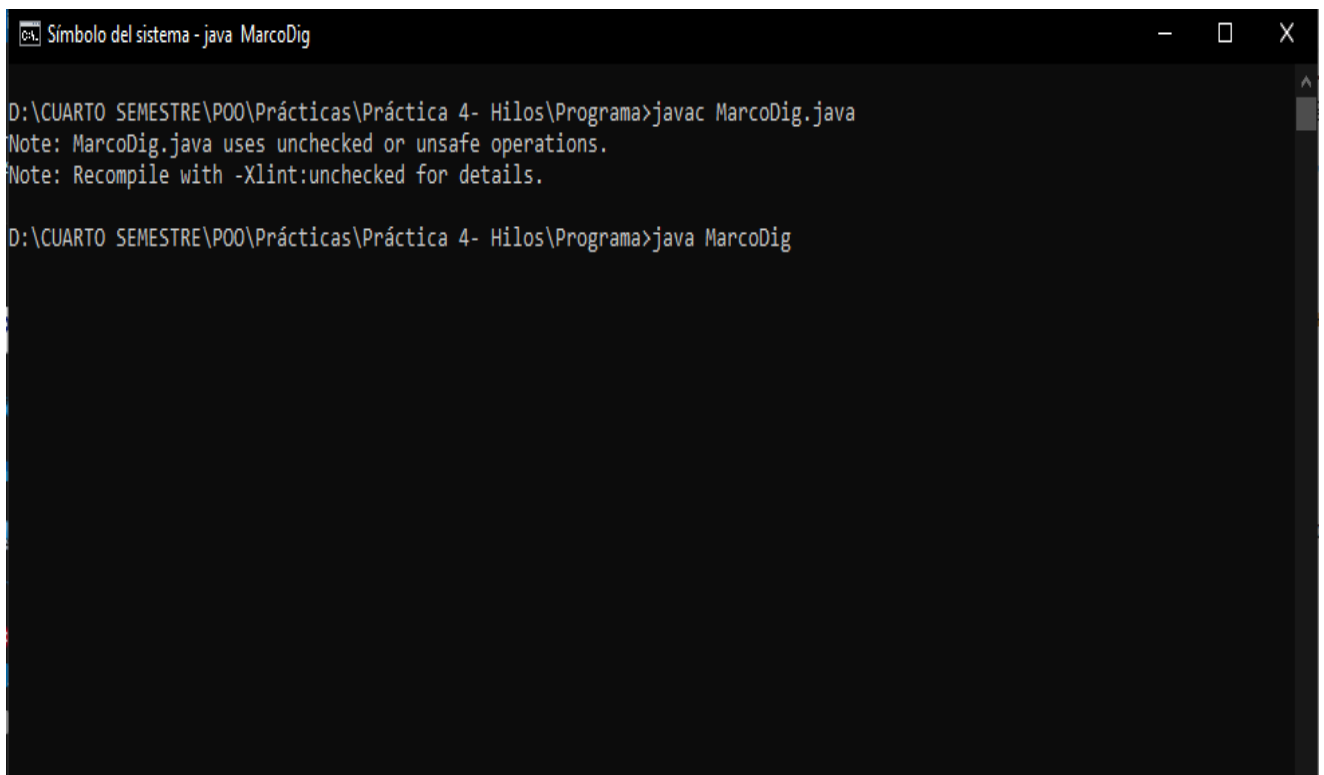
```
public void run(){
    try{
        for(i = 0; i<nombres.size(); i++){
            jling.setIcon(iconos[i % iconos.length]);
            hilo.sleep(t);
            if(i == 7){
                i=0;
            }
        }
    }
}
```

```
        catch(Exception e){  
            System.out.println("ERROR CARGA DE IMAGENES");  
        }  
    }  
}
```

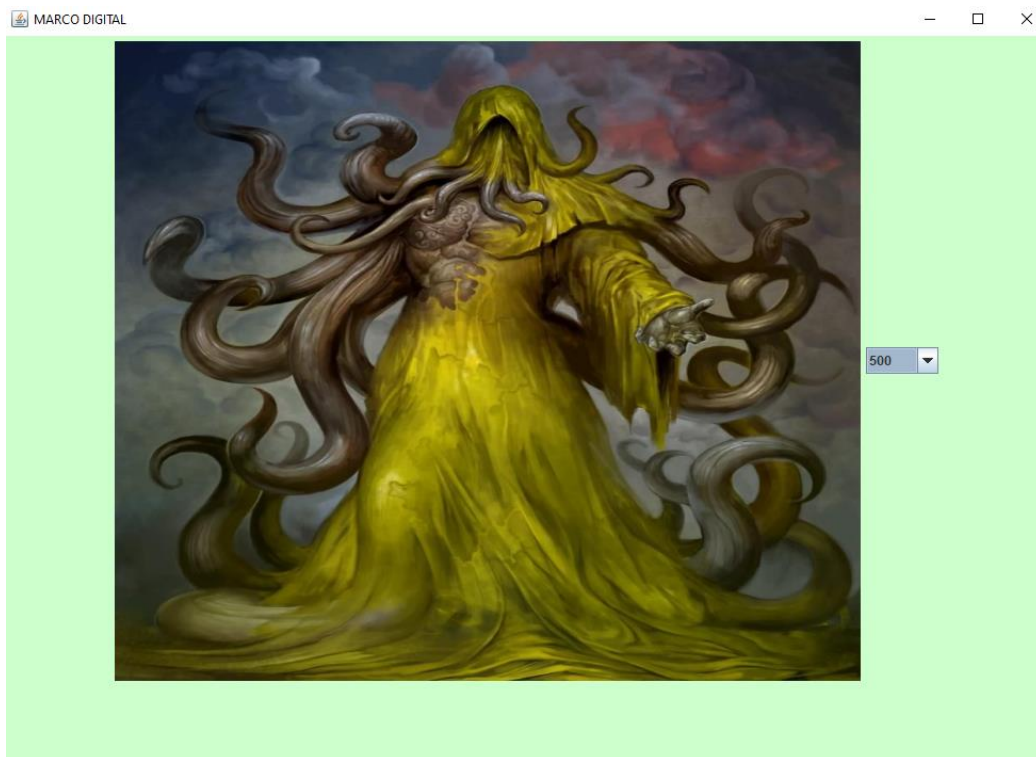
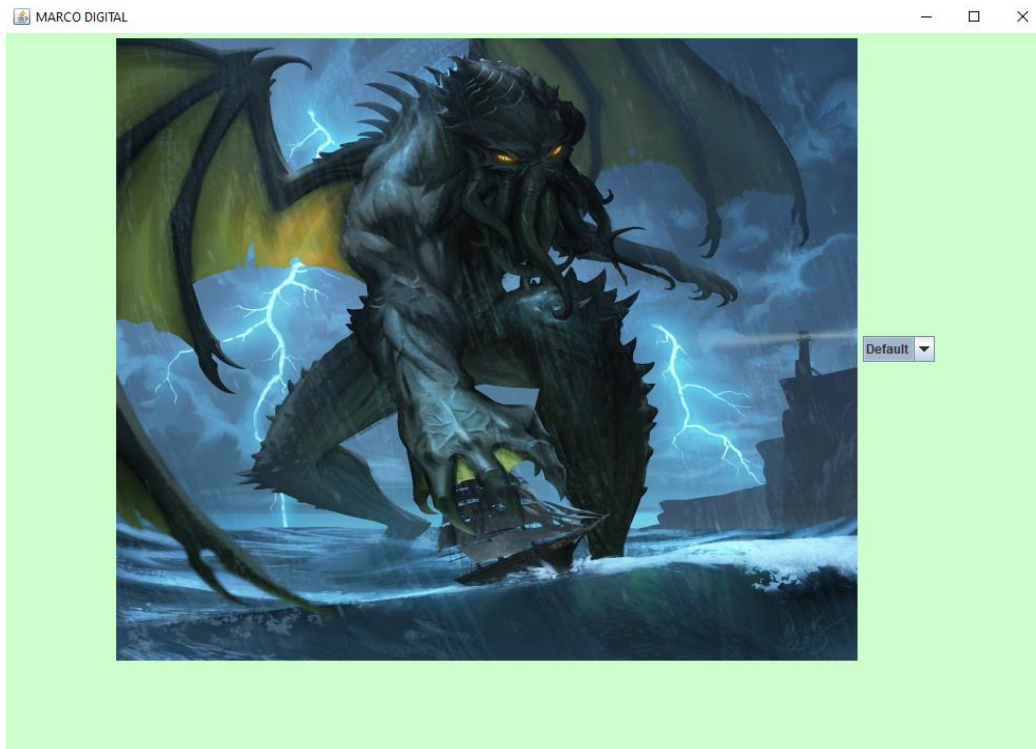
- 8) Se tiene finalmente el main que es el que nos ayudará a poder tener la clase principal para así ejecutar nuestro programa

```
    public static void main(String[] args){  
        MarcoDig marcoDig = new MarcoDig();  
    }  
}
```

- 9) Se compila y se ejecuta el programa MarcoDig.java



```
Símbolo del sistema - java MarcoDig  
D:\CUARTO SEMESTRE\POO\Prácticas\Práctica 4- Hilos\Programa>javac MarcoDig.java  
Note: MarcoDig.java uses unchecked or unsafe operations.  
Note: Recompile with -Xlint:unchecked for details.  
D:\CUARTO SEMESTRE\POO\Prácticas\Práctica 4- Hilos\Programa>java MarcoDig
```



## Conclusión

En conclusión, esta práctica me fue útil para entender mejor el concepto de hilo y como manipularlo para mejorar la eficiencia del programa y de la misma forma aplicable para el manejo de errores o excepciones que se puedan encontrar en el programa.