Concurrencia[1]

Los usuarios de computador la utilizan para permitir que sus sistemas puedan hacer mas de una operación al tiempo. Ellos asumen que pueden continuar trabajando en procesador de palabras (como Microsoft Word u Open Office), mientras otras aplicaciones descargan archivos, gestionan la cola de impresión y reproducen música desde internet a través de streaming. Aún, se espera a menudo, que una sola aplicación haga mas de una sola operación al tiempo. Por ejemplo, la aplicación de reproducción de música por streaming debe simultáneamente leer el audio digital desde la red, descomprimirlo y reproducirlo. Y el procesador de palabras debería siempre estar listo para responder a los eventos del teclado y el mouse, sin importar que tan ocupado se encuentre reformateando texto o actualizando el despliegue de dicho texto en pantalla. El software que puede hacer esas cosas es conocido como software concurrente.

La plataforma Java está diseñada desde la base para soportar programación concurrente, con soporte a concurrencia básica en el lenguaje de programación y las bibliotecas de clases de Java. Desde la versión 5.0, la plataforma Java ha incluido también APIs de concurrencia de alto nivel, sin embargo estas no serán cubiertas en el presente texto al considerarse fuera del alcance de este curso. Esta lección presenta el soporte de concurrencia básico de la plataforma.

Proceso e Hilos

En programación concurrente hay dos unidades básicas de ejecución: proceso e hilos. En el lenguaje de programación Java, la programación concurrente se refiere principalmente a hilos. Sin embargo, los procesos también son importantes.

Un sistema de computador normalmente tiene muchos procesos activos e hilos. Esto es cierto incluso en sistemas que solo tienen un simple núcleo de ejecución (un solo procesador de un solo núcleo), y por tanto solo tienen un hilo realmente ejecutándose en cualquier momento dado. El tiempo de procesamiento de un solo núcleo es compartido entre procesos e hilos a través de una característica del sistema operativo llamada segmentación de tiempo.

Es cada vez mas y mas común que los sistemas de cómputo tengan múltiples procesadores o procesadores con múltiples núcleos de ejecución. Esto mejora enormemente la capacidad del sistema para ejecución concurrente de procesos e hilos -pero la concurrencia es posible aún en sistemas simples, sin múltiples procesadores o núcleos de ejecución.

Procesos

Un proceso tiene un ambiente de ejecución autocontenido. Un proceso generalmente tiene un conjunto completo y privado de recursos básicos de ejecución; en particular, cada proceso tiene su propio espacio de memoria.

Los procesos son a menudo vistos como sinónimos de programas o aplicaciones. Sin embargo, lo que el usuario ve como una sola aplicación puede de hecho ser un conjunto de procesos cooperando. Para facilitar la comunicación entre procesos, la mayoría de sistemas operativos soportan recursos de Comunicación Inter Proceso (IPC, por sus siglas en inglés), tales como pipes y sockets. IPC es usado no solo para comunicación entre procesos sobre el mismo sistema, sino también entre procesos sobre diferentes sistemas.

[1] Traducción libre de Lesson: Concurrency. The Java Tutorials. https://docs.oracle.com/javase/tutorial/essential/concurrency/

La mayoría de implementaciones de la máquina virtual de Java corre sobre un solo proceso. Una aplicación Java puede crear procesos adicionales usando el objeto ProcessBuilder. Las aplicaciones multiprocesos están más allá del alcance del presente texto.

Hilos

Los hilos son algunas veces llamados procesos livianos. Tanto procesos como hilos proporcionan un ambiente de ejecución, pero crear un nuevo hilo requiere menos recursos que crear un nuevo proceso.

Los hilos existen dentro de un proceso -cada proceso tiene al menos un hilo. Los hilos comparten recursos dentro de un proceso, incluyendo memoria y archivos abiertos. Esto hace que la comunicación entre hilos sea más eficiente que entre procesos pero potencialmente problemática.

La ejecución multihilo es una característica esencial de la plataforma Java. Cada aplicación tiene al menos un hilo -o vario, si usted cuenta los hilos del "sistema" que llevan a cabo operaciones como gestión de memoria (como el recolector de basura de Java) o manejo de señales. Sin embargo, desde el punto de vista del programador, usted inicia con solo un hilo, llamado el hilo principal. Este hilo tiene la posibilidad de crear hilos adicionales, como se mostrará a continuación.

Objetos Hilo

Cada hilo está asociado con una instancia de la clase Thread. Hay dos estrategias básicas para usar objetos hilo para crear una aplicación concurrente:

- Directamente controlar la creación y gestión del hilo, simplemente instanciando la clase Thread cada vez que la aplicación necesite iniciar una tarea asíncrona.
- Abstraer la gestión de hilos del resto de la aplicación y pasar las tareas de la aplicación a un ejecutor.

En el presente texto discutiremos cómo utilizar objetos hilo a través de Thread. Los ejecutores hacen parte de los objetos de concurrencia de alto nivel de Java que mencionamos anteriormente y establecimos que se encontraba fuera del alcance del presente texto[2].

Definiendo e Iniciando un Hilo

Una aplicación que crea una instancia de Thread debe indicar cuál será el código que se ejecutará en el hilo. Hay dos formas de hacer esto. Lo ilustraremos con un sencillo ejemplo que primero será mostrado sin hilos y luego con cada una de las dos formas de trabajar con objetos hilo.

Sencillo Ejemplo (Cuenta Múltiplos) Sin Hilos

Tenemos en el dominio del problema una clase que representa a personas que les gusta contar todos los múltiplos de un número dado desde 1 hasta una constante MAX que tiene la misma clase. Para el ejemplo trabajaremos con tres personas: Juan que le gusta contar múltiplos de 5, Victor que le gusta contar múltiplos de 7 y Ángela que le gusta contar múltiplos de 2. El programa imprimirá en consola cuántos múltiplos encontró cada uno entre 1 y MAX. El diseño del diagrama de clases de la solución sin hilos es la siguiente:

[2] Para mayor información sobre manejo de objetos de concurrencia de alto nivel revisar High Level Concurrency Objects en The Java Tutorials: https://docs.oracle.com/javase/tutorial/essential/concurrency/highlevel.html

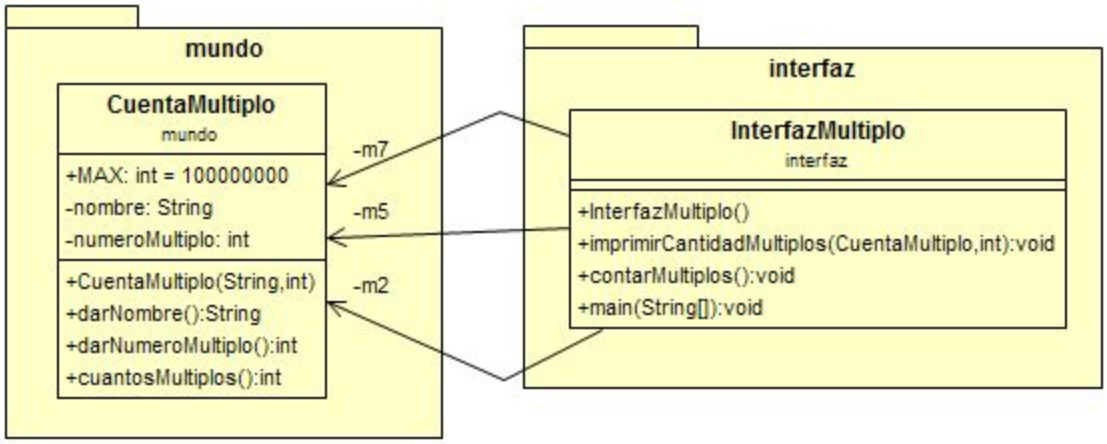

Diagrama de Clase del Cuenta Múltiplos sin Hilos

Y la implementación en Java se presenta a continuación:

```java
package mundo;

public class CuentaMultiplo {
    public static final int MAX = 100000000;
    private String nombre;
    private int numeroMultiplo;

    public CuentaMultiplo(String nom, int nm){
        nombre = nom;
        numeroMultiplo = nm;
    }

    public String darNombre(){
        return nombre;
    }

    public int darNumeroMultiplo(){
        return numeroMultiplo;
    }

    public int cuantosMultiplos(){
        int cuantosM = 0;
        for (int i = 1; i <= MAX; i++) {
            if(i%numeroMultiplo==0){
                cuantosM++;
            }
        }
        return cuantosM;
    }
}
```

```java
package interfaz;

import mundo.CuentaMultiplo;

public class InterfazMultiplo {

    private CuentaMultiplo m5;
    private CuentaMultiplo m7;
    private CuentaMultiplo m2;

    public InterfazMultiplo(){
        m5 = new CuentaMultiplo("Juan", 5);
        m7 = new CuentaMultiplo("Victor", 7);
        m2 = new CuentaMultiplo("Ángela", 2);
    }

    public void imprimirCantidadMultiplos(CuentaMultiplo m, int cantidadM){
        System.out.println(m.darNombre()+" cuenta "+cantidadM
                +" múltiplos de "+m.darNumeroMultiplo());
    }

    public void contarMultiplos(){
        int cuantos5 = m5.cuantosMultiplos();
        imprimirCantidadMultiplos(m5, cuantos5);

        int cuantos7 = m7.cuantosMultiplos();
        imprimirCantidadMultiplos(m7, cuantos7);

        int cuantos2 = m2.cuantosMultiplos();
        imprimirCantidadMultiplos(m2, cuantos2);
    }

    public static void main(String[] args){
        InterfazMultiplo interfaz = new InterfazMultiplo();
        interfaz.contarMultiplos();
    }
}
```

Implementación en Java del Cuenta Múltiplos sin Hilos

Y la salida del anterior programa es SIEMPRE la siguiente:

```
<terminated> InterfazMultiplo [Java Application] C:\
Juan cuenta 20000000 múltiplos de 5
Victor cuenta 14285714 múltiplos de 7
Ángela cuenta 50000000 múltiplos de 2
```

Ejemplo (Cuenta Múltiplos) con Hilos

Heredar de la Clase Thread

La primera forma de utilizar objetos hilo es heredar de la clase Thread. La clase Thread implementa la interface Runnable que obliga a implementar el método +run():void, sin embargo, aunque la clase Thread tiene este método declarado, no incluye ningún código en su interior, por lo que este método no hace nada.

Cuando hacemos que una clase de nuestro programa herede de Thread estamos en el deber de sobrescribir su método run para incluir allí el código que deseamos ejecutar en el hilo.

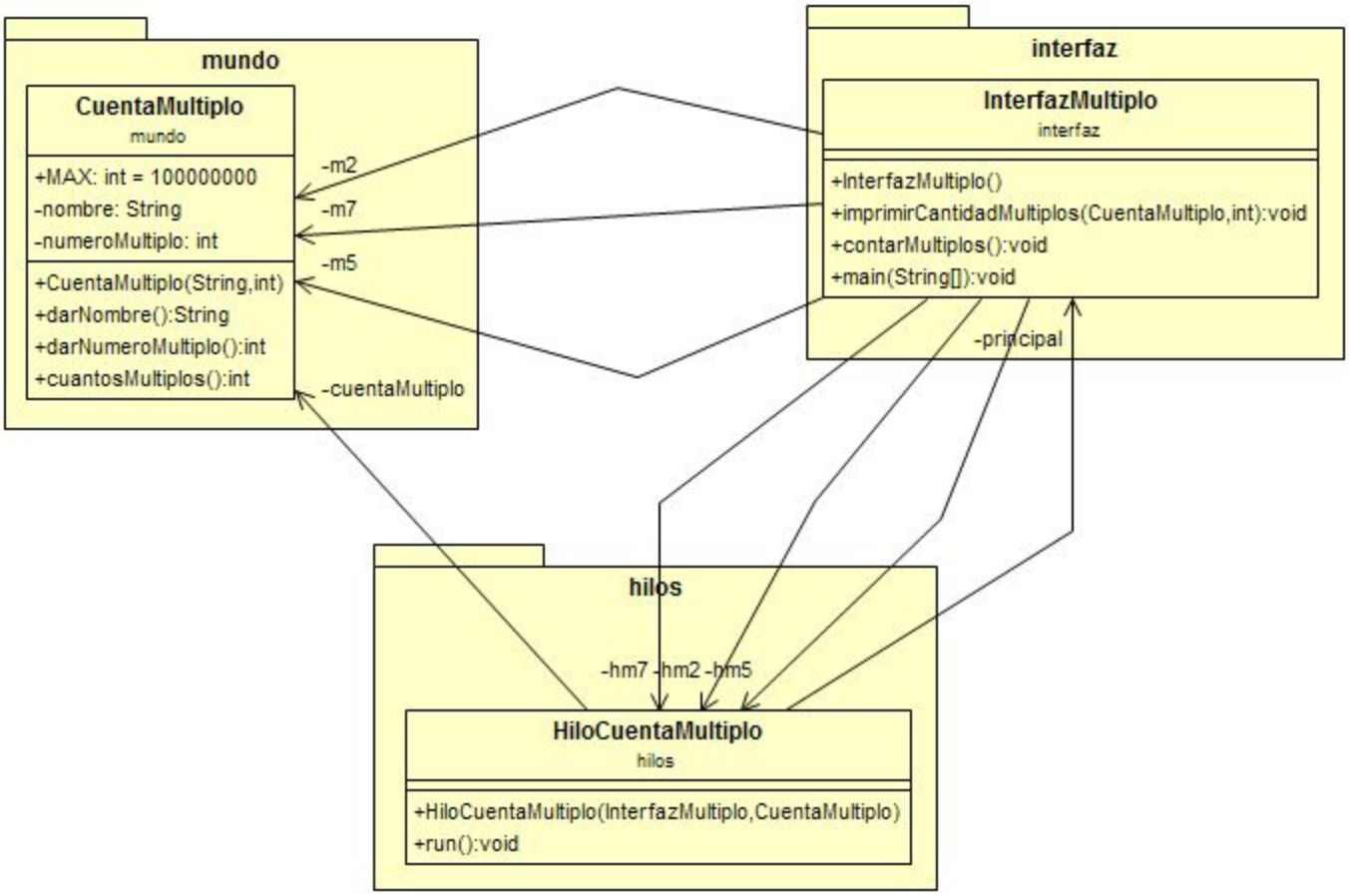

Diagrama de Clase del Cuenta Múltiplos con Hilos (extendiendo de Thread)

Si comparamos el diagrama inmediatamente anterior con el del ejemplo sin hilos podremos notar que la(s) clase(s) en el mundo no se modifican. Nace una nueva clase cuyo nombre tiene como prefijo Hilo (por convención) que tiene una relación obligatoria de multiplicidad 1 hacia una clase del mundo. Este hilo controla directamente el comportamiento de la clase CuentaMultiplo, por lo que se creará un objeto de esta clase hilo por cada objeto de la clase del mundo. Esto último se evidencia en las relaciones desde la clase InterfazMultiplo hacia la clase HiloCuentaMultiplo cuya cantidad es igual que las que se tienen hacia el mundo.

Adicionalmente, y con el objetivo de desplegar al usuario los resultados de la ejecución de los hilos, la clase del hilo tiene una relación hacia la clase principal de la interfaz.

A continuación se muestra la implementación en Java de la clase HiloCuentaMultiplo que es la nueva clase que hereda de Thread (y sobrescribe su método run) y la clase InterfazMultiplo que ahora debe modificarse para crear los hilos. Se omite la clase del mundo porque esta no fue modificada.

```java
package hilos;

import mundo.CuentaMultiplo;
import interfaz.InterfazMultiplo;

public class HiloCuentaMultiplo extends Thread{

    private CuentaMultiplo cuentaMultiplo;
    private InterfazMultiplo principal;

    public HiloCuentaMultiplo(InterfazMultiplo ppal,
            CuentaMultiplo cm){
        principal = ppal;
        cuentaMultiplo = cm;
    }

    public void run(){
        int cantidadMultiplo =
                cuentaMultiplo.cuantosMultiplos();

        principal.imprimirCantidadMultiplos(
                cuentaMultiplo, cantidadMultiplo);
    }
}
```

```java
package interfaz;
import mundo.CuentaMultiplo;
public class InterfazMultiplo {
    private CuentaMultiplo m5;
    private CuentaMultiplo m7;
    private CuentaMultiplo m2;

    private HiloCuentaMultiplo hm5;
    private HiloCuentaMultiplo hm7;
    private HiloCuentaMultiplo hm2;

    public InterfazMultiplo(){
        m5 = new CuentaMultiplo("Juan", 5);
        m7 = new CuentaMultiplo("Víctor", 7);
        m2 = new CuentaMultiplo("Ángela", 2);

        hm5 = new HiloCuentaMultiplo(this, m5);
        hm7 = new HiloCuentaMultiplo(this, m7);
        hm2 = new HiloCuentaMultiplo(this, m2);
    }

    public void imprimirCantidadMultiplos(CuentaMultiplo m,
            int cantidadM){
        System.out.println(m.darNombre()+" cuenta "+cantidadM
                +" múltiplos de "+m.darNumeroMultiplo());
    }

    public void contarMultiplos(){
        hm5.start();
        hm7.start();
        hm2.start();
    }

    public static void main(String[] args){
        InterfazMultiplo interfaz = new InterfazMultiplo();
        interfaz.contarMultiplos();
    }
}
```

Implementación en Java del Cuenta Múltiplos con Hilos (extendiendo de Thread)

Si revisamos la clase principal de la interfaz, lo primero que notamos es que aparecen unos nuevos objetos que son las instancias de la clase hilo. Estos objetos son creados en el constructor de la interfaz y enlazados cada uno a su respectivo objeto y a la interfaz que los está creando, tal como se discutió en el diseño. Es importante destacar que hasta ese momento, aunque los objetos de la clase hilo (que extiende de Thread) existen, la ejecución de los hilos de forma concurrente no ha iniciado.

La creación del hilo como unidad de ejecución en el sistema operativo y su ejecución inicia únicamente cuando al hilo se le invoca el método start, lo cual ocurre en el método contarMultiplos de nuestro ejemplo. Podríamos resumir que lo ocurre con la invocación del método start son las siguientes dos cosas: creación del hilo como nueva unidad de ejecución e invocación del método run de ese hilo para que se lleve a cabo todo el código definido en la sobrescritura de ese método en nuestra clase que extiende de Thread. El método start no es bloqueante, es decir, nunca deja bloqueado la ejecución del hilo actual sino que luego de su invocación (en la cual se crea el hilo), se da paso a la instrucción inmediatamente siguiente.

Ya que la ejecución de los cuenta múltiplos se ejecuta de forma concurrente. Si vemos el método contarMultiplos de la clase principal de la interfaz observaremos que luego del llamado a start sobre hm5, inmediatamente se ejecuta el start de hm7 y acto seguido hm2. El cuenta múltiplos en hm7 no esperará a que hm5 termine, ya que están en hilos diferentes, cada uno tendrá su propia secuencia de ejecución. De igual manera para hm2. Las ejecuciones de estos hilos se están realizando de forma independiente unos de otros.

Por lo anterior, las salidas de este programa no siempre son iguales, ya que cualquiera de los tres podría terminar primero, lo mismo el segundo y el tercero. Algunas de las posibles salidas se presentan a continuación, luego de que el programa se ejecutará en varias oportunidades:

```
<terminated> InterfazMultiplo (1) [Java Application]
Juan cuenta 20000000 múltiplos de 5
Ángela cuenta 50000000 múltiplos de 2
Victor cuenta 14285714 múltiplos de 7
```

```
<terminated> InterfazMultiplo (1) [Java Application]
Victor cuenta 14285714 múltiplos de 7
Ángela cuenta 50000000 múltiplos de 2
Juan cuenta 20000000 múltiplos de 5
```

```
<terminated> InterfazMultiplo (1) [Java Application]
Ángela cuenta 50000000 múltiplos de 2
Victor cuenta 14285714 múltiplos de 7
Juan cuenta 20000000 múltiplos de 5
```

Implementar la Interface Runnable

Ahora veremos la segunda forma de utilizar objetos hilo. Esta se lleva a cabo implementando la interface Runnable por la clase que incluirá el código a ser ejecutado en un hilo. En esta forma, la clase Thread sigue siendo necesaria, y lo que se requiere es que luego de ser creado el objeto de la clase que implementa Runnable, luego se cree un objeto de la clase Thread al cual se le pasa por parámetro en el constructor el objeto creado de la clase que que implementa Runnable y finalmente se invoca el método start del objeto creado de la clase Thread.

La interface Runnable obliga a implementar el método +run():void, y al igual que en la forma anterior, aquí deben incluirse las instrucciones que van a ser ejecutadas dentro del hilo. La ejecución (el comportamiento) ocurre igual que en la forma anterior de hacer objetos hilo. Incluso el diseño es similar, y lo único diferente en él es que la clase hilo no extiende de Thread sino que implements Runnable. En cambio la implementación en Java si cambia de la forma como se explicó anteriormente y tal como se presenta a continuación:

```java
package hilos;

import mundo.CuentaMultiplo;
import interfaz.InterfazMultiplo;

public class HiloCuentaMultiplo implements Runnable{

    private CuentaMultiplo cuentaMultiplo;
    private InterfazMultiplo principal;

    public HiloCuentaMultiplo(InterfazMultiplo ppal,
            CuentaMultiplo cm){
        principal = ppal;
        cuentaMultiplo = cm;
    }

    public void run(){
        int cantidadMultiplo =
                cuentaMultiplo.cuantosMultiplos();
        principal.imprimirCantidadMultiplos(cuentaMultiplo,
                cantidadMultiplo);
    }
}
```

```java
package interfaz;
import mundo.CuentaMultiplo;
public class InterfazMultiplo {
    private CuentaMultiplo m5;
    private CuentaMultiplo m7;
    private CuentaMultiplo m2;
    private HiloCuentaMultiplo hm5;
    private HiloCuentaMultiplo hm7;
    private HiloCuentaMultiplo hm2;
    public InterfazMultiplo(){
        m5 = new CuentaMultiplo("Juan", 5);
        m7 = new CuentaMultiplo("Victor", 7);
        m2 = new CuentaMultiplo("Ángela", 2);

        hm5 = new HiloCuentaMultiplo(this, m5);
        hm7 = new HiloCuentaMultiplo(this, m7);
        hm2 = new HiloCuentaMultiplo(this, m2);
    }
    public void contarMultiplos(){
        Thread hiloReal5 = new Thread(hm5);
        Thread hiloReal7 = new Thread(hm7);
        Thread hiloReal2 = new Thread(hm2);

        hiloReal5.start();
        hiloReal7.start();
        hiloReal2.start();
    }
    public void imprimirCantidadMultiplos(CuentaMultiplo m,
            int cantidadM){
        System.out.println(m.darNombre()+" cuenta "+cantidadM
                +" múltiplos de "+m.darNumeroMultiplo());
    }
    public static void main(String[] args){
        InterfazMultiplo interfaz = new InterfazMultiplo();
        interfaz.contarMultiplos();
    }
}
```

Implementación en Java del Cuenta Múltiplos con Hilos (implements Runnable)