Threads

Grupo #7:
Cristhian Motoche
Evelin Zuñiga
Javier Utreras

Contenido

- Introducción
- Marco Teórico
  - Definición, Instanciación y Ejecución
  - Ejemplo
  - Estado de los hilos
  - Ejemplo
  - Sincronización e Interacción
  - Ejemplo
- Conclusiones
- Recomendaciones
- Bibliografía

Introducción

Introducción

Existe una ley informática llamada ley de Wirth

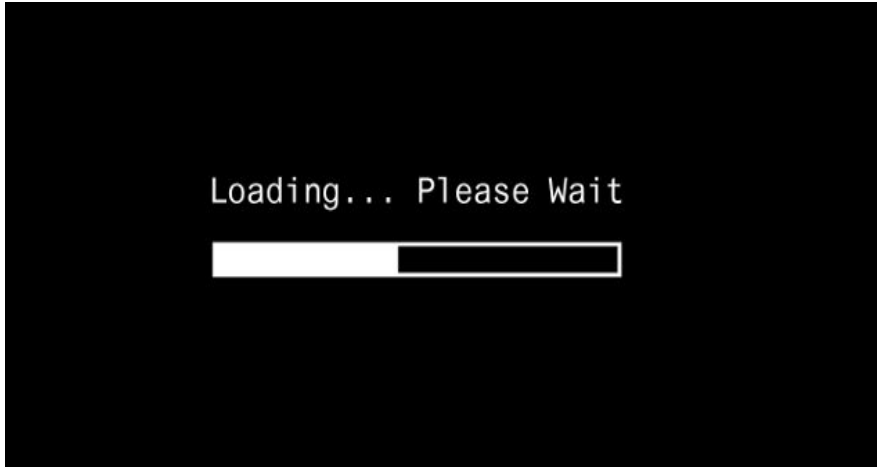
“El software se ralentiza más deprisa de lo que se acelera el hardware”

El hardware se está volviendo, claramente, más rápido a medida que pasa el tiempo y parte de ese desarrollo está cuantificado por la Ley de Moore. Los programas tienden a hacerse más grandes y complicados con el paso del tiempo y a veces los programadores se refieren a la Ley de Moore para justificar la escritura de código lento o no optimizado, pensando que no será un problema porque el hardware sobre el que correrá el programa será cada vez más rápido.

Un ejemplo de la **Ley de Wirth** que se puede observar es que el tiempo que le toma a un PC actual arrancar su sistema operativo no es menor al que le tomaría a un PC de hace cinco o diez años con un sistema operativo de la época.

¿Para que la necesitamos?

Respuesta rápida



¿Para que la necesitamos?

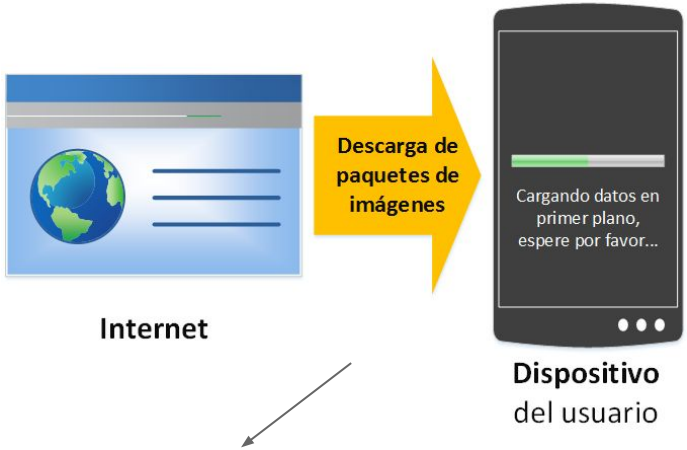
Con la multitarea el usuario nunca quedará bloqueado -pudiendo seguir usando la aplicación mientras algo muy gordo se ejecuta debajo; se eliminan la mayoría de pantallas de carga o no interrumpirán la experiencia de uso de la aplicación; y se ejecutará de manera más óptima, haciendo que el procesador no esté esperando continuamente y con cuellos de botella por llegarle un montón de cosas a la vez. Claro está, si se hace bien.

Que es un Hilo?

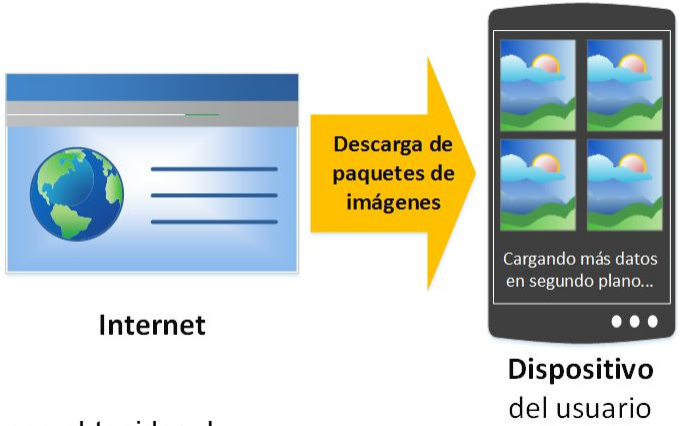
Un Hilo es un trozo de código de un programa que puede ser ejecutado al mismo tiempo que otro

¿Qué puede ejecutarse de manera simultánea a otro?

Vamos a poner el siguiente ejemplo, imagina que queremos ver un listado de 100 imágenes que se descargan desde Internet, como usuario ¿Cuál de las dos opciones siguientes elegirías?:



Ley de Wirth



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

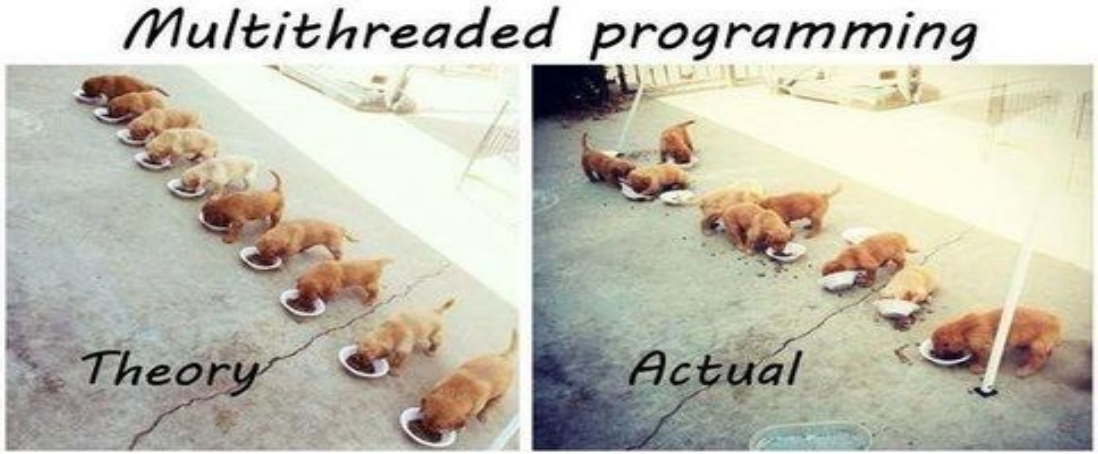
Planos de ejecución

Primer plano:

- Aquí se ejecuta únicamente un hilo llamado “hilo principal”. Aquí se programa siempre, sin conocimiento de que estábamos trabajando ya con hilos.
- Es el hilo que trabaja con las vistas, es decir, con la interfaz gráfica que ve el usuario: botones, ventanas emergentes, campos editables, etc. También, puede ser usado para hacer cálculos u otros procesamientos complejos
- El primer plano influirá en la felicidad del usuario. Aquí es donde el usuario interacciona de manera directa, además todo lo que pase aquí lo ve y lo siente.
- La mala gestión del primer plano por parte del desarrollador, será castigada por el sistema operativo

Segundo plano (o en inglés background):

- Se ejecuta todo el resto de hilos. El segundo plano tiene la característica de darse en el mismo momento que el primer plano. Aquí los hilos deberían de llevar las ejecuciones pesadas de la aplicación.
- El segundo plano el usuario no lo ve, es más, ni le interesa, para el usuario no existe. Por lo que comprenderás, que el desarrollador puede moverse libremente por este segundo plano.
- Aquí el desarrollador se puede resarcir y hacer que, por ejemplo, un método que tarde horas en ejecutarse, ya que el usuario ni lo sentirá.



PROCESO

No confundamos el término proceso con hilo.

Un proceso es el programa o aplicación en ejecución

Así, se deduce y es verdad que un proceso contiene un hilo, como mínimo el hilo principal que corre en primer plano o varios hilos

El principal más algunos en segundo plano.

HILO

Un Hilo, con ejemplo gráfico para hacer mucho más efectivo el ejercicio

EJEMPLO

Un listado de imágenes que se descargan desde Internet.

Hilo que lo hace todo
encargado de descargar los datos de Internet
Y también encargado dibujar las descargas en la pantalla

Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

HILO

Fábrica (Representa todo lo que consuma tiempo de proceso, como consultas a Internet, cálculos, etc)
Fabrica paquetes para la tienda

Internet

Hilo que lo hace todo
Encargado de descargar las imágenes de Internet.
Y también de dibujar las descargas en la pantalla

Tienda (Representa la pantalla del dispositivo)
Aquí los paquetes se muestran a los clientes en el escaparate para que los puedan consumir

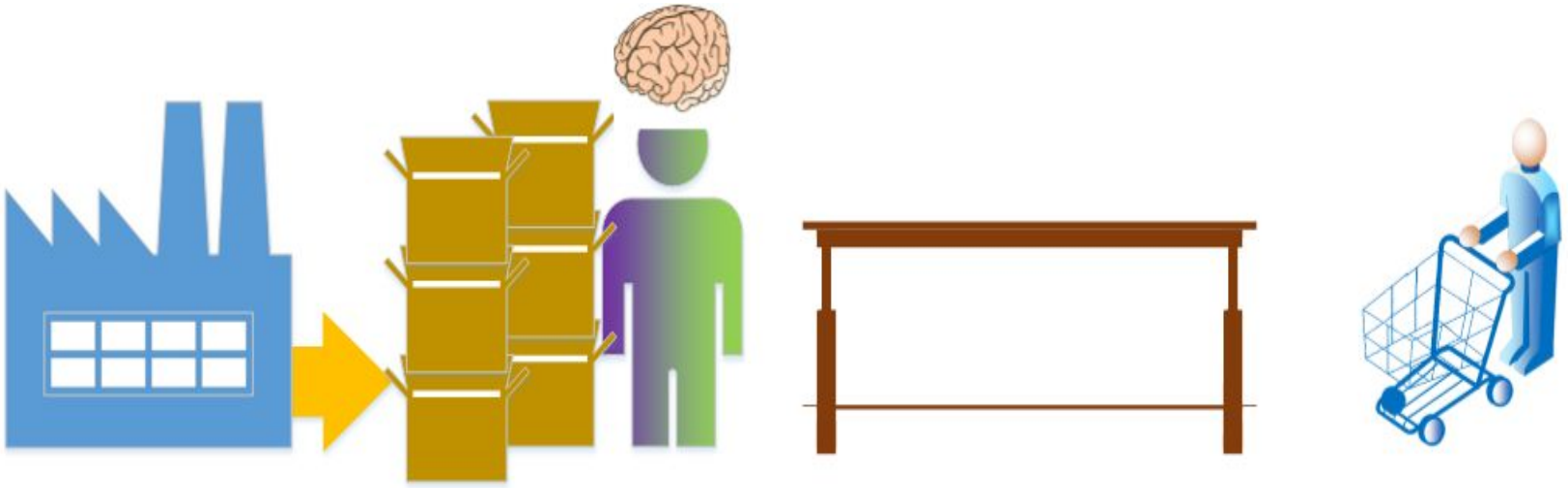
Pantalla del dispositivo

Usuario

Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

HILO

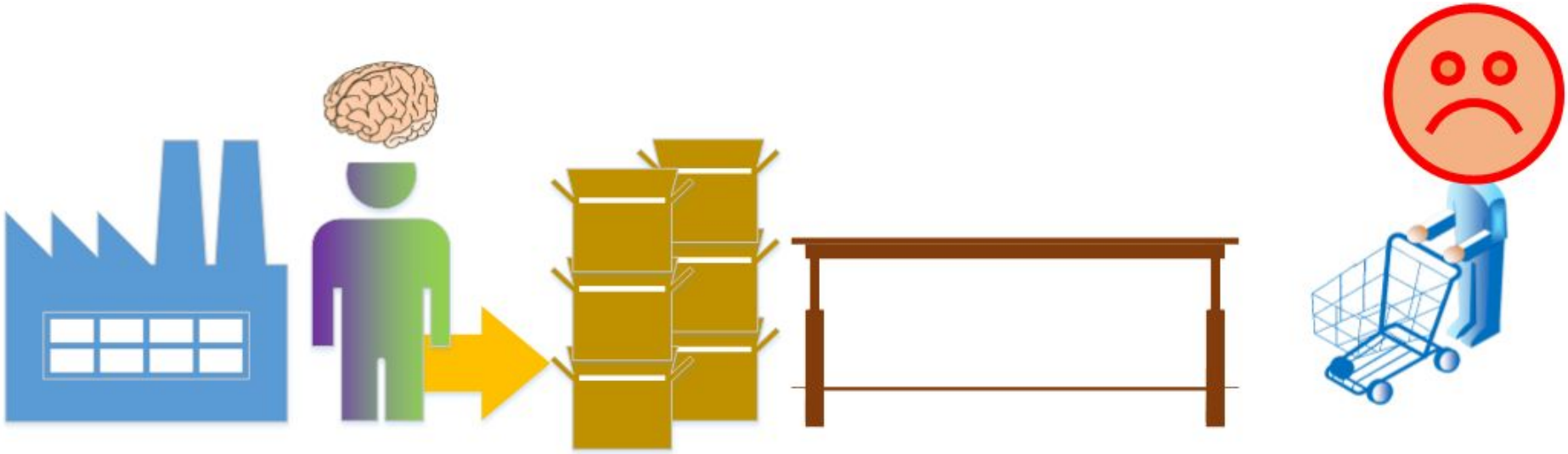
1) Primero busca todos los paquetes de datos con las “imágenes” de una fábrica que se llama “Internet”.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

HILO

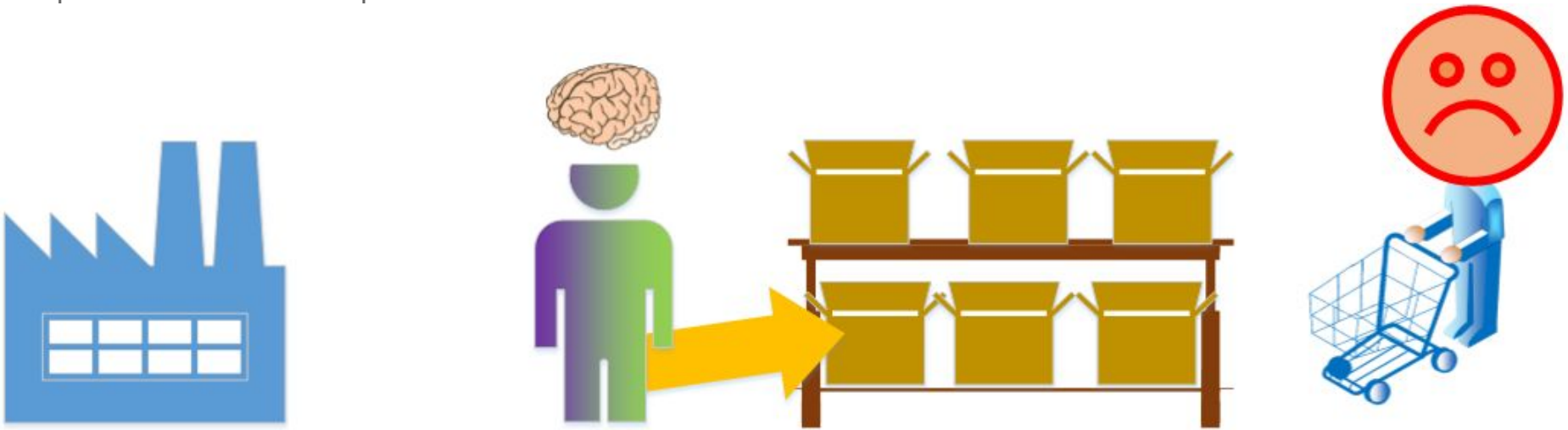
2) Lleva todos estos paquetes en camión y los descarga a una tienda llamada “dispositivo del usuario”.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

HILO

3) Luego le toca colocar todo y cada uno de los paquetes que ha trasportado al escaparate de la tienda, que para aclarar también podemos llamar como “pantalla del dispositivo”.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Hilo

4) Todo esto se hace con la finalidad de que el cliente o “usuario” los consuma. Eso sí, solo en este momento se le permite al usuario poder consumirlos -ya que es el momento que tiene algo que consumir, antes no había nada- con lo que el resto del tiempo estuvo esperando.

Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

MULTI HILO

¿Será necesario contratar a un segundo trabajador/hilo?

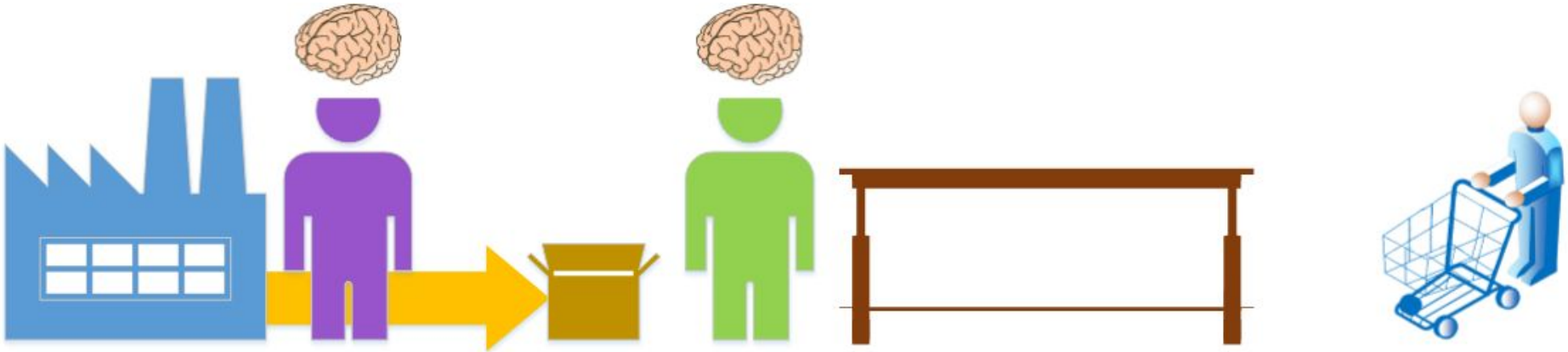
Si, pues es algo que podría hacerse en dos hilos diferentes, uno principal -encargado de dibujar en pantalla- y otro en segundo plano -encargado de descargar los datos desde Internet.

Veremos el mismo ejemplo anterior

Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

MULTITAREA EN TODO SU ESPLENDOR

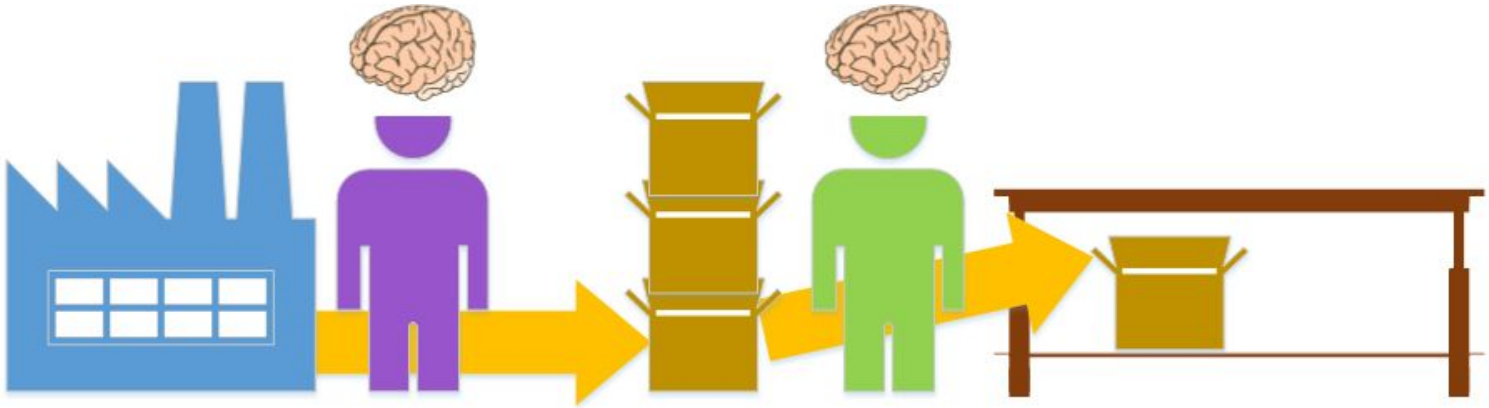
1) Para empezar, el trabajador/hilo en segundo plano toma algún paquete de datos de la fábrica que llamamos “Internet”, lo lleva en camión y lo descarga en la tienda. Donde el trabajador/hilo principal está esperando a que le llegue algún paquete para poder realizar su labor.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

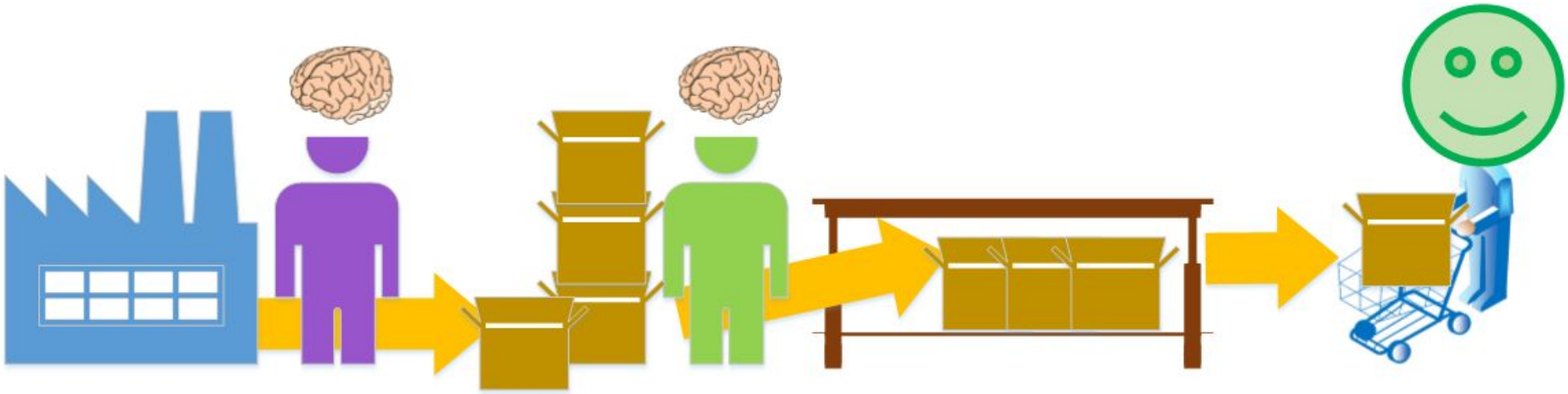
MULTITAREA EN TODO SU ESPLENDOR

2) El trabajador/hilo principal ya tiene algo que hacer. Toma el paquete que le ha llegado y lo coloca en el repisa de la tienda (“pantalla del dispositivo”). Mientras tanto, el trabajador/hilo en segundo plano no ha terminado su trabajo, con lo que continúa llevando paquetes a la tienda desde la fábrica. Cabe notar, que el cliente/usuario ya tiene al menos un paquete que consumir en la tienda.



MULTITAREA EN TODO SU ESPLENDOR

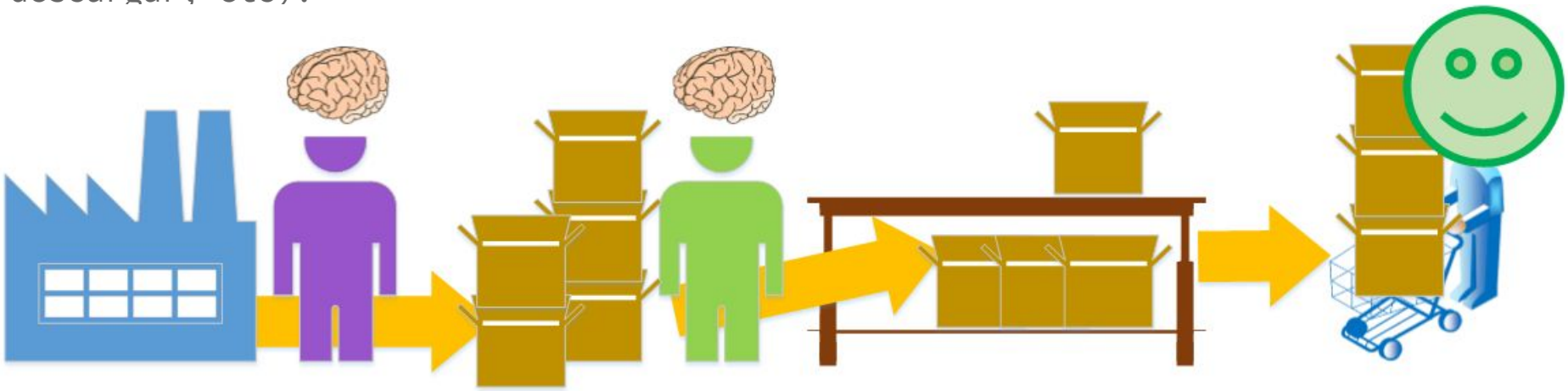
3) El proceso continúa. El trabajador/hilo en segundo plano lleva a la tienda los paquetes, para que el trabajador/hilo en primer plano los coloque en la repisa. Por lo que, el cliente/usuario puede ir consumiendo cuanto desee de lo que exista ya en la repisa.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

MULTITAREA EN TODO SU ESPLENDOR

4) Esto durará hasta que se cumpla alguna condición de finalización o interrupción de algún trabajador/hilo, las cuales pueden ser muchas (por ejemplo: apagar el dispositivo, que se hayan acabado los paquetes que descargar, etc).



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Tiempos de ejecucion

1) Un solo hilo de ejecución

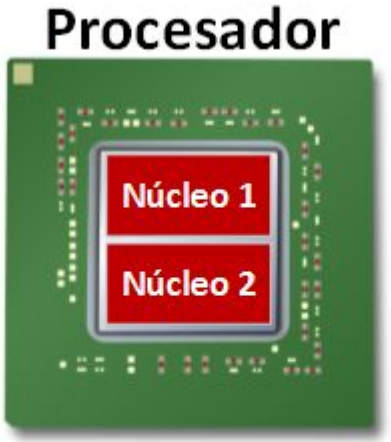
2) Dos hilos de ejecución

Imágenes obtenidas de :

Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Lugar de ejecución

Sabiendo que un procesador es el encargado de ejecutar los programas. Y que un procesador puede tener entre un núcleo y muchos. Cada núcleo ejecuta un hilo a la vez. Por lo que si tenemos un procesador con dos núcleos, podremos ejecutar dos hilos a la vez. **Si tenemos uno de cuatro núcleos podrá ejecutar un máximo de cuatro hilos de manera simultánea. Y así con todos los núcleos de los que dispongamos.**



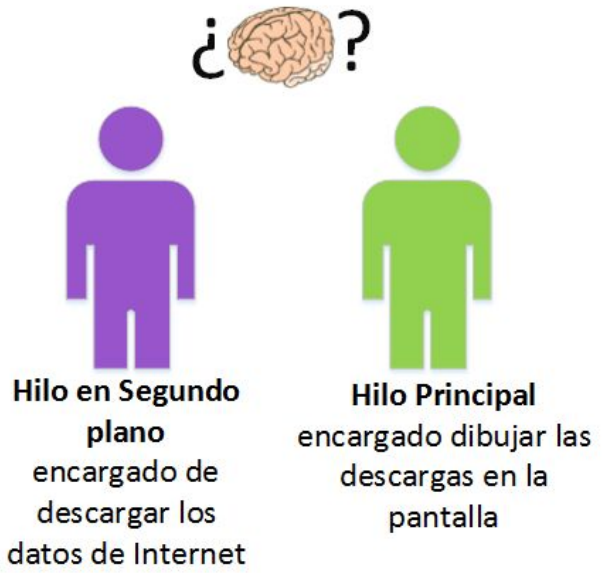
Ejemplo de 1 procesador que contiene 2 núcleos

Imágenes obtenidas de :

Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Si existe solo un núcleo ?

El procesador se encarga de decidir cuánto tiempo va a usar cada hilo el único núcleo que existe. Dicho de otro modo, el procesador partirá los hilos en pedazos muy pequeños y los juntará saltando en una línea de tiempo. Al procesarse los dos hilos de manera saltada, y gracias a la gran velocidad con la que procesan datos los procesadores hoy día, dará la sensación de paralelismo, pero no es paralelismo.



Imágenes obtenidas de :
Moya, R. (2017). *Multitarea e hilos fácil y muchas ventajas*
http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Si existe solo un núcleo ?

Si nos fijamos en la siguiente línea de tiempo, la longitud es tan larga como si lo hiciéramos todo en un hilo.

No es paralelo pero sí es concurrente. La definición de concurrencia: dos o más componentes entre ellos independientes, se ayudan para solucionar un problema común. Lo concurrente se recomienda ejecutarse en paralelo para sacar el máximo partido de la concurrencia.

Si lo ejecutamos todo en el hilo principal, siempre se va a bloquear al usuario. Pero si lo ejecutamos con hilos, aunque solo dispongamos de un núcleo, habrá en algún instante en el que el usuario tenga con que actuar, y el procesador ya se encargará de darle preferencia al usuario para no bloquearle.

En esencia la multitarea nos permite ejecutar varios procesos a la vez; es decir, de forma concurrente y por tanto eso nos permite hacer programas que se ejecuten en menor tiempo y sean más eficientes

A veces necesitamos que nuestro programa Java realice varias cosas simultáneamente.

Para conseguir que Java haga varias cosas a la vez o que el programa no se quede parado mientras realiza una tarea compleja, tenemos los hilos (Threads).

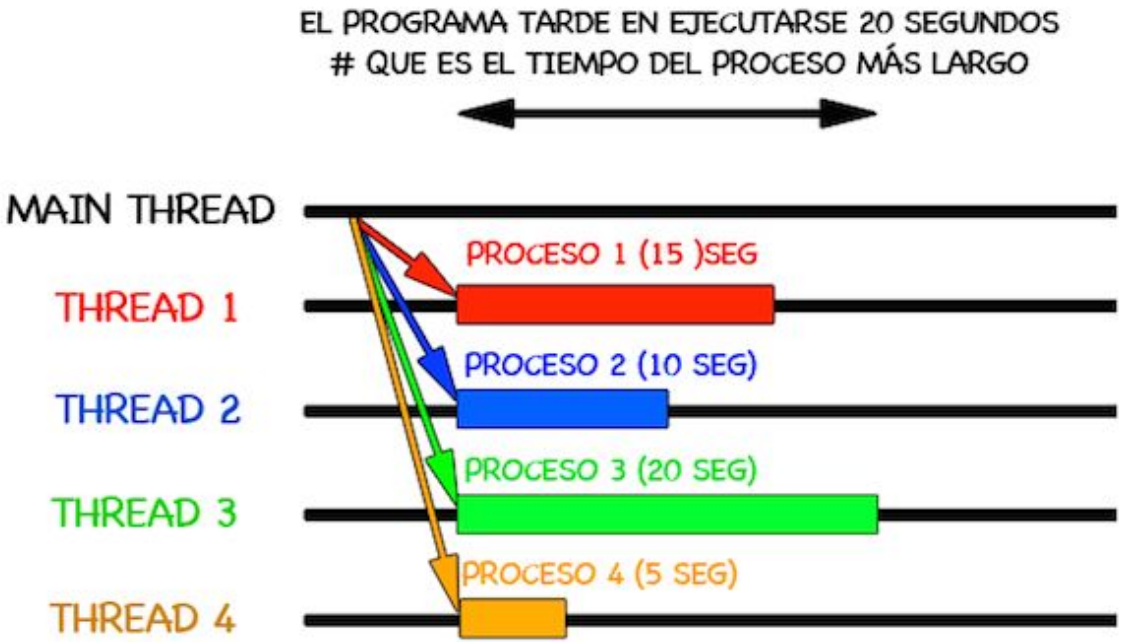
En Java, el proceso que siempre se ejecuta es el llamado **main** que es a partir del cual se inicia prácticamente todo el comportamiento de nuestra aplicación.

En ocasiones es necesario que se ejecuten varios hilos dentro de una misma aplicación al mismo tiempo (multithreading).

Ejecucion de 4 programas

Imágenes obtenidas de :

Moya, R. (2017). *Multitarea e Hilos en Java con ejemplos (Thread & Runnable)*
http://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/



Imágenes obtenidas de :

Moya, R. (2017). *Multitarea e Hilos en Java con ejemplos (Thread & Runnable)*
http://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/

Marco teórico

Definición

Un hilo o subproceso en Java comienza con una instancia de la clase **java.lang.Thread**:

a) Los métodos propios de la clase **Thread** son:
   1) start()
   2) notify()
   3) sleep()
   4) run()
   5) join()

b) Los objetos Thread se puede crear extendiendo de la clase Thread y sobrescribiendo el método: **public void run()**.
c) También los objetos **Thread** pueden ser creados implementando la interface **Runnable**.

Definición

Para definir e instanciar un nuevo Thread (hilo o subproceso) existen 2 formas:

- Extendiendo (o heredando) a la clase **java.lang.Thread**
- Implementando la interfaz **Runnable**

Imágenes obtenidas de :

Moya, R. (2017). *Multitarea e Hilos en Java con ejemplos (Thread & Runnable)* http://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/

Definición

Uso de Subclase

Cuando se crea una subclase de Thread, la subclase debería definir su propio método run() para sobremontar el métoddo run() de la clase Thread. La tarea concurrente es desarrollada en este método run().

Ejecución del método run()
Una instancia de la subclase es creada con new, luego llamamos al método start() de la thread para hacer que la máquina virtual Java ejecute el método run().

DEFINICIÓN

Implementación de la Interface Runnable

La interfaz Runnable requiere que sólo un método sea implementado, el método run(). Primero creamos una instancia de esta clase con new, luego creamos una instancia de Thread con otra sentencia new y usamos nuestra clase en el constructor. Finalmente, llamamos el método start() de la instancia de Thread para iniciar la tarea definida en el método run().

Instanciación

Si has extendido la clase Thread, el instanciar el hilo es realmente simple:

```
MiHilo h = new MiHilo();
```

Si has implementado la interfaz Runnable, es un poquito más complicado, pero solo un poco:

```
MiHilo h = new MiHilo();
Thread t = new Thread(h); // Pasas tu implementación de Runnable al nuevo Thread
```

Métodos o ejecuciones de hilos

Métodos de uso común:

void start():
usado para iniciar el cuerpo de la thread definido por el método run().
void sleep():
pone a dormir una thread por un tiempo mínimo especificado.
void join():
usado para esperar por el término de la thread, por ejemplo por término de método run().
void yield():
Mueve a la thread desde el estado de corriendo al final de la cola de procesos en espera por la CPU.
Notify(); Despierta un hilo

Ejemplo

Ejemplo de Cajera en tiempos

```java
public class Cajera {

	private String nombre;

	// Constructor, getter y setter

	public void procesarCompra(Cliente cliente, long timeStamp) {

		System.out.println("La cajera " + this.nombre +
				" COMIENZA A PROCESAR LA COMPRA DEL CLIENTE " + cliente.getNombre() +
				" EN EL TIEMPO: " + (System.currentTimeMillis() - timeStamp) / 1000     +
				"seg");

		for (int i = 0; i < cliente.getCarroCompra().length; i++) {
				this.esperarXsegundos(cliente.getCarroCompra()[i]);
				System.out.println("Procesado el producto " + (i + 1) +
				" ->Tiempo: " + (System.currentTimeMillis() - timeStamp) / 1000 +
				"seg");
		}

		System.out.println("La cajera " + this.nombre + " HA TERMINADO DE PROCESAR " +
				cliente.getNombre() + " EN EL TIEMPO: " +
				(System.currentTimeMillis() - timeStamp) / 1000 + "seg");
```

```java
	private void esperarXsegundos(int segundos) {
		try {
			Thread.sleep(segundos * 1000);
		} catch (InterruptedException ex) {
			Thread.currentThread().interrupt();
		}
	}
```

Ejemplo de Cajera en tiempos

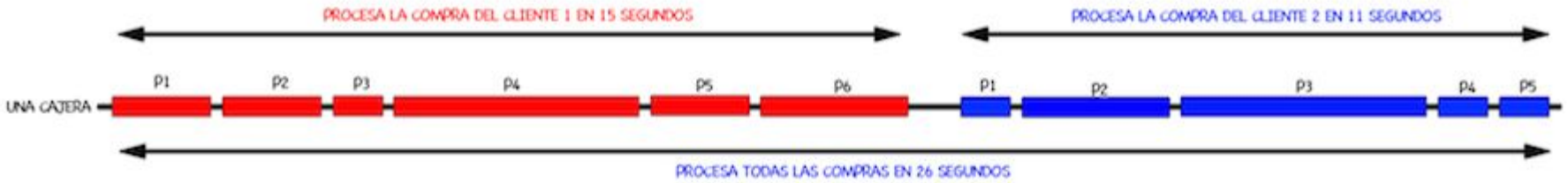
```java
public class Cliente {

        private String nombre;
        private int[] carroCompra;

        // Constructor, getter y setter

}
```

```java
public class Main {

        public static void main(String[] args) {

                Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5, 2, 3 });
                Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1, 1 });

                Cajera cajera1 = new Cajera("Cajera 1");
                Cajera cajera2 = new Cajera("Cajera 2");

                // Tiempo inicial de referencia
                long initialTime = System.currentTimeMillis();

                cajera1.procesarCompra(cliente1, initialTime);
                cajera2.procesarCompra(cliente2, initialTime);
        }
}
```

Ejemplo de Cajera en tiempos

```
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg
Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
```

Ejemplo de Cajera en tiempos



¿Y si en vez de procesar primero un cliente y después otro, procesamos los dos a la vez?, ¿Cuánto tardaría el programa en ejecutarse?

```java
public class CajeraThread extends Thread {

	private String nombre;

	private Cliente cliente;

	private long initialTime;

	// Constructor, getter & setter

	@Override
	public void run() {

		System.out.println("La cajera " + this.nombre + " COMIENZA A PROCESAR LA COMPRA DEL CLIENTE "
				+ this.cliente.getNombre() + " EN EL TIEMPO: "
				+ (System.currentTimeMillis() - this.initialTime) / 1000
				+ "seg");

		for (int i = 0; i < this.cliente.getCarroCompra().length; i++) {
			this.esperarXsegundos(cliente.getCarroCompra()[i]);
			System.out.println("Procesado el producto " + (i + 1)
			+ " del cliente " + this.cliente.getNombre() + "->Tiempo: "
			+ (System.currentTimeMillis() - this.initialTime) / 1000
			+ "seg");
		}
```

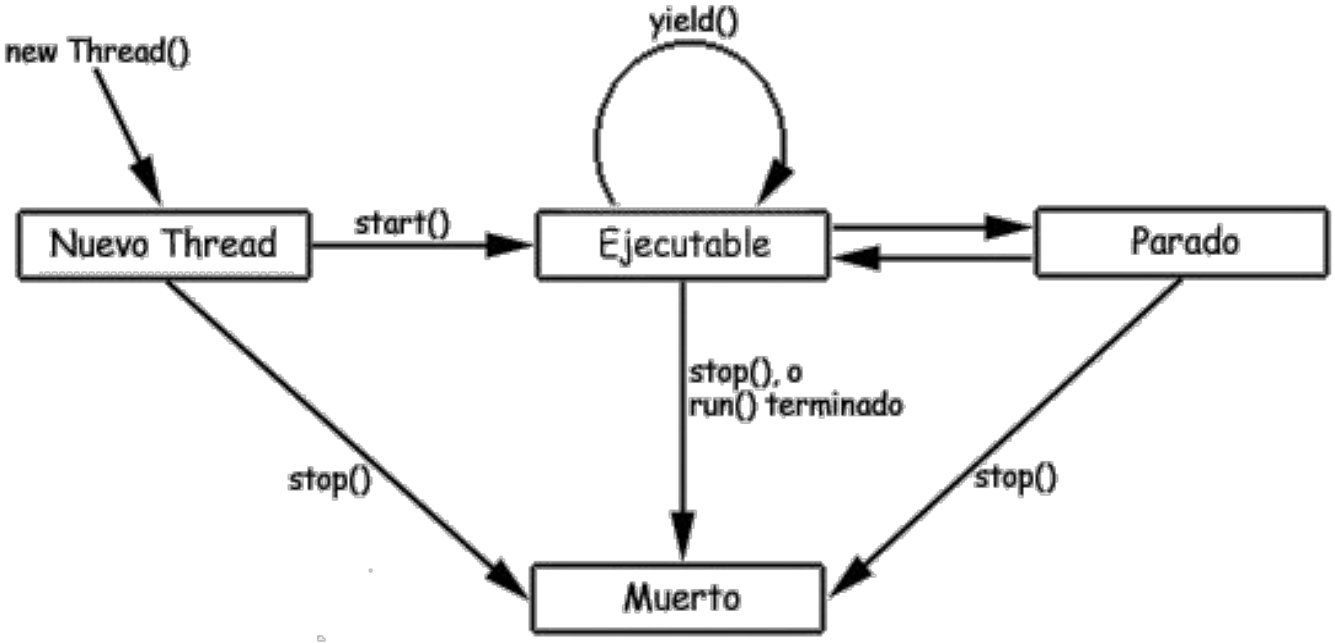
```
                System.out.println("La cajera " + this.nombre + " HA TERMINADO DE PROCESAR "
                                                + this.cliente.getNombre() + " EN EL TIEMPO: "
                                                + (System.currentTimeMillis() - this.initialTime) / 1000
                                                + "seg");
        }

        private void esperarXsegundos(int segundos) {
                try {
                        Thread.sleep(segundos * 1000);
                } catch (InterruptedException ex) {
                        Thread.currentThread().interrupt();
                }
        }

}
```

```java
public class MainThread {

	public static void main(String[] args) {

		Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5, 2, 3 });
		Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1, 1 });

		// Tiempo inicial de referencia
		long initialTime = System.currentTimeMillis();
		CajeraThread cajera1 = new CajeraThread("Cajera 1", cliente1, initialTime);
		CajeraThread cajera2 = new CajeraThread("Cajera 2", cliente2, initialTime);

		cajera1.start();
		cajera2.start();
	}
}
```

```
La cajera Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL TIEMPO: 0seg
La cajera Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
La cajera Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
La cajera Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
```

Tiempos con hilos en ejemplo cajera

Estado de los hilos



Nuevo Thread

```
Thread MiThread = new MiClaseThread();
Thread MiThread = new Thread( new UnaClaseThread,"hiloA" );
```

- Crea un nuevo hilo de ejecución pero no lo arranca, lo deja en el estado de Nuevo Thread.
- Cuando un hilo está en este estado, es simplemente un objeto Thread vacío.
- El sistema no ha destinado ningún recurso para él.
- Desde este estado solamente puede arrancarse llamando al método start(), o detenerse definitivamente, llamando al método stop().

EJECUTABLE

```
Thread MiThread = new MiClaseThread();
MiThread.start();
```

La llamada al método start() creará los recursos del sistema necesarios para que el hilo puede ejecutarse, lo incorpora a la lista de procesos disponibles para ejecución del sistema y llama al método run() del hilo de ejecución. En este momento se encuentra en el estado Ejecutable del diagrama. Y este estado es Ejecutable y no En Ejecución, porque cuando el hilo está aquí no esta corriendo Cuando el hilo se encuentra en este estado, todas las instrucciones de código que se encuentren dentro del bloque declarado para el método run(), se ejecutarán secuencialmente.

PARADO

```
Thread MiThread = new MiClaseThread();
MiThread.start();
try {
     MiThread.sleep( 10000 );
} catch( InterruptedException e ) {
     ;
     }
```

El hilo de ejecución entra en estado Parado cuando alguien llama al método suspend(), cuando se llama al método sleep(), cuando el hilo está bloqueado en un proceso de entrada/salida o cuando el hilo utiliza su método wait() para esperar a que se cumpla una determinada condición.

MUERTO

Un hilo que contenga a este método run(), morirá naturalmente después de que se complete el bucle y run() concluya.

```
public void run() {
     int i=0;
     while( i < 20 ) {
     i++;
        System.out.println( "i = "+i );
     }
     }
```

También se puede matar en cualquier momento un hilo, invocando a su método stop(). En el trozo de código siguiente:

```
Thread MiThread = new MiClaseThread();
MiThread.start();
try {
     MiThread.sleep( 10000 );
} catch( InterruptedException e ) {
     ;
     }
MiThread.stop();
```

Ejemplo

Sincronización

¿Cómo funciona la sincronización?

Con bloqueos o cerraduras (locks). Cada objeto en Java tiene un bloqueo propio que solo funciona cuando el objeto tiene código **sincronizado**.

```
public synchronized void doStuff() {
    System.out.println("synchronized");
}
```

is equivalent to this:

```
public void doStuff() {
    synchronized(this) {
        System.out.println("synchronized");
    }
}
```

Sincronización

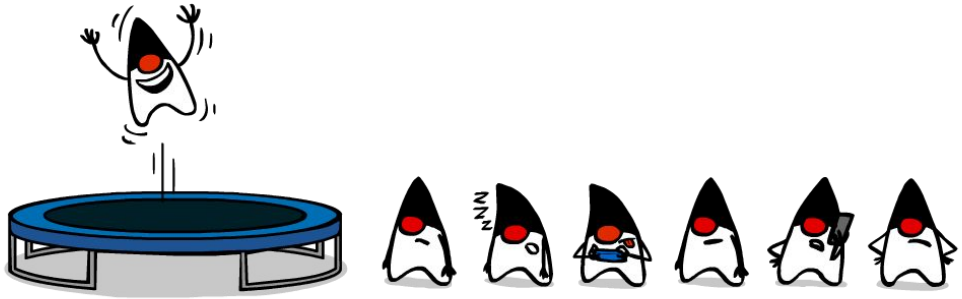
Los siguientes puntos son importantes con respecto al bloqueo y sincronización:

- Solo método (o bloques) pueden ser sincronizados, no variables ni clases.
- Cada clase solo tiene un bloqueo.
- Una clase puede tener método sincronizados y no sincronizados. Pero solo los métodos sincronizados podrán tener bloqueo, los otros no.
- Si dos hilos están a punto de ejecutar un código sincronizado de una misma instancia de una clase, solo un hilo a la vez podrá ejecutar el método. El otro hilo deberá esperar a que el primero termine para realizar su operación.
- Si un hilo pasa al estado de dormido, mantiene cualquier bloqueo que posea y no lo libera.

Sincronización

¿Qué ocurre cuando un objeto no puede obtener el bloqueo?

Cuando un hilo intenta acceder a un método o bloque que ya ha otorgado su bloqueo a algún otro, se dice que este hilo está bloqueado en el bloqueo del objeto.



Sincronización

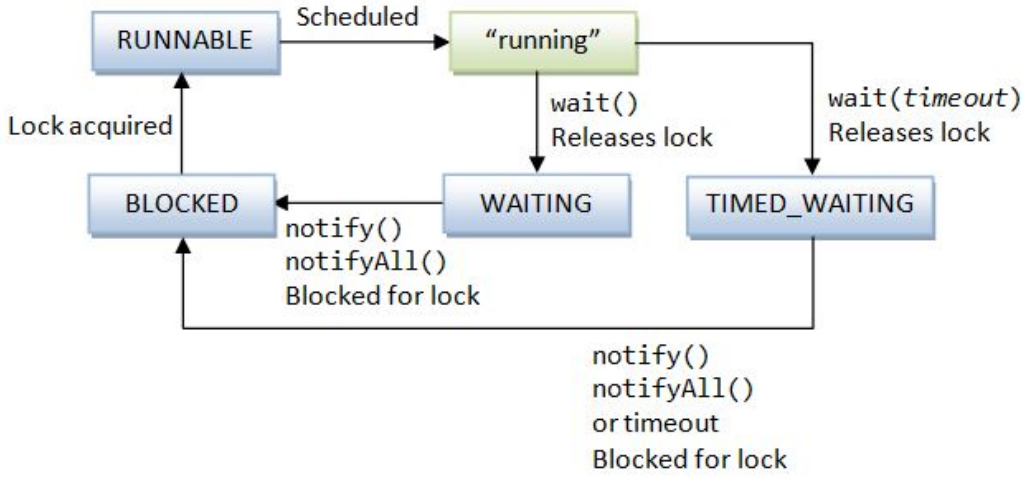
Se debe tener en cuenta los siguientes puntos:

- Los hilos que llamen a un método **no estático sincronizado** de una misma clase se bloquearán entre ellos si son invocados utilizando la misma instancia.
- Los hilos que llamen **a método estáticos sincronizados** en la misma clase siempre se bloquearán entre ellos, ya que están bloqueados en la misma instancia de clase.
- Un método estático sincronizado y un método no estático sincronizado **nunca se bloquearán**.
- Para bloques sincronizados, se tiene que ver a qué objeto exactamente se está utilizando para el bloqueo (aquello que se encuentra dentro de los paréntesis del bloque sincronizado).

Ejemplo

Interacción

El siguiente tema es la interacción entre hilos para poder comunicarse entre ellos sobre su estado y otras cosas más. La clase *Object* tiene tres métodos, *wait(), notify()* y *notifyAll()*, que ayudarán a comunicar el estado de un evento que es de interés para los otros hilos.



Interacción

Los métodos *wait* y *notify* ponen a un hilo en modo de espera hasta que **otro** hilo le notifique que existe una razón para salir de la espera. Otra cosa a tener en cuenta es que estos tres métodos solo pueden ser llamados desde un método sincronizado.

Interacción

```
1.  class ThreadA {
2.     public static void main(String [] args) {
3.        ThreadB b = new ThreadB();
4.        b.start();
5.
6.        synchronized(b) {
7.           try {
8.             System.out.println("Waiting for b to complete...");
9.             b.wait();
10.          } catch (InterruptedException e) {}
11.          System.out.println("Total is: " + b.total);
12.       }
13.    }
14. }
15.
16. class ThreadB extends Thread {
17.    int total;
18.
19.    public void run() {
20.       synchronized(this) {
21.          for(int i=0;i<100;i++) {
22.             total += i;
23.          }
24.          notify();
25.       }
26.    }
27. }
```

Interacción

En ocasiones, es preferible notificar a todos los hilos que están a la espera de un objeto para pasar del modo espera a ejecución. El método *notifyAll()* permite que esto ocurra. Todos los hilos serán notificados y comenzarán a competir para conseguir el bloqueo.

Ejemplo

Conclusiones

Conclusiones

Para evitar problemas de condiciones de carrera se debe sincronizar el código. En Java se utiliza la palabra reservada *syncronized* solo a los métodos de una clase (estáticos y no estáticos) para que estas puedan bloquear el acceso para un solo hilo a la vez.

Para coordinar actividades entre diferentes hilos se utiliza los métodos *wait(), notify()* y *notifyAll()*.

El concepto de multitarea o multiprocesamiento es bastante sencillo de entender ya que solo consiste en hacer varias cosas a la vez sin que se vea alterado el resultado final. Como ya se ha dicho en la entrada no todo se puede paralelizar y en muchas ocasiones suele ser complicado encontrar la manera de paralelizar procesos dentro de una aplicación sin que esta afecte al resultado de la misma, por tanto aunque el concepto sea fácil de entender el aplicarlo a un caso práctico puede ser complicado para que el resultado de la aplicación no se vea afectado.

Puedes llegar a ser mejor programador, tienes que ser realmente bueno en lo que se refiere a la estructura de los datos, algoritmos, diseño usando OOPS, multi-hilo y varios otros conceptos de programación, por ejemplo, recursión, divide y vencerás, prototipado y pruebas de unidad.

Bibliografía

Sierra K. & Bates B. (2015) OCA/OC Java SE 7 Programmer I & II Study Guide. McGraw Hill Education (Publisher).

Abellan, J. (2017). Multitaréa e Hilos, fácil y muchas ventajas. *Jarroba*. Retrieved 9 January 2017, from http://jarroba.com/multitarea-e-hilos-facil-y-muchas-ventajas/

Abellan, J. (2017). *Multitarea e Hilos en Java con ejemplos (Thread & Runnable) - Jarroba*. *Jarroba*. Retrieved 28 December 2016, from http://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/

Navarro, L. (2017). *Hilos en Java(Threads) parte 1*. *Monillo007.blogspot.com*. Retrieved 3 January 2017, from http://monillo007.blogspot.com/2008/01/hilos-en-java-threads-parte-1.html

Moya, R. (2017). *Multitarea e Hilos en Java con ejemplos (Thread & Runnable) - Jarroba*. *Jarroba*. Retrieved 9 January 2017, from http://jarroba.com/multitarea-e-hilos-en-java-con-ejemplos-thread-runnable/