

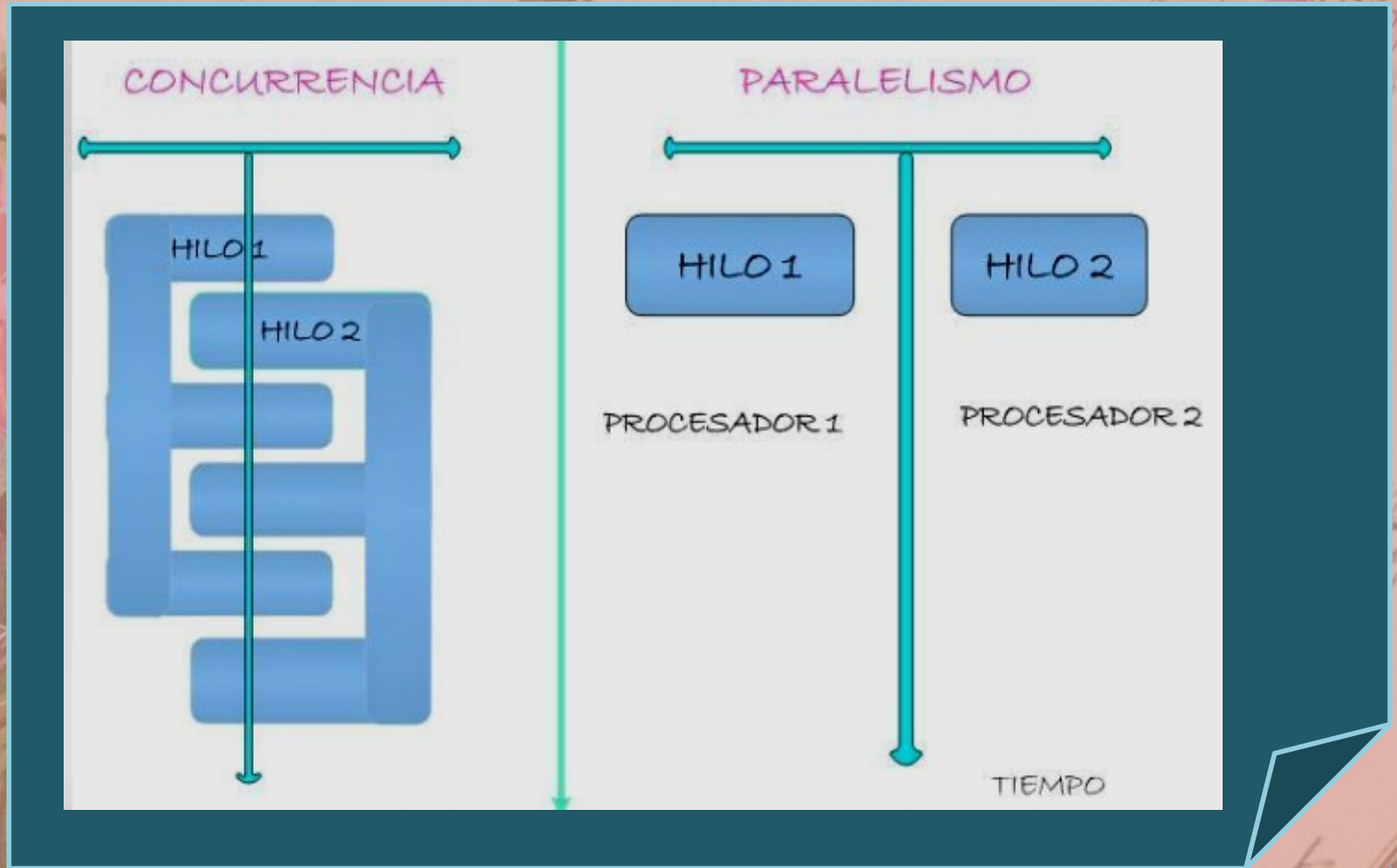
The background is a vintage postcard with a pinkish-red tint. It features a large pink rose on the left, a small Eiffel Tower stamp on the right, and various cursive text elements like 'Cottage Rose Graphics', 'CARTE POSTALE', 'Paris', and 'est. 1897'. A decorative scroll-like border frames the central text area.

Hilos y Multihilos

Subprocesamiento Múltiple

Programación Concurrente

- ❖ Un hilo es un proceso ejecutado en un momento determinado en el sistema operativo, que se realiza directamente en el procesador.
- ❖ Demonios: procesos que define el sistema en sí para poder funcionar.
- ❖ Hilos definidos por el usuario o programador, se define un comportamiento e inicia en un momento específico.

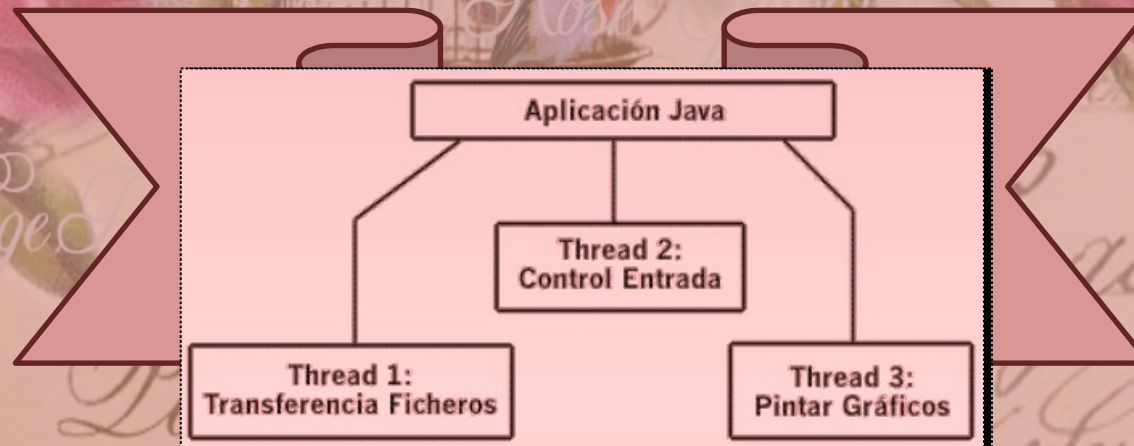



❖ Cada hilo controla un único aspecto dentro de un programa como: supervisar la entrada en un determinado periférico o controlar toda la entrada/salida del disco.

❖ Todos los hilos comparten los mismos recursos.

❖ Los hilos en los procesos: cada uno tiene su copia de código y datos (separados unos de otros.)

❖ *GRAFICAMENTE:*

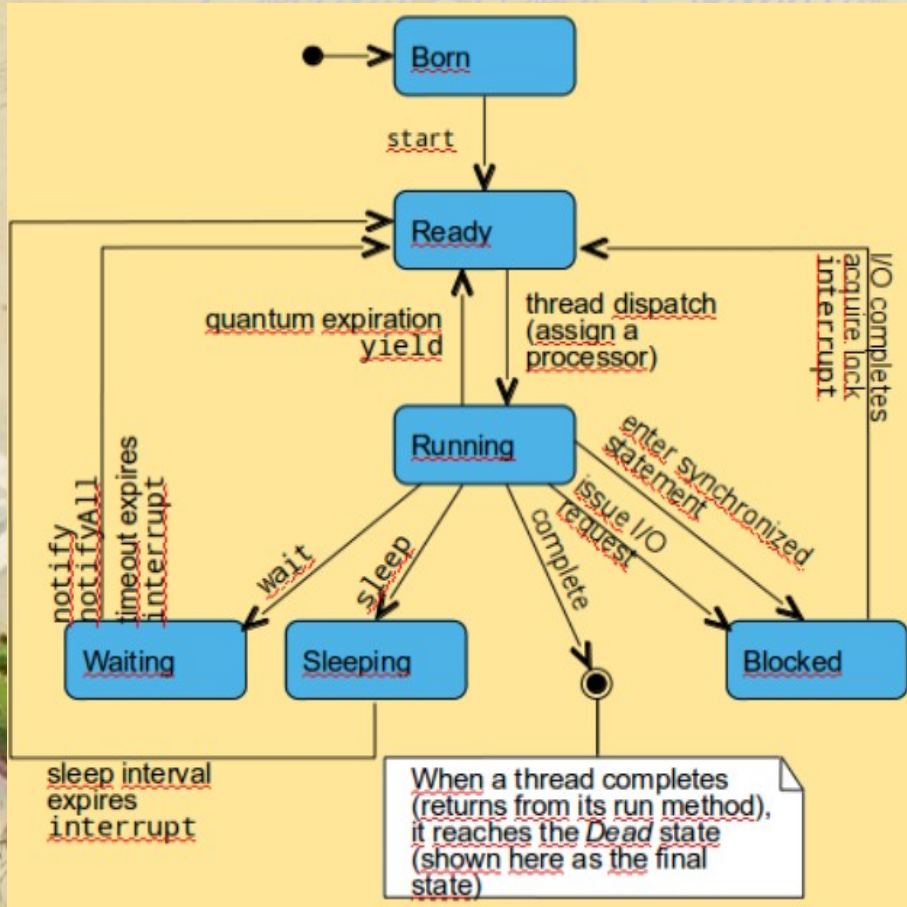


- 
- The background is a vintage postcard with a Parisian theme. It features a pink rose, a small bird perched on a branch, and a pink Eiffel Tower stamp. The text 'COTTAGE ROSE GRAPHICS' is written in a cursive font at the top. The word 'PARIS' is visible on the stamp, and 'est. 1897' is at the bottom. The overall color palette is soft and romantic, with pinks, greens, and browns.
- ❖ Multihilo (Multithread) != Multiproceso.
 - ❖ Multihilo: 2 o más tareas se ejecutan “aparentemente” a la vez, dentro de un mismo programa.
 - ❖ El multiproceso: 2 programas se ejecutan “aparentemente a la vez”, y están bajo el control del OS, no necesitan tener relación unos con otros, solo que el usuario desee que se ejecuten a la vez.
 - ❖ Normalmente las plataformas tienen una sola CPU, por lo cual los procesos se ejecutan “concurrentemente”, sino que comparten la CPU.
 - ❖ En Plataformas con varias CPU, si es posible ejecuciones simultáneas.

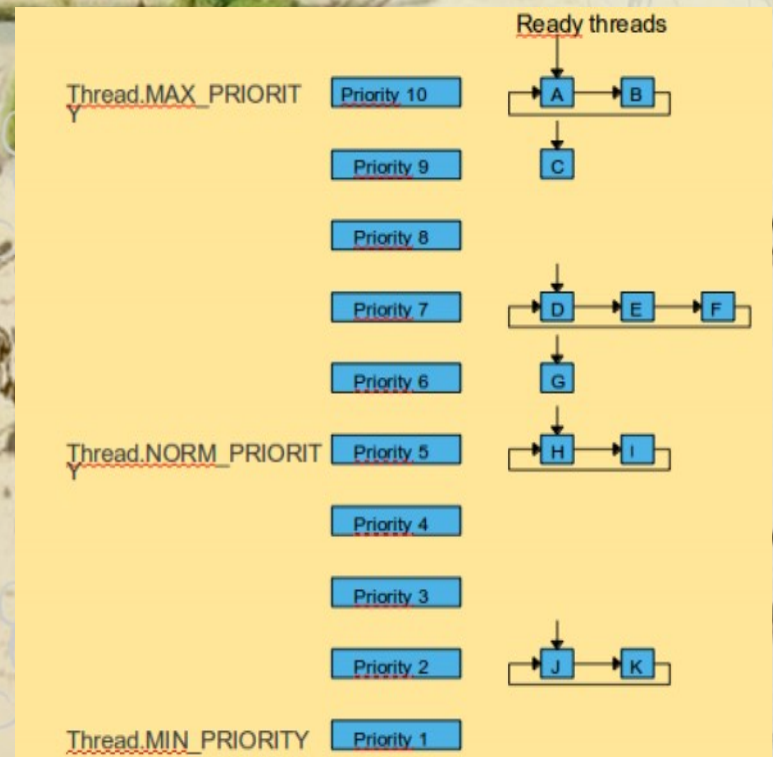
❖ Programas de flujo múltiple

- Un programa multihilo permite que cada thread comience y termine tan pronto como sea posible, este comportamiento permite una mejor respuesta a la entrada en tiempo real.
- Un programa con flujo único puede realizar su tarea ejecutando subtareas secuencialmente.

- Ciclo de vida de los hilos



- Prioridad de ejecución



❖ La clase Thread:

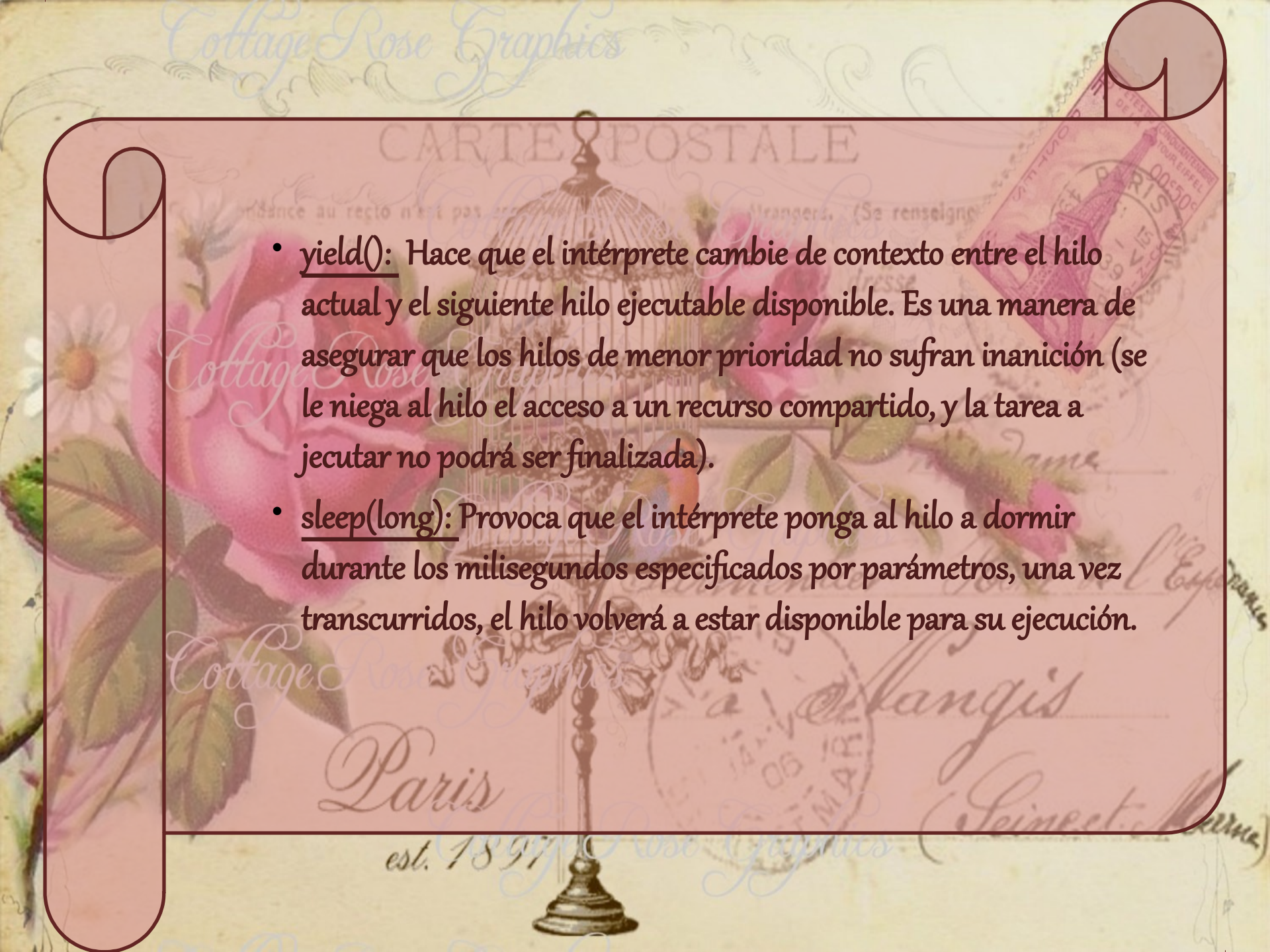
Encapsula todo el control necesario sobre los hilos de ejecución (threads). La clase Thread es la única forma de controlar el comportamiento de los hilos y para ello hace uso de ciertos métodos.

❖ Se debe distinguir entre un objeto thread de un hilo de ejecución

- Para simplificar se puede considerar al objeto thread como el panel de control de un hilo de ejecución (thread).

❖ Métodos de clase: Métodos estáticos que se deben llamar de manera directa en la clase Thread.

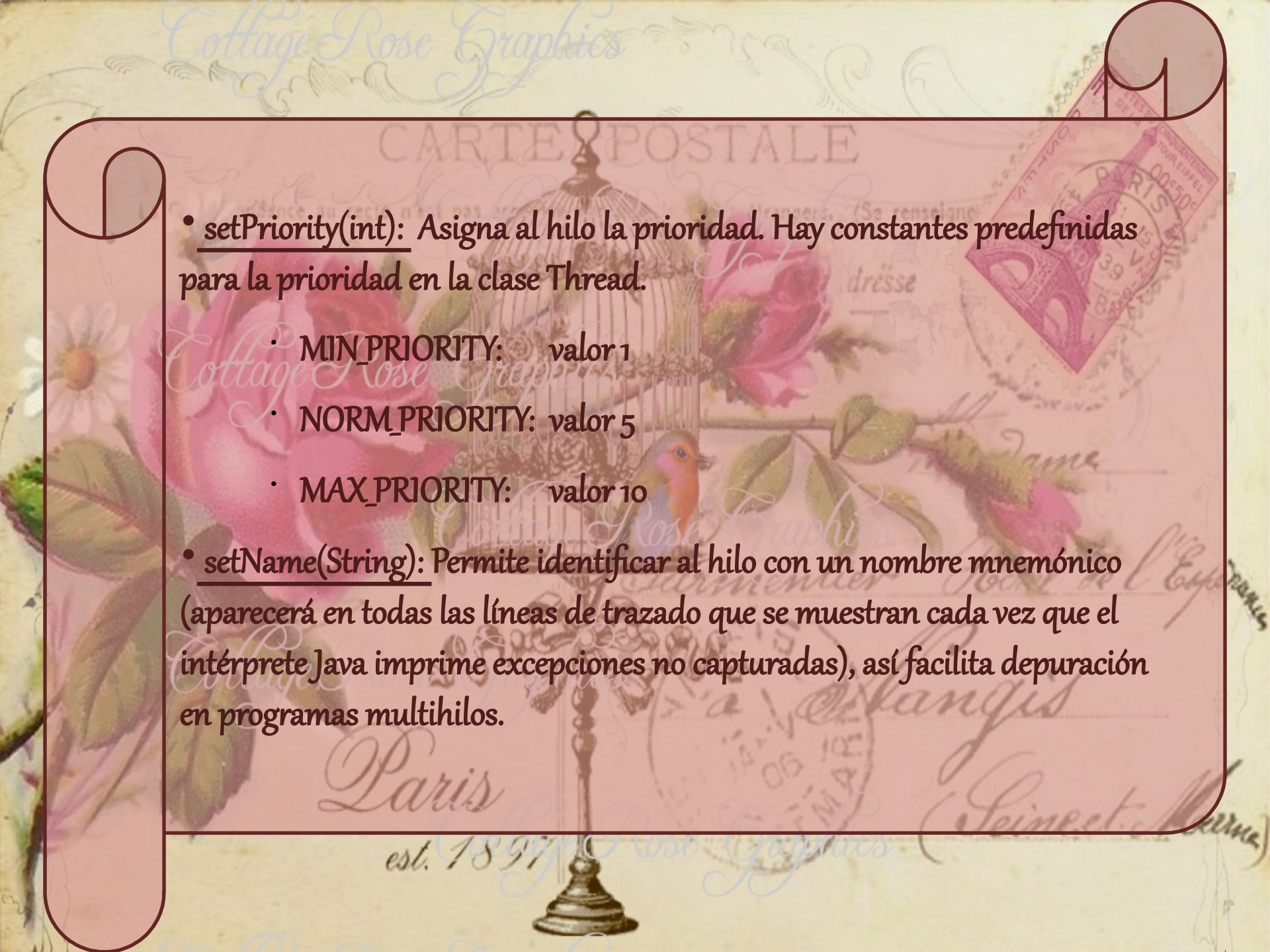
- currentThread(): Devuelve el objeto thread que representa al hilo de ejecución que se ejecuta actualmente.

- 
- yield(): Hace que el intérprete cambie de contexto entre el hilo actual y el siguiente hilo ejecutable disponible. Es una manera de asegurar que los hilos de menor prioridad no sufran inanición (se le niega al hilo el acceso a un recurso compartido, y la tarea a ejecutar no podrá ser finalizada).
 - sleep(long): Provoca que el intérprete ponga al hilo a dormir durante los milisegundos especificados por parámetros, una vez transcurridos, el hilo volverá a estar disponible para su ejecución.

- **Métodos de instancia:**

No están todos los métodos de la clase Thread, sino los más interesantes, los demás corresponden a áreas donde el estándar de Java no está completo, y es posible su obsolescencia en la próxima vs del JDK.

- start(): Indica al intérprete que cree un contexto del hilo del sistema y comience a ejecutarlo, el métodos run() de este hilo será invocado en el nuevo contexto de hilo, No se debe llamar al método start() más de una vez.
- run(): Constituye el cuerpo de un hilo en ejecución, él único método de la interfaz Runnable, es llamado por el método start() luego de que el hilo apropiado del sistema se haya inicializado. Siempre que el método run() devuelve el control, el hilo se detendrá.

- 
- setPriority(int): Asigna al hilo la prioridad. Hay constantes predefinidas para la prioridad en la clase Thread.
 - MIN_PRIORITY: valor 1
 - NORM_PRIORITY: valor 5
 - MAX_PRIORITY: valor 10
 - setName(String): Permite identificar al hilo con un nombre mnemónico (aparecerá en todas las líneas de trazado que se muestran cada vez que el intérprete Java imprime excepciones no capturadas), así facilita depuración en programas multihilos.

- Creación de un hilo:

→ Se debe extender de la clase Thread, así se heredan métodos y variables.

```
public class MiThread extends Thread {  
    public void run() {  
        .... } }
```

→ Segunda forma menos limitada, implementar Runnable

```
public class MiThread implements Runnable {  
    Thread t;  
    public void run() { // Ejecución del thread una vez creado  
        .... } }
```


- Inicio de un Thread:

→ `t1 = new TestTh("Thread 1", (int)(Math.random()*2000));`

Crea un nuevo hilo de ejecución. Los argumentos son el nombre del hilo y el tiempo que se desea esperar antes de imprimir el mensaje.

- `T1.start()`; es un método oculto en el hilo de ejecución que llama a `run()`.

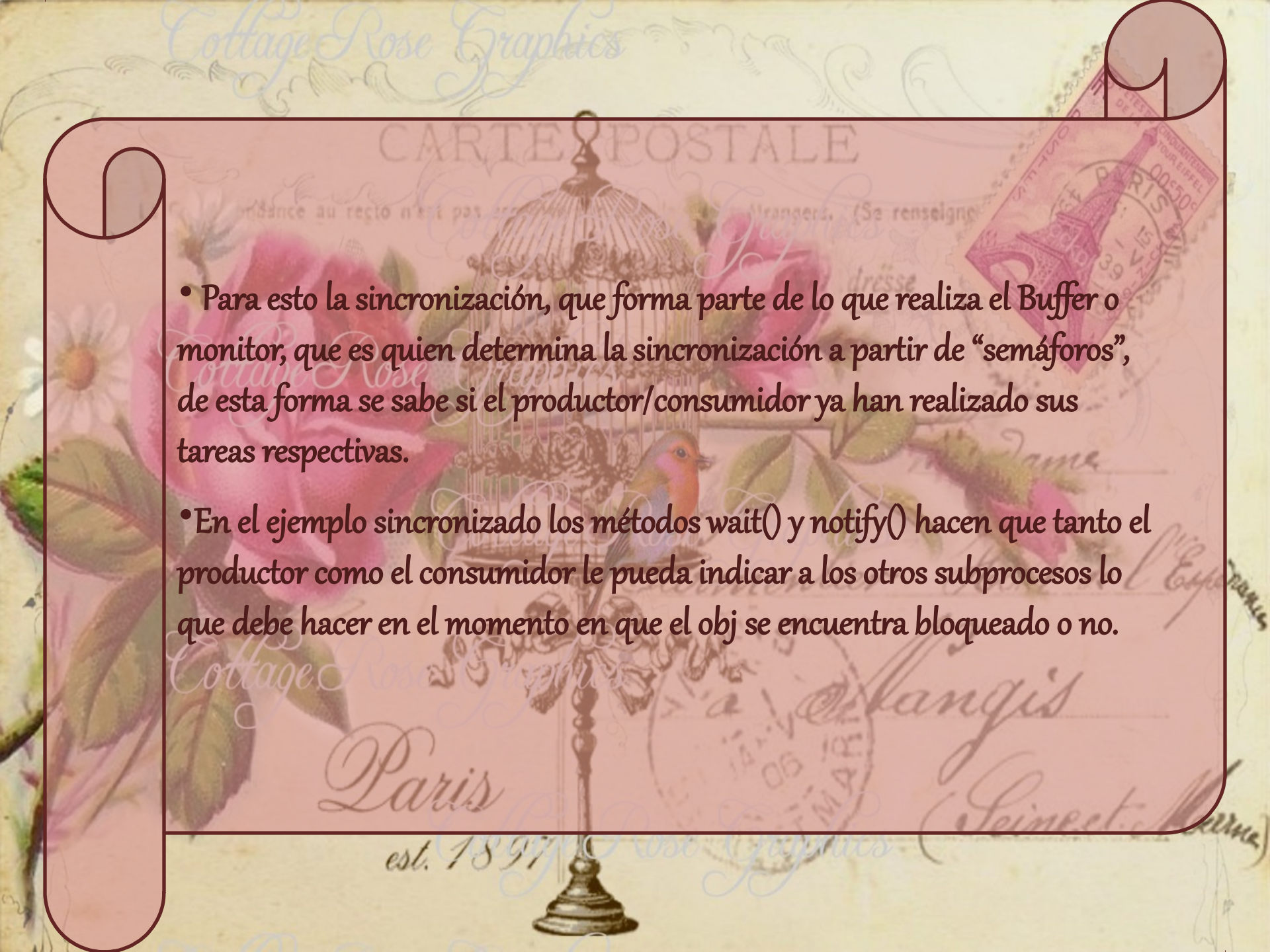
- Sincronización de subprocesos: Otra ventaja del uso de multihilos en una aplicación es la capacidad de la comunicación entre si. Se pueden diseñar hilos para utilizar objetos comunes, que cada hilo puede manipular de manera independiente, de los otros hilos de ejecución. Para esto se crean métodos atómicos.

• Sincronización de subprocesos:

• Ejemplo: productor / consumidor. Un hilo produce una salida, que otro hilo consume.

- Se crea un productor (hilo que irá sacando información por su salida).
- Un consumidor que irá recogiendo la info que saque el productor
- Un monitor o canal en memoria (buffer) que controlará el proceso de sincronización entre los hilos de ejecución.
- Puede suceder que 2 subprocesos necesiten trabajar con un mismo obj al mismo tiempo.
- Si ambos afectan concurrentemente puede darse el caso de que el obj entre en un estado inconsistente.
- Por eso se permite que sea un hilo a la vez que vaya afectando al obj.

- Exclusión mutua o sincronización de procesos: Hay un orden para que los hilos vayan ordenadamente modificando un objeto.
- Con el esquema productor / consumidor se puede controlar el hecho de obtener bloqueos por parte de un subproceso. A pesar de que esta relación puede provocar algunas situaciones de cuidado, los ejemplos con sincronización , permiten evitar inconsistencias de los objs tratados por los subprocesos.
- En el ejemplo no sincronizado, cada proceso se ejecutará sin respetar ninguna prioridad.
- El productor debe dar (no dará hasta que el consumidor esté listo para recibir y no tenga nada pendiente por recibir), y el consumidor recibe (siempre esperará a recibir).

- 
- The background is a collage with a Parisian theme. It includes a vintage postcard with the text 'CARTE POSTALE' and 'PARIS', a pink rose, a small bird perched on a decorative stand, and various cursive script elements like 'Cottage Rose Graphics' and 'Paris'. There are also some circular stamps and a small Eiffel Tower illustration.
- Para esto la sincronización, que forma parte de lo que realiza el Buffer o monitor, que es quien determina la sincronización a partir de “semáforos”, de esta forma se sabe si el productor/consumidor ya han realizado sus tareas respectivas.
 - En el ejemplo sincronizado los métodos `wait()` y `notify()` hacen que tanto el productor como el consumidor le pueda indicar a los otros subprocesos lo que debe hacer en el momento en que el obj se encuentra bloqueado o no.