Análisis y diseño de software

dit
UPM

Tema 2: Concurrencia /threads (java)

José A. Mañas

11.3.2018

ETSIT UPM

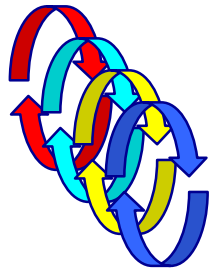
referencias

- The Java tutorials: Concurrency
  - Oracle
- *Java Threads*
  - Scott Oaks & Henry Wong
    O'Reilly Media; 3rd ed., 2004
- Java Concurrency in Practice
  - B. Goetz et al.
    Addison-Wesley, 2006
- java – vademécum / concurrencia
  - José A. Mañas

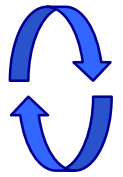
índice

- concurrencia
  - modelo java: 1 RAM + N aplicaciones ligeras
- Theads
  - clases Thread & Runnable
  - arrancar threads
  - parar threads
- propiedades de la concurrencia
  - corrección (correctness)
  - seguridad (safety)
  - vivacidad (liveness)
  - equidad (fairness)

programación

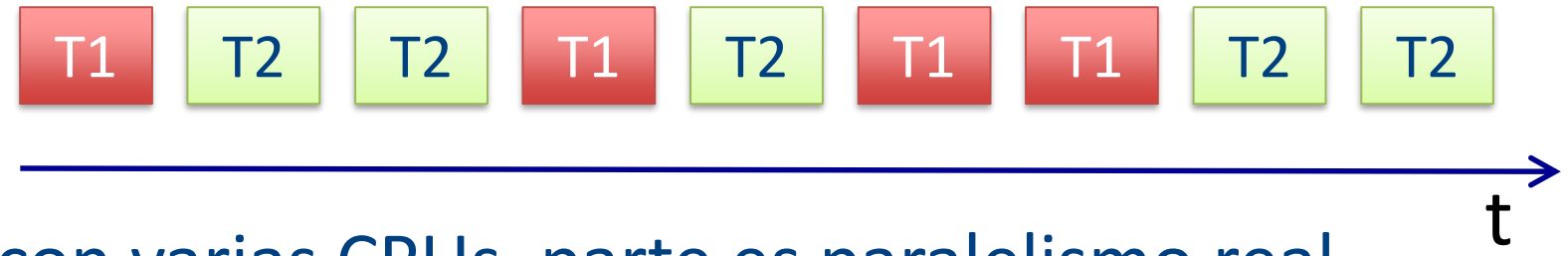


- programas concurrentes
  - describen varias actividades que podrían progresar al mismo tiempo
  - el flujo de ejecución admite muchas opciones
- programas secuenciales
  - describen 1 actividad
  - sólo hay 1 flujo de ejecución: seguir la secuencia de sentencias



concurrencia

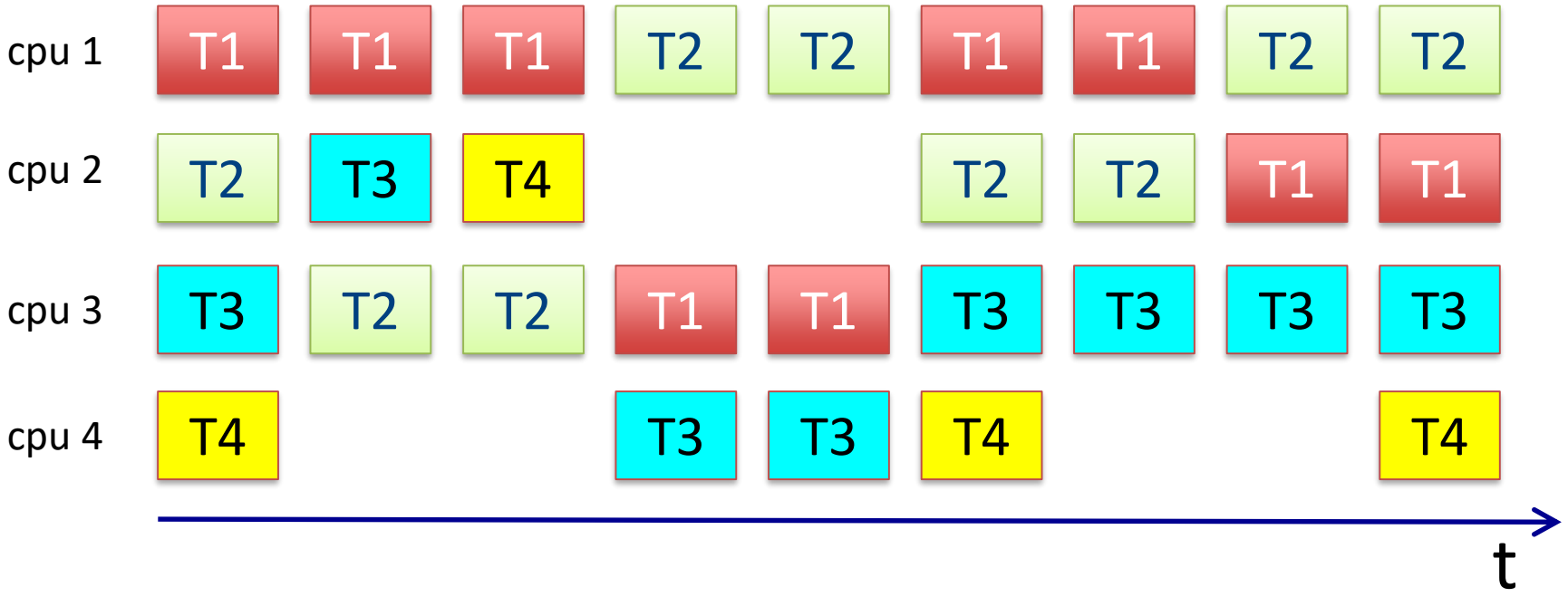
- se dice que N tareas ejecutan concurrentemente cuando se reparten el tiempo y/o la CPU para ir ejecutando
- con 1 CPU, la concurrencia es “simulada”
  - un ratito la CPU para mi, un ratito para ti



- con varias CPUs, parte es paralelismo real, parte es paralelismo simulado repartiendo t

concurrencia

- con varias CPUs, parte es paralelismo real, parte es paralelismo simulado repartiendo t



¿para qué?

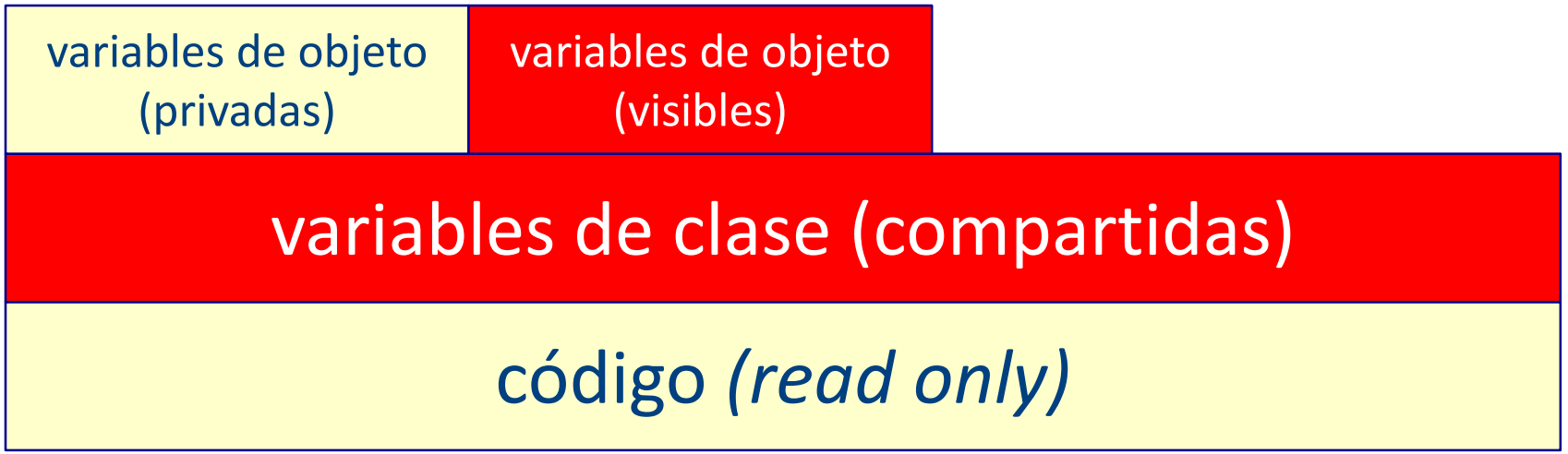
- para aprovechar mejor las CPUs
- para poder dejar una tarea esperando algo mientras otras pueden ir progresando
  - ej. interfaces de usuario reactivas (siempre están dispuestas)
  - ej. servidores con varios clientes (ej. www)
  - ej. sistemas de control industrial
- conceptual
  - porque la solución ‘natural’ es concurrente
  - porque el mundo real es concurrente

implicaciones

- concurrencia implica
  - competencia
    - por los recursos comunes
  - sincronización
    - para coordinar actividades
  - cooperación
    - para intercambiar información
- ejemplo:
  - 2 amigos preparando la comida
  - trenes

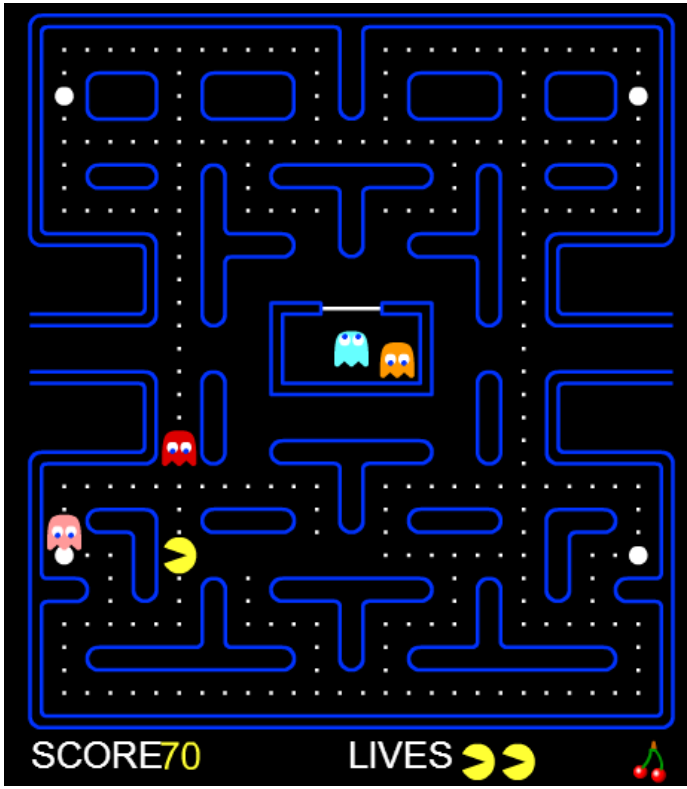
java threads

- tareas ligeras
  - memoria RAM común
  - con las reglas de visibilidad de java
    - private, de paquete, públicas



java threads

- muchos actores
  - ejecutando el mismo código
  - sobre la misma RAM
  - encuentran diferentes datos y siguen diferentes recorridos



- competir
- sincronizar
- cooperar

java threads

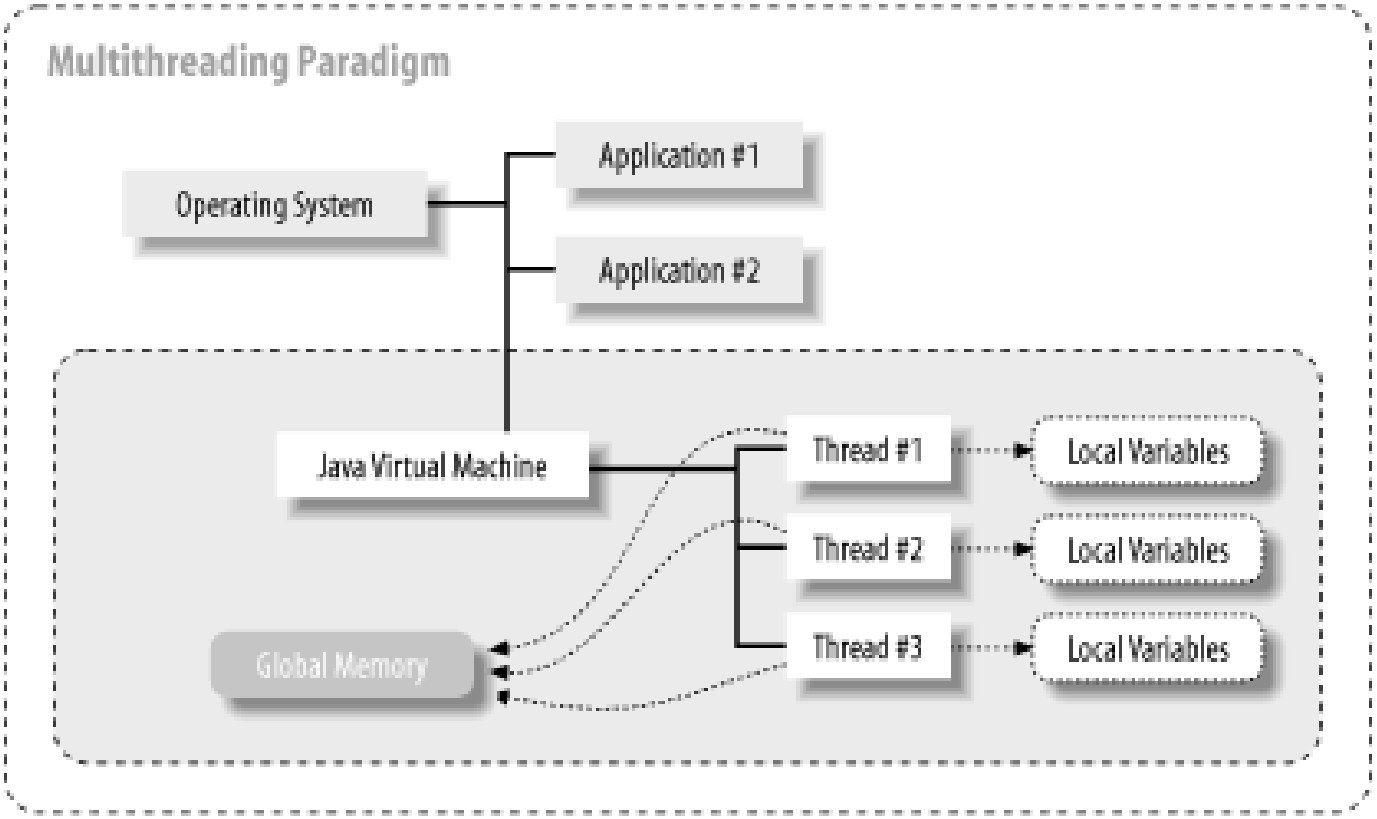


Figure 2-2. Threads in a multithreaded environment

Java Threads - Scott Oaks & Henry Wong

threads

opción 1

```
class MyR implements Runnable {
    public void run() { ... }
}
```

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

opción 2

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

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

```
t.start();
```

ejemplo

```
public class Reloj
      implements Runnable {
   private final String id;
   private final int dt;

   public Reloj(String id, int dt) {
      this.id = id;
      this.dt = dt;
   }
```

```
   @Override
   public void run() {
      while (true) {
         System.out.println(id + ": " + new Date());
         nap(dt * 1000);
      }
   }
```

```
   private void nap(long ms)  {
      try {
         Thread.sleep(ms);
      } catch (InterruptedException ignored) {
      }
   }
```

ejemplo

```
public class ECO implements Runnable {
   private final String id;
   private final BufferedReader console;

   public Eco(String id) {
      this.id = id;
      console = new BufferedReader(new InputStreamReader(System.in));
   }
```

```
   public void run() {
      try {
         while (true) {
            String line = console.readLine();
            if (line.equals("."))  break;
            System.out.printf("%s: %s: %s%n", new Date(), id, line);
         }
         System.out.println(id + ": muerto soy.");
      } catch (IOException e) {
         System.out.println(id + ": " + e);
      }
   }
```

ejemplo: uso

```
Thread R1 = new Thread(new Reloj("A1", 3));
Thread R2 = new Thread(new Reloj("A2", 5));

Thread E = new Thread(new Eco("C"));

R1.start();
R2.start();
E.start();
```

¿cuándo termina un thread?

- cuando termina el método run()

```
public class Tarea extends Thread {
        private volatile boolean funcionando = true;

        @Override
        public void run() {
                while (funcionando) {
                        … … …
                }
        }

        public void parar() {
                funcionando = false;
        }
}
```

volátil

- algunos campos pueden declararse
  - volatile
- con esto se le dice a la máquina virtual java
  - esa variable puede cambiar en cualquier momento
  - no la metas en tu memoria privada (por ejemplo, en la caché de la CPU)
- consecuencia
  - quien lee la variable seguro que lee lo último que alguien haya escrito en ella

ejemplo

```
public class Reloj
      implements Runnable {
   private final String id;
   private final int dt;
   private volatile boolean running;

   public Reloj(String id, int dt) {
      this.id = id;
      this.dt = dt;
   }
```

```
   public void parar() {
      running = false;
   }
```

```
   @Override
   public void run() {
      running = true;
      while (running) {
         System.out.println(id + ": " + new Date());
         nap(dt * 1000);
      }
   }
```

executors

- Se intenta tratar por separado 2 aspectos
  - preparación de threads
  - ejecución de threads

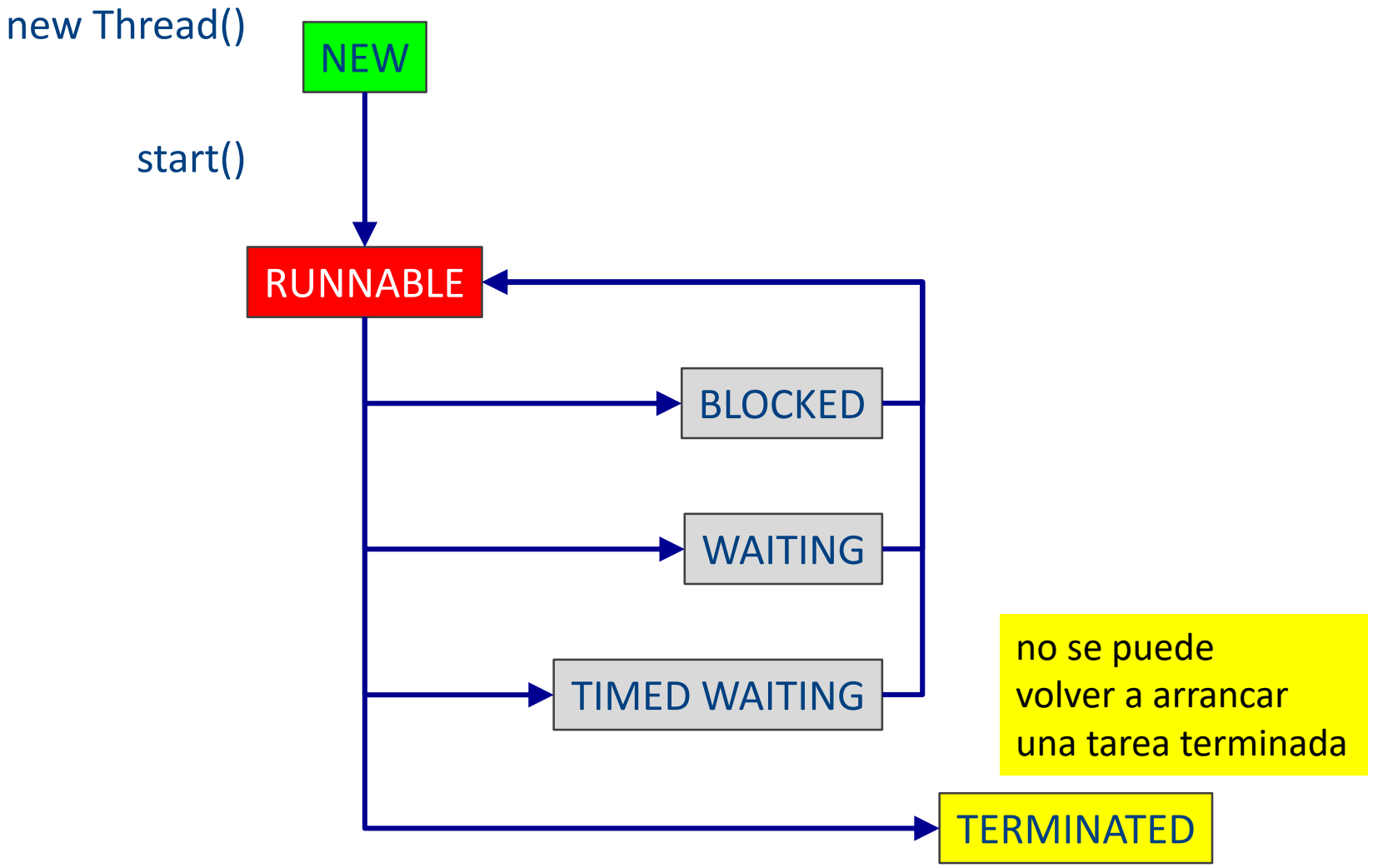
```
Executor executor = ...;
executor.execute(new RunnableTask1());
executor.execute(new RunnableTask2());
...
```

  - ... y puede haber diferentes políticas de lanzamiento
    - en paralelo | una tras otra | de n en n | ...

executors

- ExecutorService executor =
  - Executors.newSingleThreadExecutor();
  - Executors.newFixedThreadPool(5);
  - Executors.newScheduledThreadPool(10);
  - … … …

state


new Thread()
NEW
start()
RUNNABLE
BLOCKED
WAITING
TIMED WAITING
TERMINATED
no se puede
volver a arrancar
una tarea terminada

¿cuántas threads?

- el programa principal
  + las tareas que arrancamos (hasta que terminen)
  + lo que se le ocurre al IDE

```
Set<Thread> threadSet =
    Thread.getAllStackTraces().keySet();
for (Thread thread : threadSet)
    System.out.println(thread);
```

propiedades

- corrección *(correctness)*
  - da el resultado correcto
- seguridad (safety)
  - no pasa nada malo
- vivacidad (liveness)
  - en algún momento hace lo que se espera
- equidad (fairness)
  - las threads se reparten la CPU con justicia

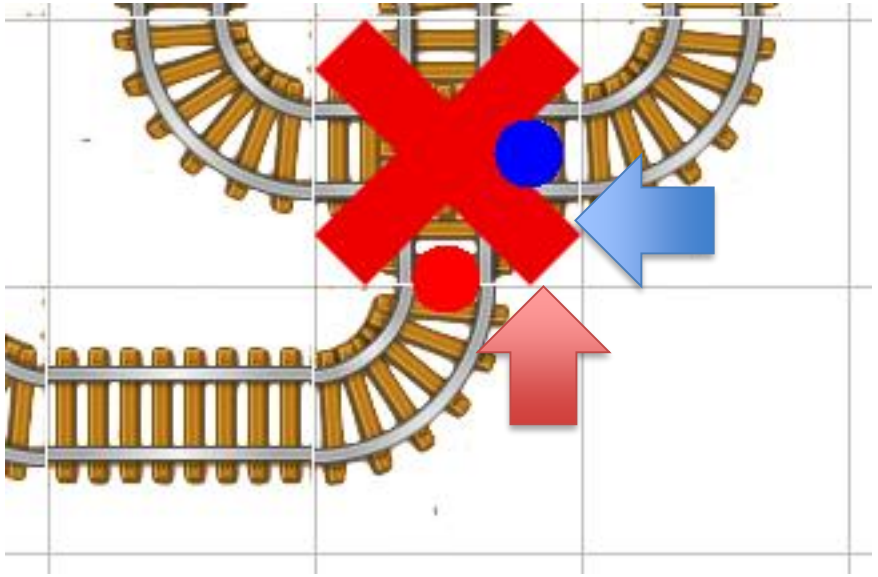
corrección

- se genera el resultado correcto
- se puede usar JUnit
  pruebas unitarias
  - assertEquals(esperado, obtenido)

- OJO debe generar SIEMPRE el resultado correcto
  - problema: indeterminismo:
    - a veces sí, a veces no

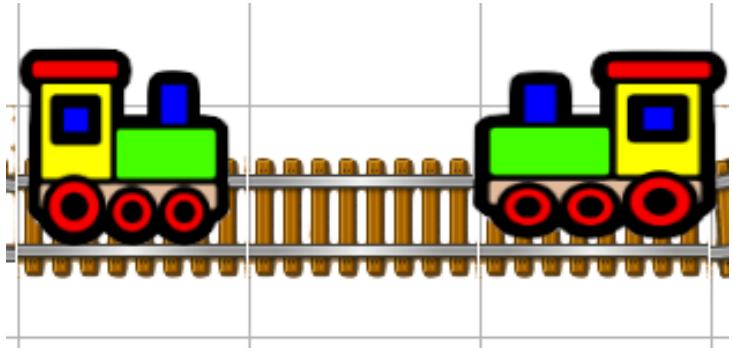
seguridad - *safety*

- *nothing bad ever happens*
  nunca funciona mal
- cosas que pueden ir mal
  - carreras → valores incorrectos



vivacidad - *livenesss*

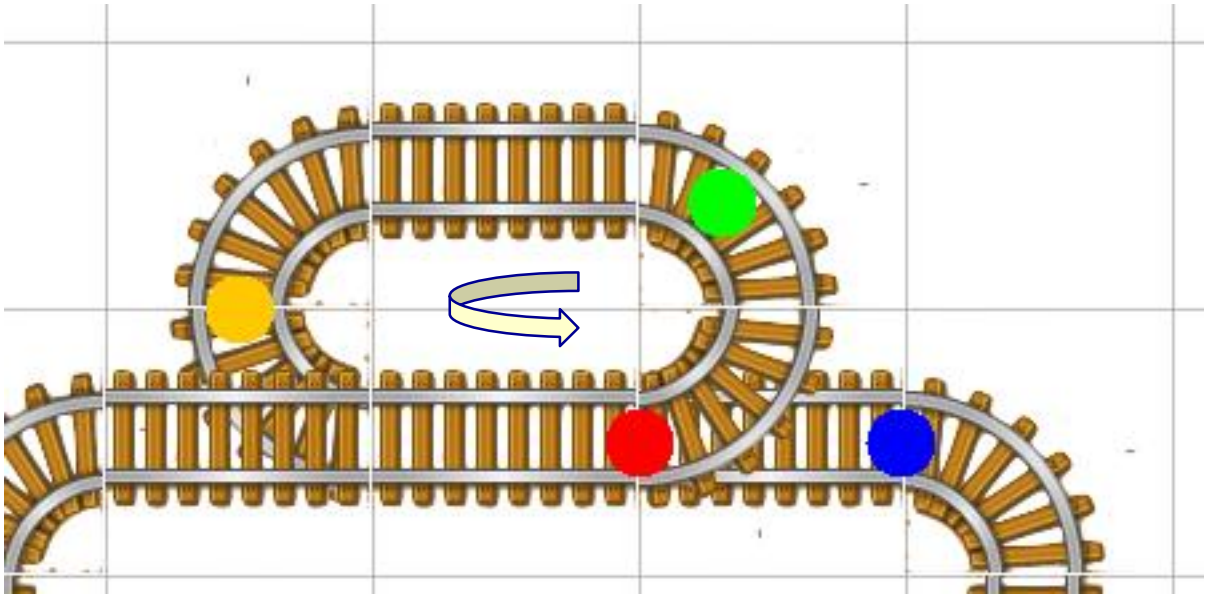
- *something good eventually happens*
  en algún momento hace lo correcto
- cosas que pueden ir mal
  - *deadlock* – interbloqueo
    - el programa se queda muerto



  - *livelock* – bloqueo activo
    - el programa da vueltas intentando evitar un deadlock; pero el resultado es dar vueltas sin sentido: círculo vicioso

equidad - *fairness*

- los recursos se reparten con justicia
- lo contrario
  - inanición – *starvation*
    - a alguna thread nunca se le da oportunidad



java threads

- tareas ligeras (light weight concurrency)
  - memoria RAM común
  - con las reglas de visibilidad de java
    - private, **de paquete, públicas**

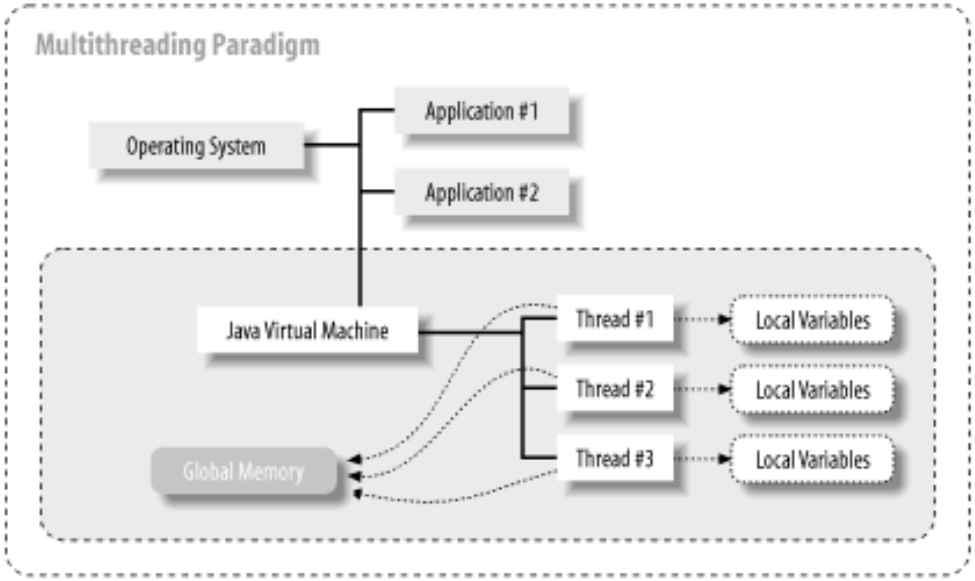


Figure 2-2. Threads in a multithreaded environment

memoria

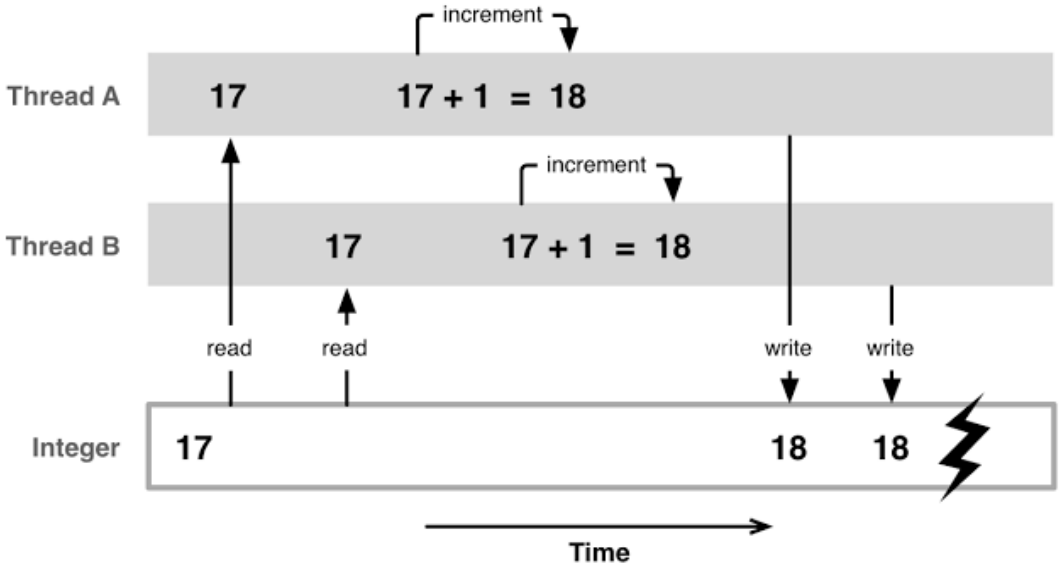
- la memoria común
  - + bueno: es muy eficiente cambiar de thread: ligeros
  - + bueno: es muy fácil compartir datos entre threads
  - – malo: es muy fácil que un thread corrompa los datos de otro

memoria: variables locales

- cada thread tiene sus variables locales
  - parámetros o argumentos de métodos
  - variables creadas dentro de los métodos
  - en principio son privadas;
    pero puede haber compartición si se pasan referencias
    - objetos compartidos
    - arrays compartidos

memoria compartida: carreras

- carreras *(race conditions)*
  1. yo me traigo el dato a mi memoria local
  2. tu te llevas el dato a tu memoria local
  3. modificamos nuestros datos
  4. yo lo devuelvo, tu lo devuelves
  5. ¿quién tiene la última palabra?



pruebas

- es muy difícil hacer pruebas de las propiedades
  - a veces se consigue demostrar usando técnicas matemáticas
    - demostración de teoremas
    - análisis de todas las ejecuciones posibles
- recomendación
  - programar de forma muy bien estructurada para asegurarnos de que no hay malas combinaciones