20151021

dit
UPM

Programación concurrente — Hebras

Juan Antonio de la Puente <jpuente@dit.upm.es>

ETSIT UPM

Referencias

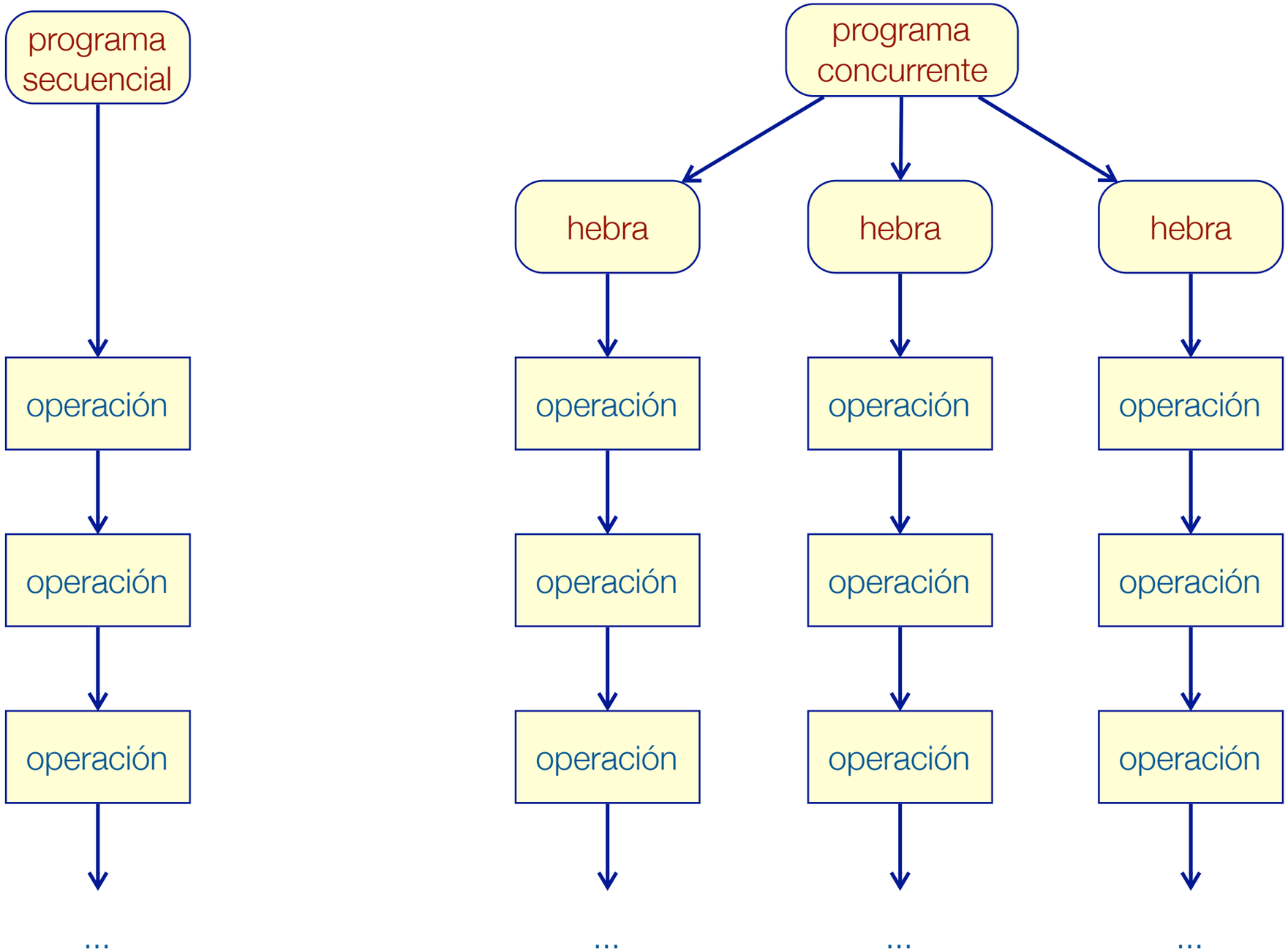
- Scott Oaks & Henry Wong
  Java Threads
  O'Reilly Media; 3rd ed (2004)
- Kathy Sierra & Bert Bates
  Head First Java, ch. 15
  O'Reilly Media; 2nd ed (2005)
- Mordechai Ben-Ari
  Principles of Concurrent and Distributed Programming
  Addison-Wesley; 2nd ed (2006)

Programas concurrentes

Programas secuenciales y concurrentes

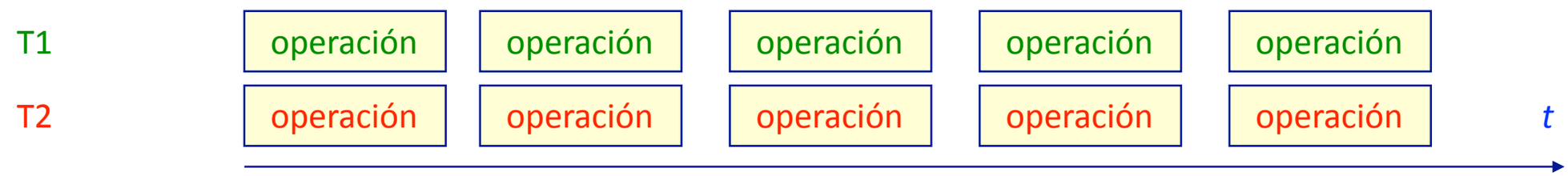
- Los programa que hemos visto hasta ahora son programas secuenciales
  - ejecutan una operación detrás de otra (en secuencia)
    - aunque el orden puede variar (alternativas, bucles, etc).
  - sólo hacen una cosa a la vez
    - sólo hay un flujo de ejecución
- A veces necesitamos que un programa haga varias cosas al mismo tiempo
  - varias tareas o actividades que progresan en paralelo
    - en Java se llaman *threads* o *hebras*
- Estos programas se llaman programas concurrentes
  - tienen varios flujos de ejecución
    - cada uno de ellos ejecuta una secuencia de operaciones

Ejecución concurrente y hebras

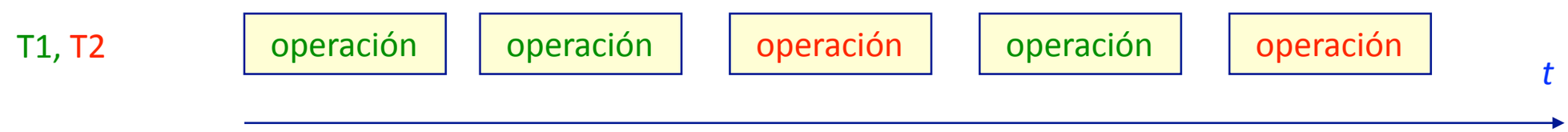


¿Cómo funciona?

- En un sistema con varios procesadores se puede ejecutar cada tarea en un procesador
  - ejecución simultánea (paralelismo físico)



- En un monoprocesador se intercalan las operaciones de las tareas
  - multiplexado en tiempo (paralelismo lógico)



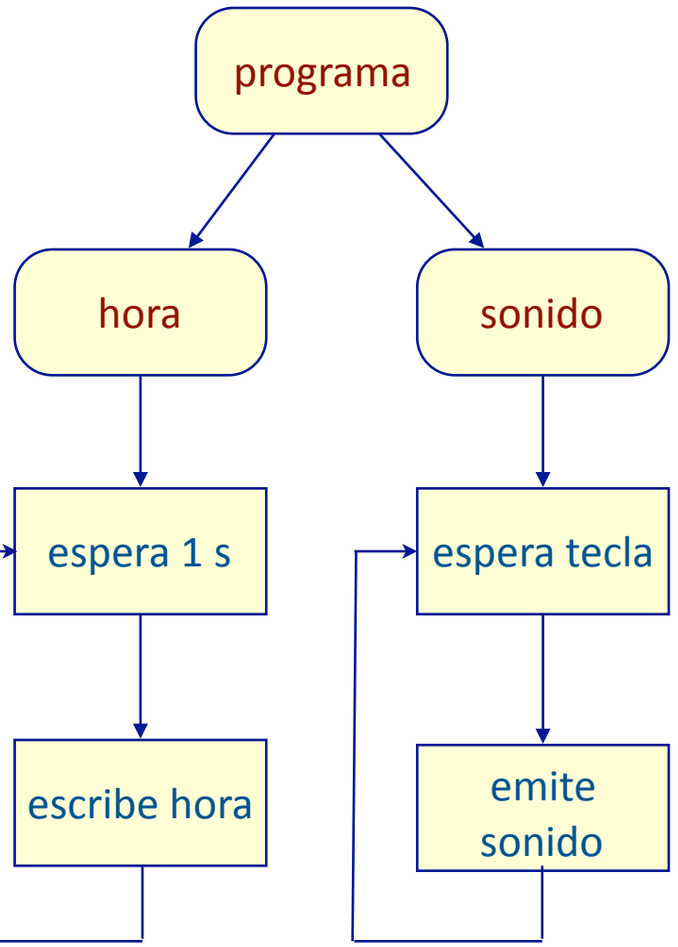
- La máquina virtual (o el sistema operativo) se encarga de los detalles
- Desde un punto de vista lógico son equivalentes

Aplicaciones

- Interfaces de usuario reactivas
  - atención a sucesos asíncronos
  - gestión de ventanas, *widgets*, etc.
- Servidores
  - atención a múltiples clientes
  - gestión de protocolos de comunicación
- Mejoras de prestaciones
  - ejecución en multiprocesadores
- Cálculos complejos
  - ejecución de algoritmos en paralelo

Ejemplo

- Programa con dos actividades:
  - escribir la hora cada 1 s
  - emitir un sonido cuando se pulsa la tecla *intro*
- Sería muy complicado hacerlo con un programa secuencial
  - la tecla se puede pulsar en cualquier momento
    - es un suceso asíncrono
    - difícil de mezclar con la escritura de la hora
- Mejor con dos hebras



Ejemplo (continuación)

```
/* hora */
while(true) {
  sleep(1000);
  System.out.println(
    new Date().toString());
}
```

```
/* linea */
while(true) {
  nextLine();
  beep();
}
```

- Veremos el programa completo más adelante

Hebras (*threads*) en Java

Threads en Java

- En Java las actividades concurrentes se llaman *threads* (hebras, hilos)
- Se crean extendiendo la clase Thread

```
class Tarea extends Thread {
  @Override
  public void run() {…}   // código concurrente
}
Tarea t = new Tarea();
```

- Es obligatorio redefinir el método run()
  - contiene el código que se ejecuta concurrentemente con otras hebras

Arrancar una hebra

- Hay que arrancar la ejecución de cada hebra para que empiece a ejecutarse
  - método `start()`

- Ejemplo

```
Tarea t = new Tarea(); // se crea la hebra t
...
t.start();             // se empieza a ejecutar
```

  - ahora se ejecutan a la vez el método `t.run()` y el método `main()`

Hebras en un programa

- ¿Cuántas hebras hay en un programa?
  - una hebra inicial que ejecuta el método main()
  - todas las que se arranquen en el programa con start()
  - la máquina virtual y la interfaz gráfica pueden crear hebras adicionales

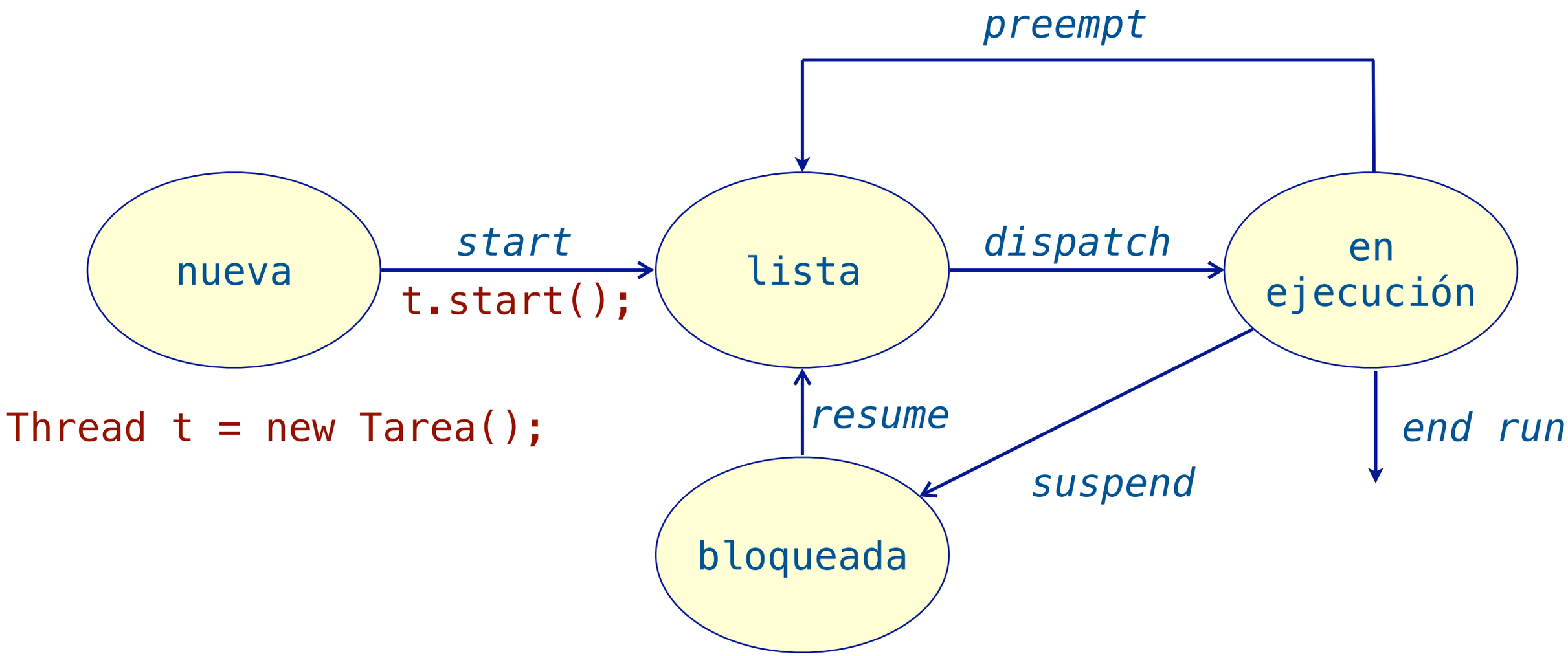
Terminación de hebras y programas

- Una hebra termina cuando termina el método run()
  - o cuando se fuerza su terminación mediante una interrupción
    - lo veremos más tarde

- Un programa termina cuando terminan todas sus hebras
  - cuando termina main() y todas las demás hebras que hayan arrancado en el programa

- El programa también termina si
  - se hace exit() desde alguna hebra
  - se lanza una excepción que se propaga fuera de run() o main()

Estados de las hebras




```
Thread t = new Tarea();
```

Ejemplo: Hora

```
import java.util.*;
public class Hora extends Thread {

  @Override
  public void run()  {                  /* código concurrente */
    try {
      while (true) {
        sleep(1000);                    /* esperar 1000 ms */
        System.out.println(new Date().toString());
      }
    } catch (InterruptedException e) {
        return;                         /* terminar esta hebra */
    }
  }
}
```

Ejemplo: Sonido

```
import java.awt.Toolkit;
import java.util.Scanner;
public class Sonido extends Thread {

   @Override
   public void run() {                    /* código concurrente */
      Scanner sc = new Scanner(System.in);
      while(true) {
         sc.nextLine();
         Toolkit.getDefaultToolkit().beep();
      }
   }
}
```

Ejemplo: Reloj

```
public class Reloj {

  public static void main(String[] args) {
      Hora hora   = new Hora () ;
      Sonido sonido = new Sonido();
      hora.start();
      sonido.start();
    }
  }
}
```

Ejercicio

- Creación y activación de hebras
  - Importar el proyecto *Threads* del repositorio https://github.com/ALED-UPM/Tema3
  - Ejecutar el programa *Reloj*
  - Añadir una hebra que escriba “Hola” en el terminal cada 5 s, y volver a ejecutar el programa con esta modificación

Otras formas de crear hebras

La clase *java.lang.Thread*

```
public class Thread  implements Runnable {

  /* constructores */
  public Thread();
  public Thread(Runnable target);

  /* código concurrente */
  void run();

  /* arrancar la ejecución */
  void start();

  /* suspender la ejecución durante un tiempo */
  static native void sleep(long millis)
    throws InterruptedException;

  /* esperar que termine la hebra */
  void join() throws InterruptedException;

  /* interrumpir la ejecución de la hebra */
  public void interrupt() throws SecurityException;
  ...
}
```

La interfaz *Runnable*

- Java sólo admite herencia simple
  - si se hereda de `Thread` no se puede heredar de otra clase
- `java.lang.Runnable` es una interfaz que incluye el método `run`
  - permite crear hebras sin heredar de `Thread`

```
public interface Runnable {
   public void run ();
}
```

```
class Actividad implements Runnable {
  public void run() {…}
}
Thread t = new Thread(new Actividad());
```

Ejemplo: Hora2

```
import java.util.*;
public class Hora2 implements Runnable  {

  public void run()  {
    try {
      while (true) {
        Thread.sleep(1000);
        System.out.println(new Date().toString());
      }
    } catch (InterruptedException e) {
        return;
    }
  }
}
```

Ejemplo: Reloj2

```
public class Reloj2 {
  public static void main(String[] args) {
    Runnable escribeHora = new Hora2();
    /* no es un thread */
    Thread hora = new Thread(escribeHora);
    hora.start();
    /* ahora sí */
    ...
  }
}
```

Objetos *Runnable* internos

```
public class Reloj3 {

  public static void main(String[] args) {
    Runnable escribeHora = new Runnable () {
      public void run()  {...}
    };
    new Thread(escribeHora).start();
    ...
  }
}
```

- El método run queda oculto
  - no se puede invocar desde fuera

Método *run* interno con *Thread*

```
public class Reloj4 {

  public static void main(String[] args) {
    Thread hora = new Thread () {
      public void run()  {
        ...
      }
    };
    hora.start();
    ...
  }
}
```

- Otra forma de declarar el método run para que quede oculto

Más compacto todavía

```
public class Reloj5 {

  public static void main(String[] args) {
    new Thread () {
      public void run()  {
        ...
      }
    }.start();
    ...
  }
}
```

Interrupciones

Método *interrupt*

- Método de la clase *Thread*
- Uso habitual: despertar asíncronamente a una hebra bloqueada
- Si la hebra está bloqueada en un *wait*, *join* o *sleep*, se eleva la excepción InterruptedException
  - Incluir la sentencia de bloqueo un bloque *try/catch*
  - Indicar que el método puede elevar InterruptedException
- Si la hebra está haciendo otra cosa, se cambia un indicador
  - consultar con *interrupted()*
- Si la hebra está bloqueada en otras operaciones, la excepción que se eleva es diferente
  - Ej. bloqueo en operación de E/S

Ejemplo: Segundero

```
public class Segundero extends Thread {

  public void run() {
    int segundos = 0;
    try {
      while (true) {
        sleep(1000); // La hebra se bloquea un segundo
        segundos++;
        System.out.println("Segundos: " + segundos);
       }
    } catch (InterruptedException e) {
      System.err.println("Hebra interrumpida");
      return;  // La hebra ha sido desbloqueada mediante la
    }          // invocación de interrupt
  }
}
```

Ejemplo: Temporizador

```
import java.util.*;
public class Temporizador {

 public static void main (String args[]) {
    // Segundos de espera hasta terminar
    int tiempoEspera = 2;
    Segundero segundero = new Segundero();
    segundero.start();
    try {
       Thread.sleep(tiempoEspera * 1000);
       segundero.interrupt(); // Fin de la espera
    } catch (Exception e) {
       System.err.println("Error esperando ");
    }
  }
}
```

Detener una hebra

- Lanzar una interrupción no es una buena forma de detener la ejecución
- Mejor algo así:

```
public class Segundero extends Thread {
   private boolean activo = true;

   public void run() {
     while (activo) {
        ...
     }
   }
   public void detener() {
    activo = false;
   }
}
```

Problemas

- La hebra que llama a detener() es distinta que la que ejecuta run()
  - por ejemplo, la hebra principal y la hebra *segundero*
- La variable `activo` se puede modificar mientras se está leyendo
  - no es seguro que lo que lee *segundero* sea el valor correcto
  - muchos problemas, los veremos en detalle
  - de momento, la solución es declarar que `activo` es volátil

```
private volatile boolean activo = true;
```

- Lo estudiaremos a fondo más adelante

Ejercicio

- Detener la ejecución de una hebra
  - Modificar las clases *Segundero* y *Temporizador* para detener la cuenta de segundos mediante una variable, como en los ejemplos anteriores
  - ¿Tiene sentido mantener el manejador de `InterruptedException` en *Segundero*? ¿Por qué?

Propiedades de los programas concurrentes

Propiedades de los programas

- Corrección (*correctness*)
  - el resultado es correcto
- Seguridad (*safety*)
  - nunca pasará nada malo
- Vivacidad (*liveness*)
  - en algún momento se hará lo que se tiene que hacer
- Equidad (*fairness*)
  - todas las hebras tienen la posibilidad de avanzar

Pruebas

- Algunas propiedades son difíciles de probar
- A veces se puede hacer con matemáticas
  - como demostraciones de teoremas
  - examinando todas las secuencias de ejecución posibles
- Es necesario poner mucha atención al escribir el programa para estar seguros de que es correcto

Resumen

Resumen

- Los programas concurrentes tienen varias hebras de ejecución (*threads*) que avanzan a la vez
- En Java las hebras son objetos de una clase
  - que extiende Thread
  - que implementa Runnable
- El método run() determina qué hace la hebra
- Para arrancar una hebra t se hace t.start()
- Una hebra termina cuando se llega al final de run()
  - también por interrupciones
- Un programa termina cuando terminan todas sus hebras
  - también la hebra inicial que ejecuta main()

Ejemplos

- Código en github
  - https://github.com/ALED-UPM/Tema3.git