PRÁCTICA 9.- Contrastando Programas con y sin hilos

Objetivo

Desarrollar una aplicación que muestre las diferencias de funcionalidad entre un programa de flujo único contra otro de flujo múltiple.

Objetivos específicos:

- Desarrollar un programa en dos versiones: con hilos y sin hilos.
- Evaluar el desempeño de ambos programas y sacar las conclusiones pertinentes.

Introducción

El lenguaje Java proporciona en su estructura, el soporte para la programación multihilos. Un programa multihilos contiene dos o más partes que pueden correr concurrentemente. Cada parte de este programa es llamada un hilo, y cada hilo define una trayectoria separada de ejecución. Así, multihilo es una forma especializada de multitarea.

Se puede afirmar que estamos familiarizados con multitarea debido a que prácticamente la mayoría de los sistemas operativos la soportan. Sin embargo hay dos tipos distintos de multitarea: basada en procesos y basada en hilos. Para la mayoría la primera forma es la más familiar. Si entendemos por proceso un programa que se está ejecutando, entonces multitarea basada en procesos es la capacidad que permite a la computadora de ejecutar dos o más programas concurrentemente. Así, podemos tener el editor de textos ejecutándose a la par del navegador web y visitar una página. En la multitarea basada en procesos, un programa es la unidad más pequeña de código que puede ser controlada por el administrador interno.

En un sistema multitarea basado en hilos, el hilo es la unidad más pequeña de código despachado. Esto quiere decir que un solo programa puede realizar dos o más tareas simultáneamente. Por ejemplo, un editor de texto puede estar dando formato a una sección de texto al mismo tiempo que se está imprimiendo. Cada una de estas acciones está siendo realizada por su propio hilo. Se puede afirmar entonces, que multitarea basada en procesos es la imagen global, mientras que multitarea basada en hilos maneja los detalles.

La programación multihilos posibilita el escribir programas eficientes que hacen un uso máximo de la capacidad de procesamiento disponible en el sistema. Una manera en que los multihilos alcanzan esta meta es manteniendo el tiempo muerto en un mínimo.

En esta práctica se presentan dos secciones: en la primera se presenta el hilo principal que todo programa Java tiene y además, se presentan las dos maneras en que se puede poner a funcionar un hilo, con la clase *Thread* o implementando la interface *Runnable*; la segunda sección presenta un ejemplo en el que en una pestaña de un *jTabbedPane* se crean dos circunferencias sin el uso de hilos y en la otra pestaña, se crean las dos circunferencias usando hilos.

Correlación con los temas y subtemas del programa de estudio vigente

Se relaciona directamente con la Unidad IV: Programación Concurrente (Multihilos) y específicamente con los puntos 4.1 y 4.2 que en el temario son:

4.1. Concepto de hilo.

4.2. Comparación de un programa de flujo único contra uno de flujo múltiple

Material y equipo necesario

- Computadora o laptop.
- Software requerido: NetBeans 7.2 en adelante.

Metodología

Parte Uno. Maneras de activar hilos

El sistema multihilos de Java está construido sobre la clase *Thread* con sus métodos y la interface *Runnable*. Para crear un nuevo hilo el programa puede extender la clase *Thread* o bien, implementar la interface *Runnable*. Pero antes de ver estas dos opciones de creación de hilos empecemos trabajando con el hilo que todo programa de Java tiene: el hilo *main* o principal.

El hilo *main*

Cuando un programa en Java se ejecuta, un hilo empieza a correr inmediatamente. Este hilo es nombrado main thread o hilo principal del programa y se crea y ejecuta automáticamente. Este hilo es importante por dos razones:

Es el hilo sobre el que otros hilos “hijos” pueden ser derivados.

Con frecuencia, es el último hilo en terminar su ejecución dado que realiza varias tareas de cierre.

Aunque el hilo principal es creado automáticamente y arrancado en su ejecución, puede ser controlado por medio de un objeto *Thread*. Para hacerlo se obtiene primero la referencia al hilo main por medio del método *currentThread()*. Para ilustrar lo anterior se pide se reproduzca el ejemplo siguiente:

Paso 1. Crear proyecto nuevo con el nombre *Practica_9_1_1_CurrentThread*, tal como se ha ilustrado pero ahora dejando la casilla de la creación de la clase *main* marcada, para crear una aplicación en modo consola.

```
public class Practica_9_1_1_CurrentThread {
   public static void main(String[] args) {


   }
}
```

Paso 2. Dentro de la clase *main* teclear el código que se presenta a continuación:

```
public class Practica_9_1_1_CurrentThread {
   public static void main(String[] args) {
      Thread t =Thread.currentThread();
      System.out.println("Current Thread: "+ t);

      //cambiar el nombre del hilo
      t.setName("Mi Hilo");
      System.out.println("Current Thread: "+ t);

      try{
         for (int i = 5; i > 0; i--) {
            System.out.println(i);
            Thread.sleep(1000);
         }
      }catch(InterruptedException e){
         System.out.println("Main Thread Interrupted");
```

```
    }
  }
}
```

Resultado de la ejecución

```
Current Thread: Thread[main,5,main]
Current Thread: Thread[Mi Hilo,5,main]
5
4
3
2
1
```

Puede observarse como en el segundo mensaje ya aprece el nombre del hilo como Mi Hilo, dado que se cambió con la instrucción *Thread*.*setName*("MI Hilo"); además al ejecutar el programa puede verse el tiempo de retardo entre la impresión de los números.

Declaración de hilos

Como ya se dijo arriba, se pueden crear hilos de dos maneras en leguaje Java:

- Implementando la interface *Runnable*.
- Heredar (extend) de la clase *Thread* misma.

A continuación se mostrará un mismo ejercicio desarrollado dos veces, permitiendo que se vean los pasos en la creación de hilos y que se constate que efectivamente, se obtienen los mismos resultados en ambas formas de creación de hilos.

Implementando la interface *Runnable*

La manera más fácil de crear un hilo es creando una clase que implemente la interface *Runnable*. Esta interface abstrae una unidad de código ejecutable. Usted puede crear un hilo en cualquier objeto que implemente *Runnable*. Para implementar *Runnable*, una clase solo le basta implementar el método *run().*

Dentro de *run()*, se definirá el código que constituye el nuevo hilo. Es importante entender que *run()* puede llamar a otros métodos, usar otras clases, y declarar variables tal y como el main thread lo hace. La única diferencia es que *run()*

establece la entrada para un nuevo hilo de ejecución, que es concurrente dentro de su programa. Cuando *run()* se finaliza (returns), el hilo que inició, termina.

Después de crear la clase que implementa *Runnable*, instanciará un objeto de la clase *Thread* desde dentro de la clase. Aunque la clase *Thread* define varios constructores, el que mostraremos aquí es de la forma siguiente.

```
Thread(Runnable threadOb, String threadName)
```

Donde *threadOb* es una instancia de la clase que implementa *Runnable* y *threadName* es el nombre del nuevo hilo.

Una vez que un nuevo hilo es creado, se iniciará su ejecución usando el método *start()*, el cual hace una llamada al método run() del hilo.

En resumen:

```
class NewHilo implements Runnable{
    Thread t;  // instancia de la clase Thread
    // constructor para la clase donde se crea el hilo y se inicia
    NewHilo(){
      t = new Thread(this, "NuevoThread"); // el nombre es opcional
      t.start();   // es una llamada al método run() del hilo
     }
     public void run(){
      // aquí va el código que el hilo ejecutará
     }
}
class PruebaHilo {
   public static void main( String args[]) {
         NewHilo hilo1 = new NewHilo();
         . . . . . .
}
```

Ejemplo de creación de hilo implementando la interface *Runnable*

Paso 1. Crear un nuevo proyecto con nombre *Practica9_ThreadRunnable* cuidando de NO desmarcar la casilla de Create class *main*.

Paso 2. Teclear el código siguiente:

```
class NewThread implements Runnable{
   Thread t;

   public NewThread() {
      t = new Thread(this, "Demo Thread");
      System.out.println("Child Thread: " +t);
      t.start(); // start the thread
   }
   // this is the entry point for the second thread
   public void run(){
      try{
         for(int i = 5; i>0; i--){
            System.out.println("Child Thread: " +i);
            Thread.sleep(500);
         }
       }catch(InterruptedException e){
         System.out.println("Child Interrupted " );
       }
      System.out.println("Exiting child thread. " );

   }

}
public class Main {

   public static void main(String[] args) {
     new NewThread();
     try{
         for(int i = 5; i>0; i--){
```

```
            System.out.println("Main Thread: " +i);
            Thread.sleep(1100);
          }
        }catch(InterruptedException e){
            System.out.println("Main Thread Interrupted " );
        }
        System.out.println("Main Thread exiting " );
      }

}
```

La salida del programa puede ser como la que se muestra enseguida:

```
Child Thread: Thread[Demo Thread,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Child Thread: 3
Main Thread: 4
Child Thread: 2
Child Thread: 1
Exiting child thread.
Main Thread: 3
Main Thread: 2
Main Thread: 1
Main Thread exiting
```

Mismo programa ahora desarrollado con herencia de la clase *Thread*

La segunda manera de crear un hilo es creando una nueva clase que herede (***extends***) de la clase *Thread*, y luego crear una instancia de esta clase. La clase extendida debe sobrescribir el método *run()* el cual es el punto de entrada para el nuevo hilo. También ella debe llamar a *start()* para iniciar la ejecución del nuevo hilo. Enseguida se rescribe el mismo programa anterior aplicando esta manera de crear hilos.

```
class NewHilo extends Thread{

    // constructor para la clase
    NewHilo(){
        super("Nuevo Hilo");
        start();
     }
     public void run(){
       // aquí va el código que el hilo ejecutará
     }
}
class PruebaHilo {
   public static void main( String args[]) {
         NewHilo hilo1 = new NewHilo();
         . . . . . .
```

Paso 1. Crear un nuevo proyecto con nombre *Practica9_ThreadClass* cuidando de NO desmarcar la casilla de create main class.

Paso 2. Teclear el código siguiente:

```
class NewThread extends Thread{


   NewThread(){
      super("Nuevo Hilo");
      System.out.println("Child thread:"+ this);
      start(); // inicia el hilo
   }


   public void run(){
      try{
         for(int i=5;i>0;i--){
              System.out.println("Child thread: "+i);
              Thread.sleep(500*i);
```

```
        }
      }catch(InterruptedException e){
          System.out.println("Child Interrupted:");
      }
        System.out.println("Exiting Child thread.");
    }
}

public class Main {

   public static void main(String[] args) {
      NewThread obj = new NewThread();

        for(int i=5;i>0;i--){
            System.out.println("Main thread: "+i);
            Thread.sleep(1000);

        }
      }catch(InterruptedException e){
          System.out.println("Main Interrupted:");
      }
        System.out.println("Exiting Main thread.");
    }
   }
```

La salida es como la siguiente:

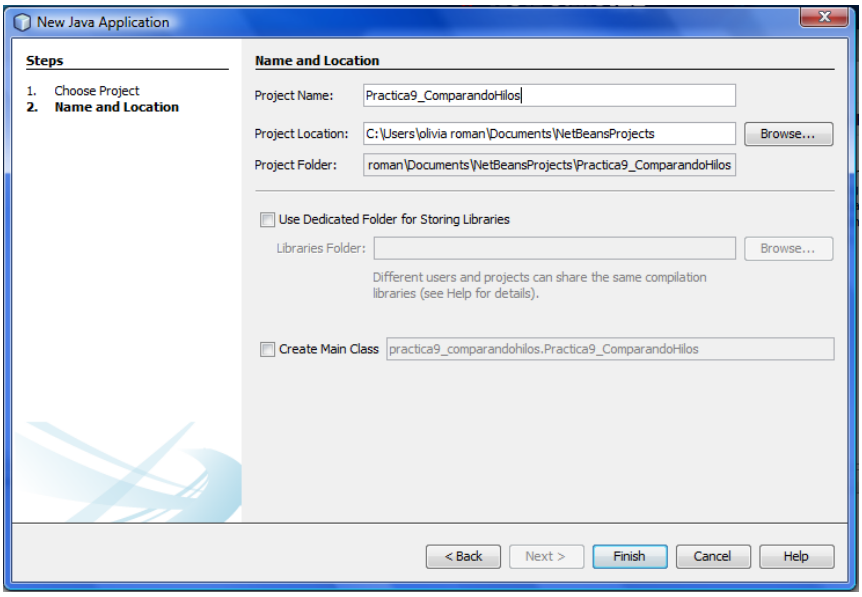
```
Child Thread: Thread[Nuevo Hilo,5,main]
Main Thread: 5
Child Thread: 5
Child Thread: 4
Child Thread: 3
Main Thread: 4
Child Thread: 2
Child Thread: 1
Exiting Child thread.
Main Thread: 3
```

Main Thread: 2
Main Thread: 1
Main Thread exiting.

Parte Dos. Comparación entre un programa de flujo único contra el mismo programa adaptado para el uso de hilos.

En esta sección se desarrollará una aplicación que utiliza un *TabbedPane* con dos pestañas. Para mostrar la diferencia entre los dos tipos de procesos, sin hilos y con hilos, se pintarán los radios de dos circunferencias comenzando con un ángulo de cero y luego irlos variando hasta completar los 360 grados. Este proceso se realiza en las dos pestañas. En la primera pestaña se ha puesto de manera secuencial el pintar los radios de las dos circunferencias. Se observará claramente como se pinta la primera circunferencia completamente y luego se procede a dibujar la segunda, reflejando un proceso secuencial debido al flujo único. En la segunda pestaña se inicializan dos hilos, de manera que cada uno se encarga de pintar los radios de una circunferencia y puede verse cómo se pintan de manera concurrente, "simultáneamente", gracias al apoyo de los hilos creados. Así pues se muestra visualmente un proceso concurrente en comparación con uno de flujo único.

Paso 1. Crear un proyecto nuevo con nombre *Practica9_ComparandoHiilos*. Asegúrese de desmarcar la casilla del Create Main Class.



Paso 2. Añadir un nuevo formulario con nombre *VentanaGraf*. Los pasos para realizar esta tarea ya se conocen.

Paso 3. Nuestra área de dibujo será un *JPanel* el cual primeramente será adaptarlo para que nos permita dibujar las circunferencias, así que crearemos una nueva clase que herede todo el comportamiento de la clase *JPanel*. Nos interesa crear un método para pintar dentro de este componente, que sea diferente al paint o repaint, ya que queremos pintar las líneas cuando se pulse un botón y no cuando se cree y aparezca el Panel. Es decir que si nos apoyáramos poniendo nuestro código en el método *paint()*, dado que éste es llamado al crearse el componente, se aparecería el panel YA con las líneas pintadas y no tendríamos oportunidad de ver cómo se comporta con hilos o sin hilos.

Así que empezamos por añadir una nueva clase a nuestro proyecto con nombre *PanelGraf*. Recuerde ir a la ventana de proyectos, usar el botón derecho, escoger new y luego Java Class. El código que nos aporta es el siguiente:

```
public class PanelGraf {



}
```

Paso 4. Hacemos que nuestra clase herede de *JPanel*. Escribir *extends JPanel*.

```
import javax.swing.JPanel;

public class PanelGraf extends JPanel{

}
```

Seguramente le marca en rojo la palabra *JPanel* debido a que no reconoce esta clase, para ello usted tiene varias maneras de corregirlo: escribir completo *javax.swing.JPanel* después de la palabra *extends*; escribir el *import javax.swing.JPanel* usted mismo; con botón derecho abrir menú contextual y escoger *Fix Imports*; y por último mover el cursor al botoncillo rojo en el margen izquierdo, hacer clic ahí y en la nueva ventana seleccionar *Add imports from javax.swing.JPanel*.

Paso 5. Sobrescribimos el método *paint()* del *JPanel* para hacer que tenga fondo blanco y sea el área de dibujo. Para ello hacer clic botón derecho dentro de las llaves de la clase *PanelGraf* y escoger *Add Code*, y enseguida buscamos la opción Override Method. Puede suceder que aparezca una ventana con las categorías en

la que están agrupados los métodos, si este es el caso, abra la categoría *JComponent* haciendo clic en el signo más que está a la izquierda, con lo que se mostrarán los métodos que ésta contiene. También puede suceder que aparezca esta última ventana desde el principio.

Empiece a teclear la palabra paint, y conforme avanza se irán mostrando los métodos que cumplan con empezar con las letras que lleve usted tecleadas. Localice el método *paint(Graphics)* y marque la casilla de la izquierda. Luego botón *Generate*.

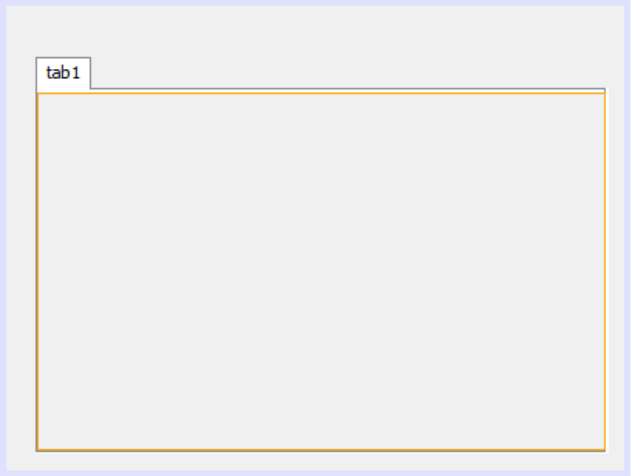
```
public class PanelGraf extends JPanel{


  @Override
  public void paint(Graphics g) {
    super.paint(g);
  }
}
```

Escriba en la siguiente línea debajo de *super.paint()*, la instrucción para darle color blanco al fondo del panel.

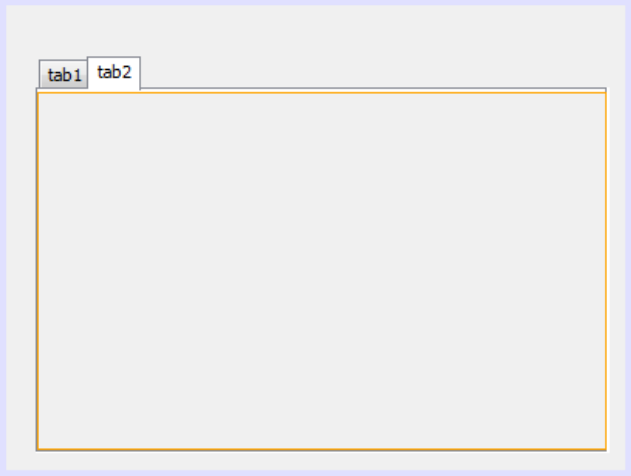
```
public class PanelGraf extends JPanel{


  @Override
  public void paint(Graphics g) {
    super.paint(g);
    setBackground(Color.white);
  }
 }
```

Paso 6. Dejamos de manera temporal el código de la clase *PanelGraf* para movernos al diseño del formulario *VentanaGraf*. Vamos a diseñar nuestra área de trabajo. Como vamos a trabajar de dos maneras nuestro programa, sin hilos y usando hilos, utilizaremos un *JTabbedPane* con dos pestañas, una para cada tipo de proceso. Las pestañas para que puedan recibir varios componentes, les añadiremos un Panel (el original) a cada una, aunque DE HECHO, es al momento de añadir cada panel al *TabbedPane*, que se crea cada pestaña. Realizado lo anterior debemos tener un resultado semejante a las imágenes siguientes:

Después de añadir un Panel



Después de añadir el segundo Panel



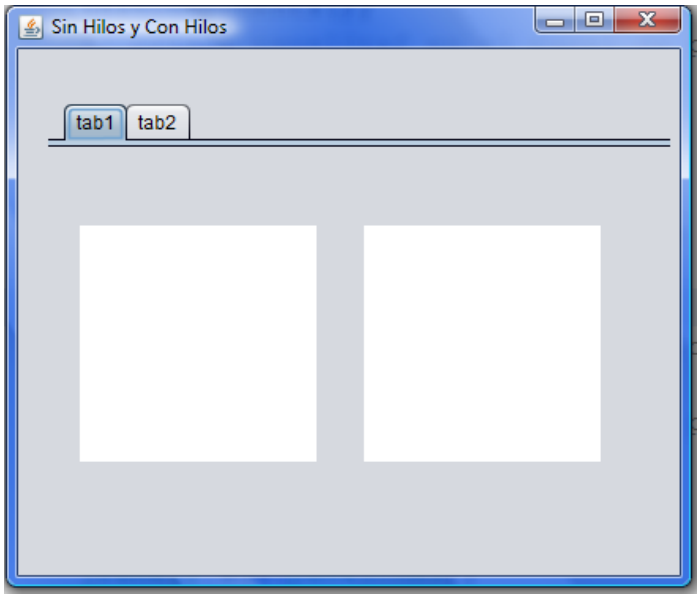
Paso 7. Ahora vamos a crear 4 instancias de nuestra clase *PanelGraf*, dos para cada pestaña del *TabbedPane*. Las vamos a crear, dimensionar, colocar y ubicar en la pestaña correcta todo por medio de instrucciones. Nos movemos a la ventana de código del formulario.

Escribir debajo de la primera línea de la clase, la declaración de nuestras cuatro instancias:

```
public class VentanaGraf extends javax.swing.JFrame {
  PanelGraf panelSin1;
  PanelGraf panelSin2;
  PanelGraf panelCon3;
  PanelGraf panelCon4;
```

Vamos al constructor para terminar de crearlas, escribimos el código siguiente:

```
public class VentanaGraf extends javax.swing.JFrame {
  PanelGraf panelSin1;
  PanelGraf panelSin2;
  PanelGraf panelCon3;
  PanelGraf panelCon4;


  public VentanaGraf() {
    // aprovechamos constructor del padre para poner título
    super("Sin Hilos y Con Hilos");
    initComponents();
    // primer panelGraf en el panel de pestaña uno
    panelSin1 = new PanelGraf();
    panelSin1.setSize(150, 150);
    panelSin1.setLocation(20, 50);
    jPanel1.add(panelSin1);
    // segundo panelGraf en el panel de pestaña uno
    panelSin2 = new PanelGraf();
    panelSin2.setSize(150, 150);
    panelSin2.setLocation(200, 50);  // colocado más a la derecha
    jPanel1.add(panelSin2);
    // tercer panelGraf en el panel de pestaña dos
    panelCon3 = new PanelGraf();
    panelCon3.setSize(150, 150);
    panelCon3.setLocation(20, 50);
    jPanel2.add(panelCon3);
    // cuarto panelGraf en el panel de pestaña dos
    panelCon4 = new PanelGraf();
    panelCon4.setSize(150, 150);
    panelCon4.setLocation(200, 50);  // colocado más a la derecha
    jPanel2.add(panelCon4);
  }...
```

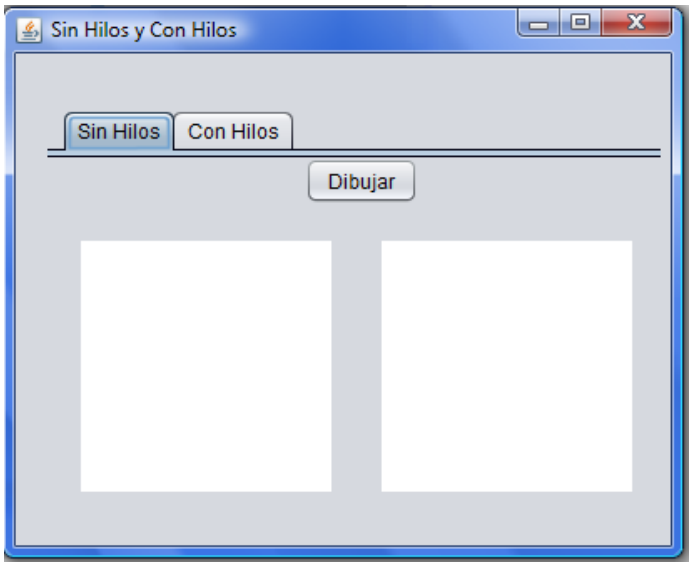


Al ejecutar el programa esto es lo que se ve en pestaña uno. La pestaña dos presenta los dos paneles que le corresponden.

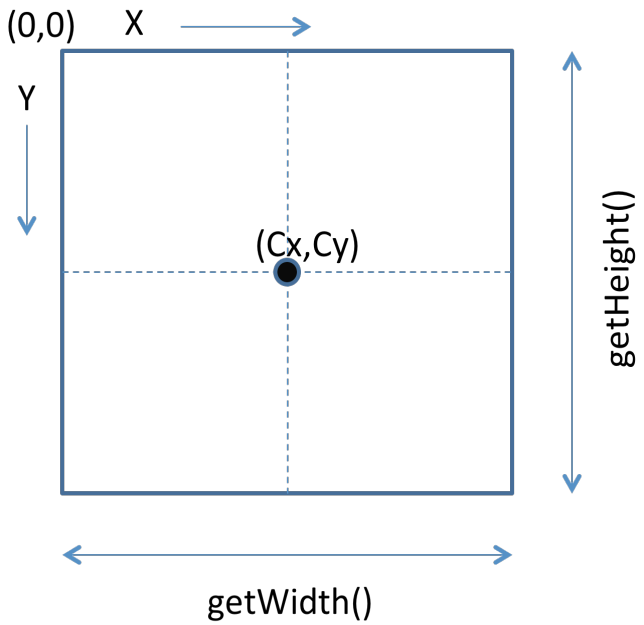
Paso 8. Añadimos un botón a cada pestaña, que será el que dé la orden de dibujar las líneas.

Arrastre un botón a la pestaña uno y colóquelo arriba de los dos paneles. Repita para la pestaña dos.

Aprovechar para cambiar el texto de cada pestaña: “Sin Hilos” y “Con Hilos”.



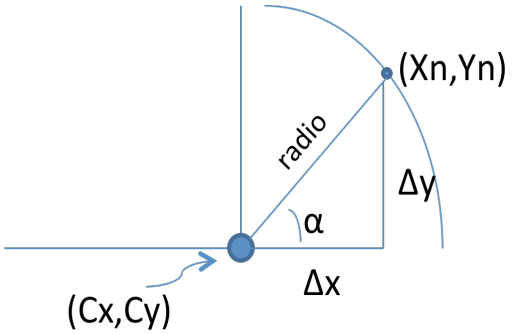
Paso 9. Vamos a desarrollar el método *pintar()* para la clase *PanelGraf*, en donde se plasmarán las instrucciones para pintar de manera secuencial las líneas que parten del centro del panel hacia el borde de una circunferencia, y que al ir variando el ángulo hasta el valor de 360 permite se aprecie la circunferencia, pero sin pintar explícitamente el contorno de la misma.



Para poder desarrollar nuestro código hay una serie de elementos que se deben considerar:

- El punto origen (0,0) se encuentra en la esquina superior izquierda.
- El valor de la coordenada **x** aumenta hacia la derecha.
- El valor de la ordenada **y** aumenta hacia abajo.
- El ancho del panel se obtiene por medio de la función *getWidth()* y el alto por medio de *getHeight()*.
- El centro de encuentra ubicado en Cx = getWidth() /2, Cy = getHeight() / 2.
- El radio de la circunferencia a dibujar será de la mitad del ancho del panel, radio = getWidth() /2.

Para calcular cualquier punto sobre el borde de la circunferencia, se recurre a trigonometría.



$$Xn = Cx + radio * \cos \alpha$$
$$Yn = Cy - radio * \sin \alpha$$

Observe que para calcular el punto (Xn,Yn) se calculan los valores de Δx y Δy.

Con las fórmulas para las funciones seno y coseno, y haciendo el despeje correspondiente, tenemos Δx = radio * cos( ) y Δy = radio * sin( )

Para encontrar el valor de Xn, el Δx se suma al valor de Cx ya que las x aumentan a la derecha (Xn = Cx + Δx). Para el caso del valor de Yn, cuando Δy tiene valor positivo, se debe encontrar un punto hacia arriba, por lo que deberá restarse de Cy porque de lo contrario se iría más abajo (Xy = Cy - Δy).

Con la información anterior ya estamos listos para escribir el código correspondiente y sobre todo poder entenderlo. El método *pintar()* de la clase *PanelGraf* requiere un elemento Graphics para poder llamar a los métodos de dibujo a partir de él. Se tomará el Graphics en el método *paint()* y se pondrá en una variable de nombre *gra* que podamos utilizar. Considerando lo dicho, el código queda de la siguiente manera:

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;



public class PanelGraf extends JPanel{
  Graphics gra;


  public void pintar(){
    int cx = getWidth()/2;
```

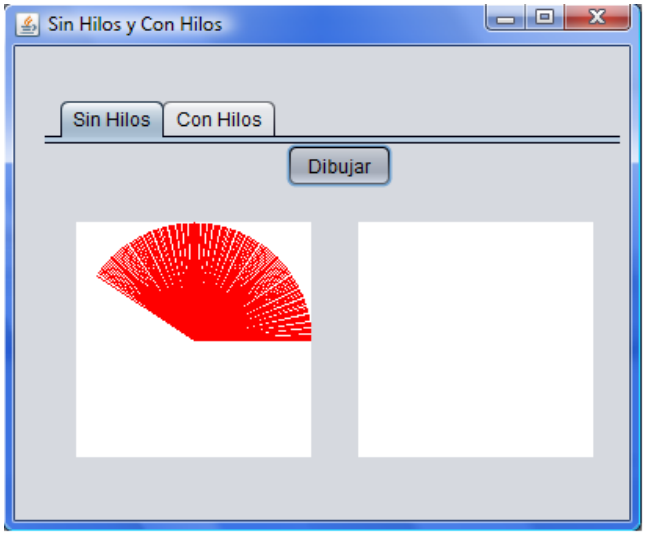
```
        int cy = getHeight()/2;
        // definimos el color rojo para las líneas
        gra.setColor(Color.red);
        int radio = getWidth()/2;
        // ciclo que varía el ángulo desde 0 hasta 360 y pinta líneas
        for(int ang=0; ang< 360; ang++){
            int xn = cx + (int)(radio * Math.cos(Math.toRadians(ang)));
            int yn = cy - (int)(radio * Math.sin(Math.toRadians(ang)));
            gra.drawLine(cx, cy, xn, yn);
            // ciclos vacíos para hacer tiempo, provocar retraso
            for(int i=0; i<100000;i++){
                for(int j =0;j< 100;j++){

                } // las llaves de los dos ciclos for vacíos no son necesarias
            }
        }
    }

    @Override
    public void paint(Graphics g) {
        super.paint(g);
        setBackground(Color.white);
        gra = getGraphics();
    }
}
```

Nota: Se pusieron dos ciclos *for* anidados para hacer que el dibujado fuera más lento. No tienen otra razón para estar ahí. Sin embargo moviendo el número límite de cualquiera de los dos ciclos, podemos hacer más lento o más rápido el pintado de cada figura.

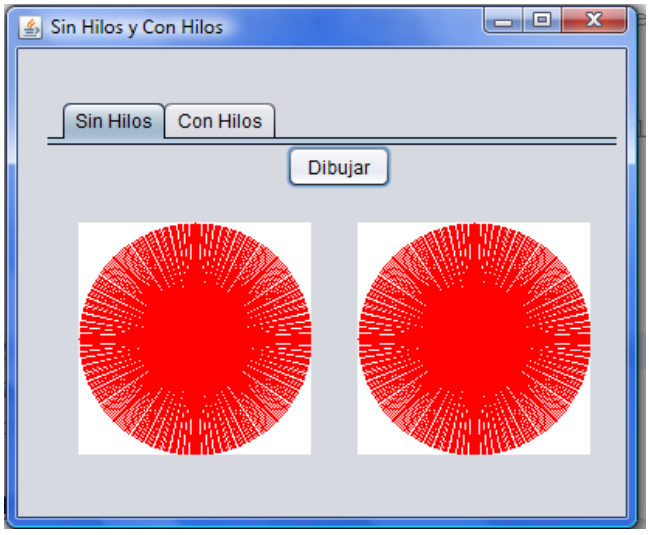
Paso 10. Ahora ya estamos listos para dibujar, para ello haremos que el botón de la primera pestaña le hable al método pintar de los dos objetos *PanelGraf* que pusimos ahí. Programamos el evento de *actionPerformed* del botón.

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
  panelSin1.pintar();
  panelSin2.pintar();

}
```

Ejecutamos el programa y al hacer clic en el botón dibujar del primer panel tenemos la siguiente imagen tomada durante la ejecución:



Al terminar la ejecución luce así:



Paso 11. A continuación procederemos a modificar la clase *PanelGraf* de manera que podamos trabajar hilos con ella. El mecanismo a usar es a través de la implementación de la interface *Runnable*, ya que hicimos que heredara de *JPanel* y por lo tanto es la única vía posible. Además declaramos una instancia de la clase *Thread* con nombre t y añadimos un constructor a la clase *VentanaGraf* donde podamos terminar con la declaración de t.

```
public class PanelGraf extends JPanel implements Runnable{
  Graphics gra;
  Thread t;

  public PanelGraf() {   // constructor
    super();  // constructor clase padre
    t = new Thread(this);
  }
```

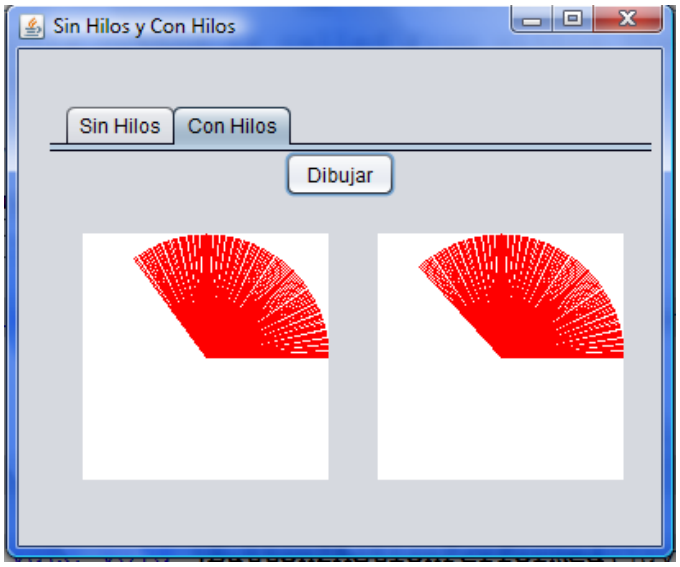
Con este código declaramos a t como un atributo de la clase, es decir que un hilo es un atributo. Cuando ese hilo se inicialice (start) ejecutará su método run y ahí es donde estará la acción que queremos se realice y es la llamada al método *pintar()*. Por lo tanto adecuamos el método run del hilo a lo siguiente:

```
 @Override
 public void run() {
   pintar();
 }
```

Paso 12. Estamos listos para hacer que los botones de la segunda pestaña en nuestro formulario *VentanaGraf*, ejecuten los hilos de los paneles tipo *PanelGraf* que declaramos. Al evento de acción del botón se le programa la llamada al start de los hilos.

```
 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
   panelCon3.t.start();
   panelCon4.t.start();
 }
```

Al ejecutar nuestro programa y pulsando el botón de la pestaña “Con Hilos” se obtiene el siguiente comportamiento:



Puede verse una diferencia significativa en cuanto a la forma de comportarse de ambos mecanismos de dibujo.

Sugerencias Didácticas

Primeramente se le pide al alumno que comprenda el tema de interfaces, qué es una interface, cómo se define, qué características tiene, para qué se usan, cómo se implementan y que desarrolle un par de ejercicios que usen interfaces; de esta manera tendrá las bases necesarias para comprender la creación de hilos implementando la interface *Runnable*.

De igual manera es importante que repase los conocimientos sobre herencia: qué es la herencia, cómo se hace uso de la herencia, cuando es necesaria, etc., etc.

Se le solicitará al alumno que encuentre las semejanzas y las diferencias entre las dos maneras de crear un hilo.

El alumno deberá repasar sus conocimientos de trigonometría para entender cómo el programa pinta los radios de las circunferencias.

Reporte del alumno (resultados)

El alumno deberá presentar evidencia de haber realizado el proyecto. Entregará además sus comentarios, sugerencias y el aprendizaje que le dejó esta práctica.

El alumno adecuará el método *run()* o el método *pintar()* para que en el caso del uso de hilos, no se utilicen los ciclos de retraso sino el método *sleep().*

Bibliografía preliminar

1. Java The Complete Reference Eighth Edition
   Herbert Schildt

   Oracle Press 2011

2. En la página oficial de Java:
   docs.oracle.com/javase/tutorial/java/