Un sistema operativo puede realizar multiprogramación al reasignar rápidamente tiempos de la CPU entre muchos programas, dando el aspecto de paralelismo, al ejecutarse concurrentemente. Aunque el verdadero paralelismos se logra con una computadora con varias CPU.

Java soporta varios hilos de ejecución. Un proceso de Java puede crear y manejar, dentro de sí mismo, varias secuencias de ejecución concurrentes o paralelos. Cada una de estas secuencias es un hilo independiente y todos ellos comparten tanto el espacio de dirección como los recursos del sistema operativo. Por lo tanto, cada hilo puede acceder a todos los datos y procedimientos del proceso, pero tiene su propio contador de programa y su pila de llamadas a métodos.

Todos los programas que hemos aprendido a desarrollar se ejecutan en forma secuencial, por ejemplo cuando llamamos a un método desde la main hasta que esta no finalice no continúan ejecutándose las instrucciones de la main.

Esta forma de resolver los problemas en muchas situaciones no será la más eficiente. Imaginemos si implementamos un algoritmo que busca en el disco duro la cantidad de archivos con extensión java, éste proceso requiere de mucho tiempo ya que el acceso a disco es costoso. Mientras esta búsqueda no finalice nuestro programa no puede desarrollar otras actividades, por ejemplo no podría estar accediendo a un servidor de internet, procesando tablas de una base de datos etc.

En el lenguaje Java se ha creado el concepto de hilo para poder ejecutar algoritmos en forma concurrente, es decir que comience la ejecución de la función pero continúe con la ejecución de la función main o la función desde donde se llamó al hilo.
Con los hilos podremos sacar ventajas en seguir la ejecución del programa y que no se bloquee en algoritmos complejos o que accedan a recursos lentos.

Otra gran ventaja con el empleo de los hilos es que los computadores actuales tienen múltiples procesadores y podremos ejecutar en esos casos hilos en forma paralela, con esto nuestros programas se ejecutarán mucho más rápido.
Los primeros problemas que resolvamos con hilos tienen por objetivo entender su creación y uso, más allá de su utilidad real.

Problema:

Mostrar el número "0" mil veces y el número "1" mil veces. Utilizar la programación secuencial vista hasta ahora.

```java
public class MostrarCeroUno {

    public void mostrar0() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("0-");
    }

    public void mostrar1() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("1-");
    }

    public static void main(String[] args) {
        MostrarCeroUno m=new MostrarCeroUno();
        m.mostrar0();
        m.mostrar1();
    }
}
```

Problema:
Mostrar el número "0" mil veces y el número "1" mil veces. Hacer la impresión de las listas de valores en dos hilos.

```java
public class MostrarCeroUnoHilo {

    public static void main(String[] args) {
        HiloMostrarCero h1 = new HiloMostrarCero();
        h1.start();
        HiloMostrarUno h2 = new HiloMostrarUno();
        h2.start();
    }
}

class HiloMostrarCero extends Thread {

    @Override
    public void run() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("0-");
    }
}

class HiloMostrarUno extends Thread {

    @Override
    public void run() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("1-");
    }
}
```

Segunda forma de crear un hilo.

Recién vimos que podemos crear un hilo planteando una clase que herede de la clase 'Thread'. Disponemos de una segunda forma de crear hilos en Java, debemos definir un objeto de la clase Thread y una clase que implemente la interface 'Runnable'. El programa anterior con esta otra metodología queda:

```java
public class MostrarCeroUnoHilo {

    public static void main(String[] args) {
        HiloMostrarCero h1 = new HiloMostrarCero();
        HiloMostrarUno h2 = new HiloMostrarUno();
    }
}

class HiloMostrarCero implements Runnable {
    private Thread t;

    public HiloMostrarCero() {
        t = new Thread(this);
        t.start();
    }

    @Override
    public void run() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("0-");
    }
}

class HiloMostrarUno implements Runnable {
    private Thread t;

    public HiloMostrarUno() {
        t = new Thread(this);
        t.start();
    }

    @Override
    public void run() {
        for (int f = 1; f <= 1000; f++)
            System.out.print("1-");
    }
}
```

Problema:
Desarrollar un programa que cargue en un objeto de la clase 'JTextArea' todos los directorios y archivos de la unidad "C". Como podemos imaginar esta es una actividad larga y lenta, por lo que si no la hacemos dentro de un hilo nuestro programa no responderá a las instrucciones del operador durante varios minutos.
Dispondremos de un botón para finalizar el programa en cualquier momento.

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class FormularioBusqueda extends JFrame implements ActionListener {
    JTextArea textarea1;
    JScrollPane scrollpane1;
    JButton boton1;

    FormularioBusqueda() {
        setLayout(null);
        textarea1 = new JTextArea();
        scrollpane1 = new JScrollPane(textarea1);
        scrollpane1.setBounds(10, 30, 760, 300);
        add(scrollpane1);

        boton1 = new JButton("Salir");
        boton1.addActionListener(this);
        boton1.setBounds(320, 350, 100, 30);
        add(boton1);

        textarea1.setText("");
        HiloBusqueda hb = new HiloBusqueda("c:\\", textarea1);
        hb.start();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boton1)
            System.exit(0);

    }

    public static void main(String[] arguments) {
        FormularioBusqueda fb;
        fb = new FormularioBusqueda();
        fb.setBounds(0, 0, 800, 640);
        fb.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        fb.setVisible(true);
    }
}

class HiloBusqueda extends Thread {
    String directorio;
    JTextArea ta;

    public HiloBusqueda(String directorio, JTextArea ta) {
        this.directorio = directorio;
        this.ta = ta;
    }

    @Override
    public void run() {
        leer(directorio);
    }

    private void leer(String inicio) {

        File ar = new File(inicio);
        String[] dir = ar.list();
        if (dir != null)
            for (int f = 0; f < dir.length; f++) {
                File ar2 = new File(inicio + dir[f]);
                if (ar2.isFile())
                    ta.append(inicio + dir[f] + "\n");
                if (ar2.isDirectory()) {
                    ta.append(inicio + dir[f].toUpperCase() + " -->
[Directorio]\n");
                    leer(inicio + dir[f] + "\\");
                }

            }
    }
}
```

Si codificamos el mismo programa sin emplear el concepto de hilos podremos comprobar que el JFrame no aparece en pantalla hasta que finaliza el recorrido completo del disco duro (recorremos la carpeta 'c:\windows' en lugar de 'c:\' para que no demore tanto tiempo):

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;

public class FormularioBusqueda extends JFrame implements ActionListener {
    JTextArea textarea1;
    JScrollPane scrollpane1;
    JButton boton1;

    FormularioBusqueda() {
        setLayout(null);
        textarea1 = new JTextArea();
        scrollpane1 = new JScrollPane(textarea1);
        scrollpane1.setBounds(10, 30, 760, 300);
        add(scrollpane1);

        boton1 = new JButton("Salir");
        boton1.addActionListener(this);
        boton1.setBounds(320, 350, 100, 30);
        add(boton1);

        textarea1.setText("");
        Busqueda hb = new Busqueda("c:\\windows\\", textarea1);
        hb.iniciar();
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == boton1)
            System.exit(0);

    }

    public static void main(String[] arguments) {
        FormularioBusqueda fb;
        fb = new FormularioBusqueda();
        fb.setBounds(0, 0, 800, 640);
        fb.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        fb.setVisible(true);
    }
}

class Busqueda {
    String directorio;
    JTextArea ta;

    public Busqueda(String directorio, JTextArea ta) {
        this.directorio = directorio;
        this.ta = ta;
    }

    public void iniciar() {
        leer(directorio);
    }

    private void leer(String inicio) {

        File ar = new File(inicio);
        String[] dir = ar.list();
        if (dir != null)
            for (int f = 0; f < dir.length; f++) {
                File ar2 = new File(inicio + dir[f]);
                if (ar2.isFile())
                    ta.append(inicio + dir[f] + "\n");
                if (ar2.isDirectory()) {
                    ta.append(inicio + dir[f].toUpperCase() + " -->
[Directorio]\n");
                    leer(inicio + dir[f] + "\\");
                }

            }
    }
}
```

Problema:
Crear un vector con quinientos millones de elementos con valores aleatorios. Proceder a buscar el mayor sin el empleo de hilos, luego buscar el mayor pero utilizando dos hilos, uno que busque el mayor en la primera parte del vector y otro que busque en la segunda parte. Mostrar la cantidad de tiempo consumido en la búsqueda del mayor con hilos y sin hilos.

```java
import java.util.Date;

public class MayorVector {

    public static void main(String[] args) {
        System.out.println("Cantidad de núcleos del procesador:" +
Runtime.getRuntime().availableProcessors());
        int[] v = new int[500_000_000];
        System.out.println("Inicio de la carga del vector.");
        for (int f = 0; f < v.length; f++)
            v[f] = (int) (Math.random() * 2_000_000_000);
        System.out.println("Fin de la carga del vector.");
        Date d1 = new Date();

        HiloMayor hilo1 = new HiloMayor();
        hilo1.fijarRango(0, v.length / 2, v);
        HiloMayor hilo2 = new HiloMayor();
        hilo2.fijarRango(v.length / 2 + 1, v.length - 1, v);
        hilo1.start();
        hilo2.start();

        while (hilo1.isAlive() || hilo2.isAlive()) ;

        System.out.print("Mayor elemento del vector:");
        if (hilo1.may > hilo2.may)
            System.out.println(hilo1.may);
        else
            System.out.println(hilo2.may);
        Date d2 = new Date();
        long milisegundos = (d2.getTime() - d1.getTime());

        System.out.println("Milisegundos requeridos con 2 hilos:" + milisegundos);

        d1 = new Date();
        int may = v[0];
        for (int f = 1; f < v.length; f++) {
            if (v[f] > may)
                may = v[f];
        }
        System.out.println("Mayor elemento del vector:" + may);
        d2 = new Date();
        milisegundos = (d2.getTime() - d1.getTime());
        System.out.println("Milisegundos requeridos sin hilos:" + milisegundos);

    }
}

class HiloMayor extends Thread {
    int[] v;
    int ini, fin;
    int may;

    void fijarRango(int i, int f, int[] v) {
        this.ini = i;
        this.fin = f;
        this.v = v;
    }

    public void run() {
        may = v[ini];
        for (int f = ini + 1; f < fin; f++) {
            if (v[f] > may)
                may = v[f];
        }
    }
}
```

Si la computadora tiene más de un núcleo podemos comprobar que la búsqueda del mayor requiere menos tiempo al emplear dos hilos que se ejecutan en forma paralela:

Los métodos sincronizados en Java solo pueden tener un hilo ejecutándose dentro de ellos al mismo tiempo.
Son necesarios cuando un método accede a un recurso que puede ser consumido por un único proceso.
Para evitar que un algoritmos sea ejecutado por más de un hilo en forma simultanea, Java nos permite definir un método con el modificador: synchronized.
Cuando un método se lo define synchronized luego solo un hilo puede estar ejecutándolo en un mismo momento.

Problema:

Confeccionar un programa que imprima la cantidad de archivos de texto de un directorio que contienen una determinada palabra. Efectuar la búsqueda dentro de un hilo.
Lanzar la búsqueda de dos palabras en dos hilos separados.

```java
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class BuscarPalabra implements Runnable {
    private String palabra;
    private Thread hilo;
    private int cant;

    public BuscarPalabra(String palabra) {
        this.palabra = palabra;
        hilo = new Thread(this);
        hilo.start();
        while (hilo.isAlive())
            ;
        System.out.println("La palabra " + palabra + " se encuentra contenida en " +
                           cant + " archivos");
    }

    @Override
    public void run() {
        File ar = new File("C:\\documentos\\");
        String[] directorio = ar.list();
        for (String arch : directorio) {
            if (tiene(arch))
                cant++;
        }
    }

    private synchronized boolean tiene(String archivo) {
        boolean existe = false;
        try {
            FileReader fr = new FileReader("c:\\documentos\\" + archivo);
            BufferedReader br = new BufferedReader(fr);
            String linea = br.readLine();
            while (linea != null) {
                if (linea.indexOf(palabra) != -1)
                    existe = true;
                linea = br.readLine();
            }
            br.close();
            fr.close();
        } catch (IOException e) {
            System.out.println(e.toString());
        }
        return existe;
    }

    public static void main(String[] ar) {
        new BuscarPalabra("rojo");
        new BuscarPalabra("verde");

    }

}
```