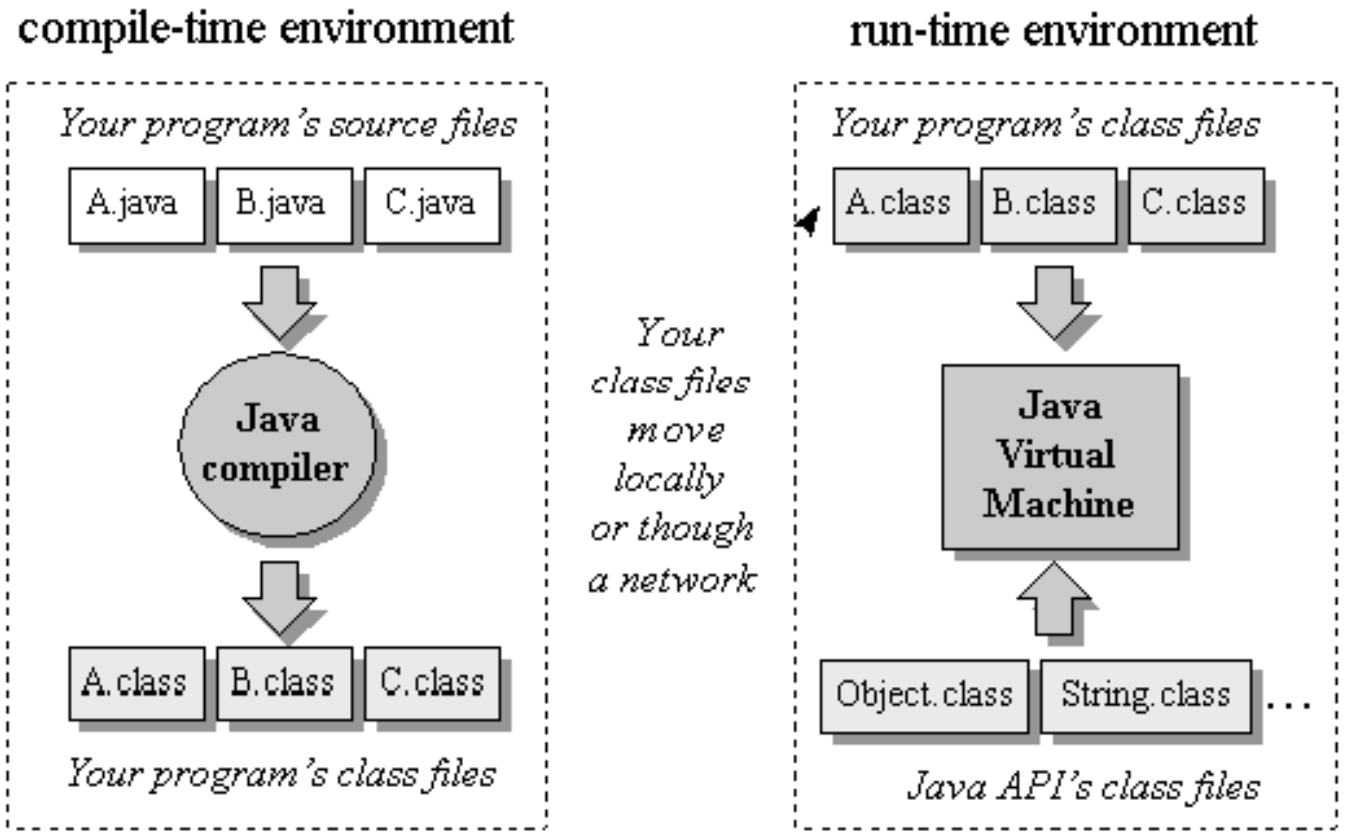
# La Máquina Virtual Java

Sincronización de hilos

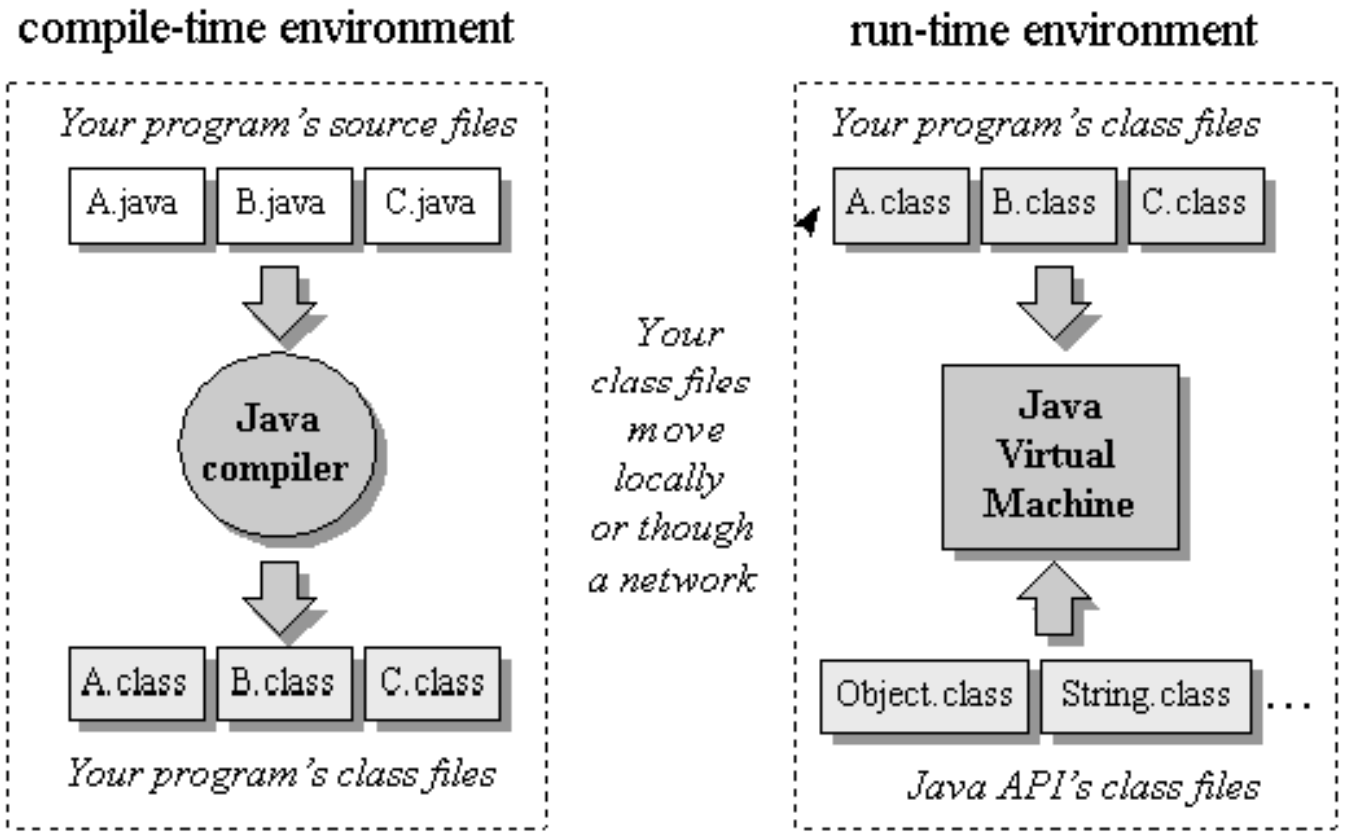
# ¿Dónde esta la MVJ?

- La arquitectura de Java cuenta con cuatro tecnologías distintas, relacionadas entre sí



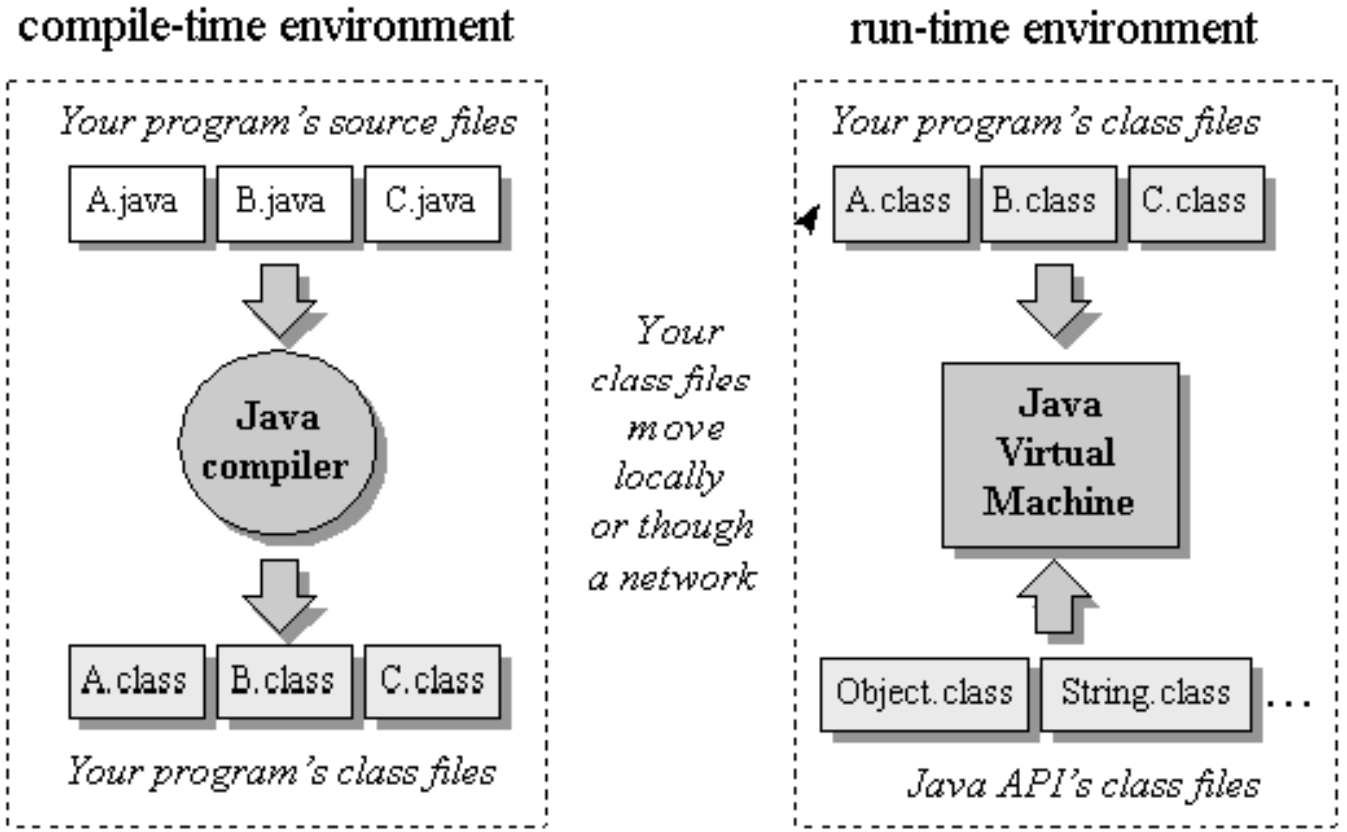
# ¿Dónde esta la MVJ?

- Un programa Java esta expresado en archivos escritos en el *lenguaje de programación Java*.
- Posteriormente se compilan generando a *los archivos de clase Java*.
- Se ejecutan por la *Máquina Virtual de Java*.



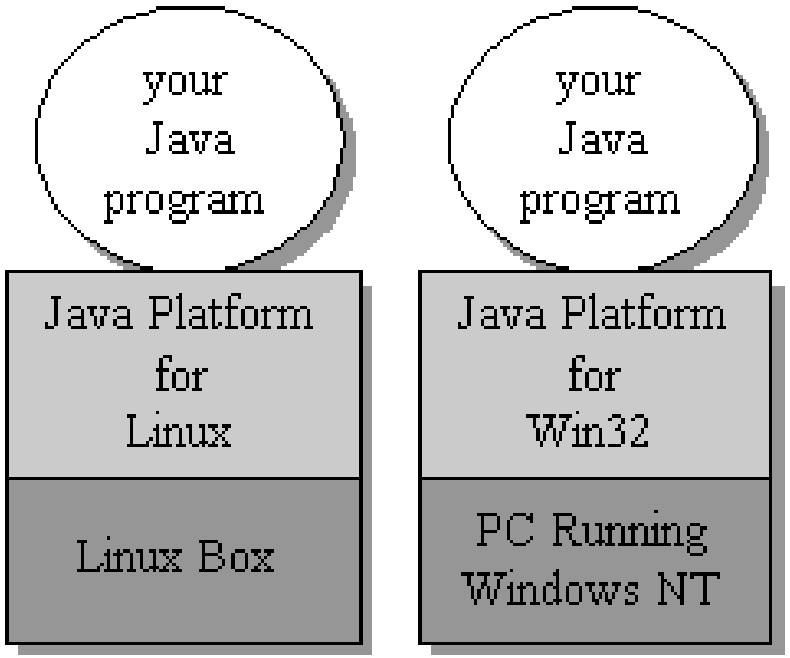
# ¿Dónde esta la MVJ?

- Antes de, el programa debe de ser escrito
- Obteniendo acceso a determinados recursos del sistema, llamando a métodos de clases que utilizan *la interfaz de programación de aplicaciones Java* (API Java).



# Java Platform

- El Conjunto entre la MVJ y el API de Java
- Portabilidad: Los programas Java pueden ser ejecutados en cualquier tipo de computadora
- Implementación en Software



# Monitores

## Exclusión mutua

- Los subprocesos trabajan de forma independiente
- No existe interacción entre ellos
- Implementada mediante el *bloqueo de objetos*

## Cooperación

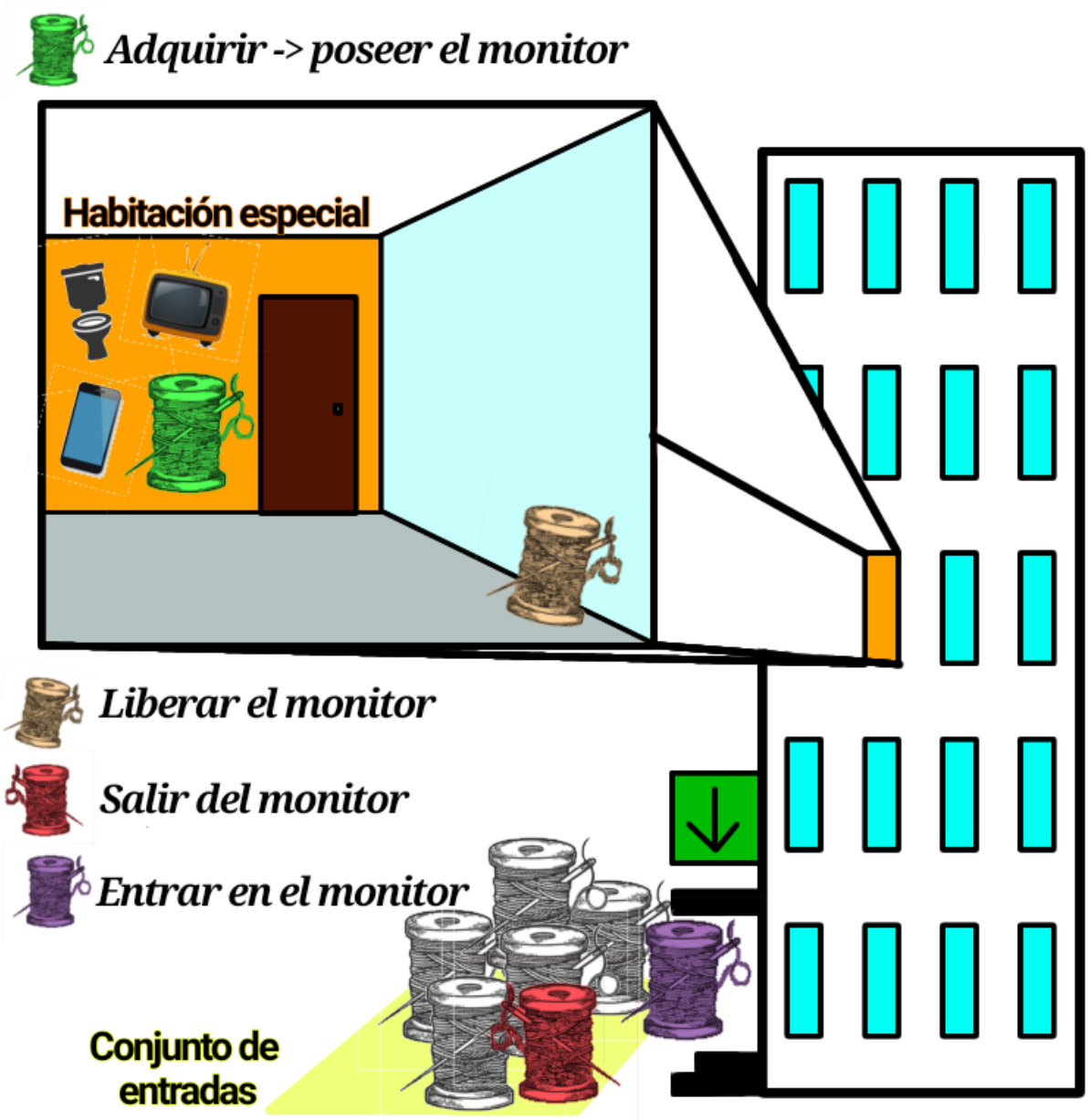
- Los subprocesos trabajan en conjunto para lograr un objetivo en común

# Monitores

- Cada objeto tiene un bloqueo
- Y un conjunto de instrucciones para manipularlo
  - wait
  - notify
  - notifyAll
  - Thread.interrupt

# Monitor

- Cualquier entidad que posea bloqueos y conjuntos de espera
- Cualquier objeto puede servir como *monitor*

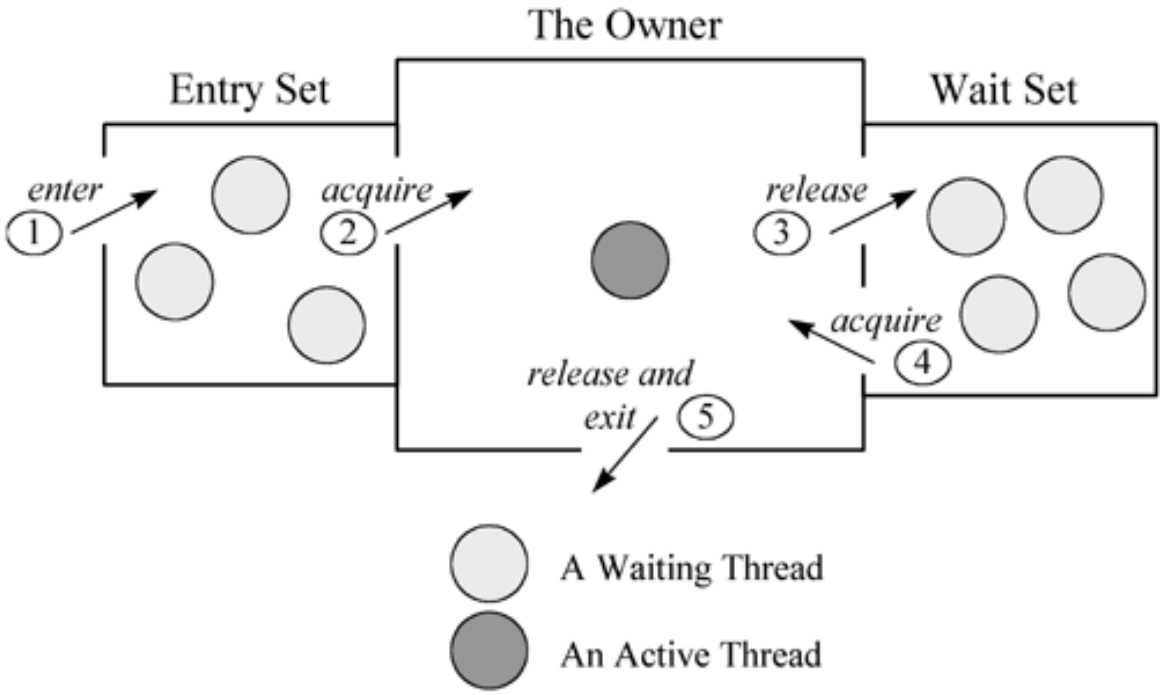


# Monitor Regions

- Código que debe ejecutarse como indivisible con respecto a un monitor
- Para adquirir el monitor, se debe llegar a esta región del código
- Para ejecutar el código, se debe de adquirir el monitor
- Acceso exclusivo de los datos hasta que se libere el monitor


# Monitor de la MVJ

- *Notify and wait*
- Un proceso con el monitor adquirido puede suspenderse (wait)(→*conjunto de espera*)
- Otro hilo *notificará* al hilo en espera (mientras tiene al monitor)
- Cuando se libere el monitor, el hilo en espera *resucitará* y adquirirá al monitor

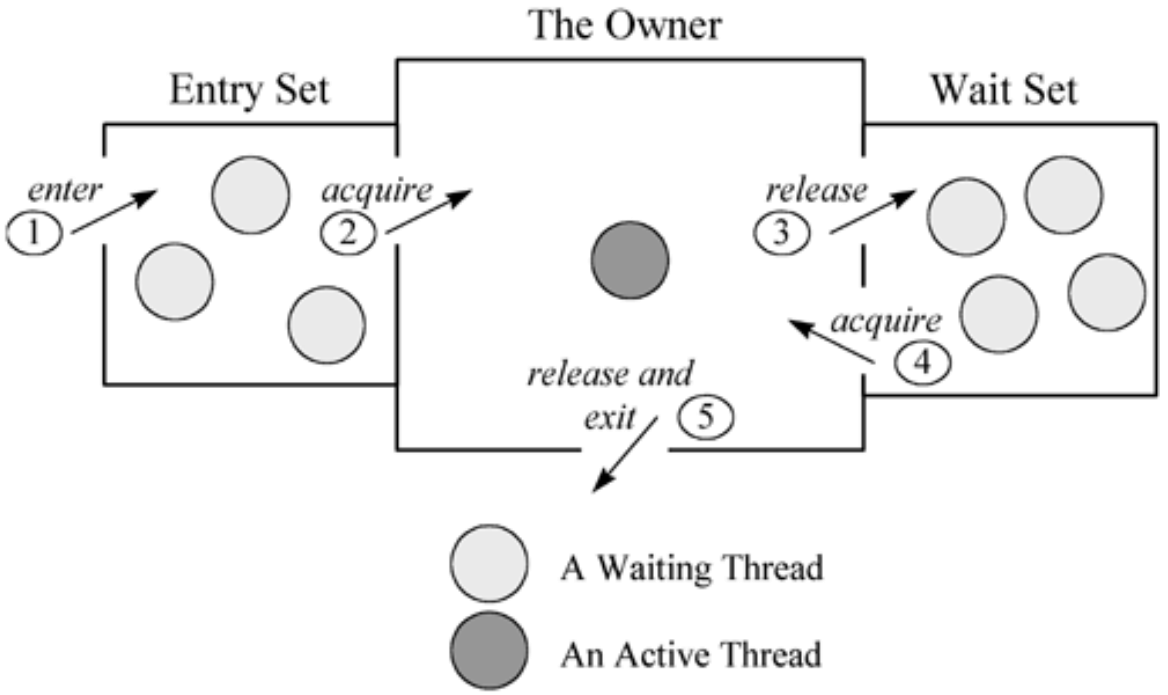


# Monitor de la MVJ

- Hilos del *conjunto de entrada* competirán para adquirir el monitor

Ó

- Hilos del *conjunto de entrada* competirán con uno o varios del *conjunto de espera* para adquirir el monitor



# Monitor de la MVJ

- Los hilos que lancen un *wait* pueden especificar un tiempo de espera
- Si el hilo en *espera* no recibe *notificación* en el tiempo especificado, la MVJ lo *notifica* automáticamente, y el hilo *resucitará*

### notify

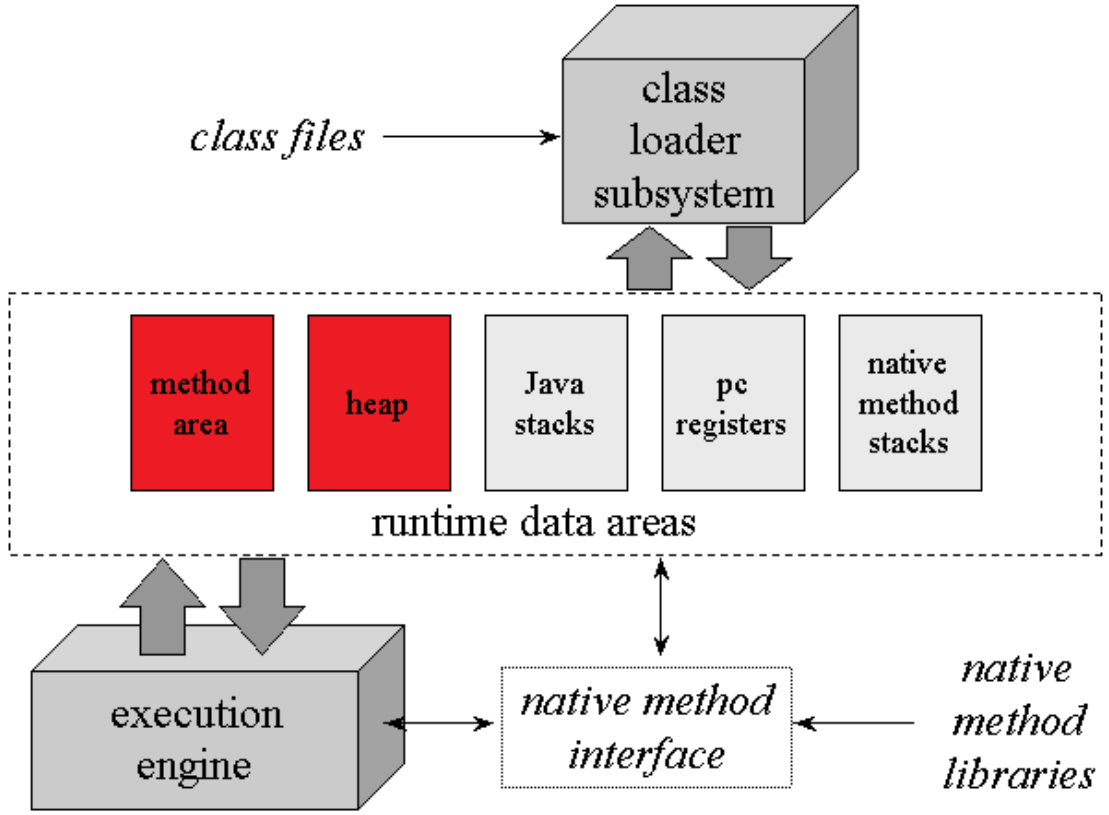
- La MVJ selecciona un hilo aleatoriamente para *resucitarlo*

### notifyAll

- La MVJ notifica a todos los hilos para su eventual resucitación

# Object locking

- *Method area* (<- variables de instancia) y *heap* (<- variables de clase) son compartidos por todos los subprocesos
- Cada objeto y clase se asocia lógicamente con un monitor
- Para la exclusión mutua, la MVJ asocia un bloqueo (*mutex*)

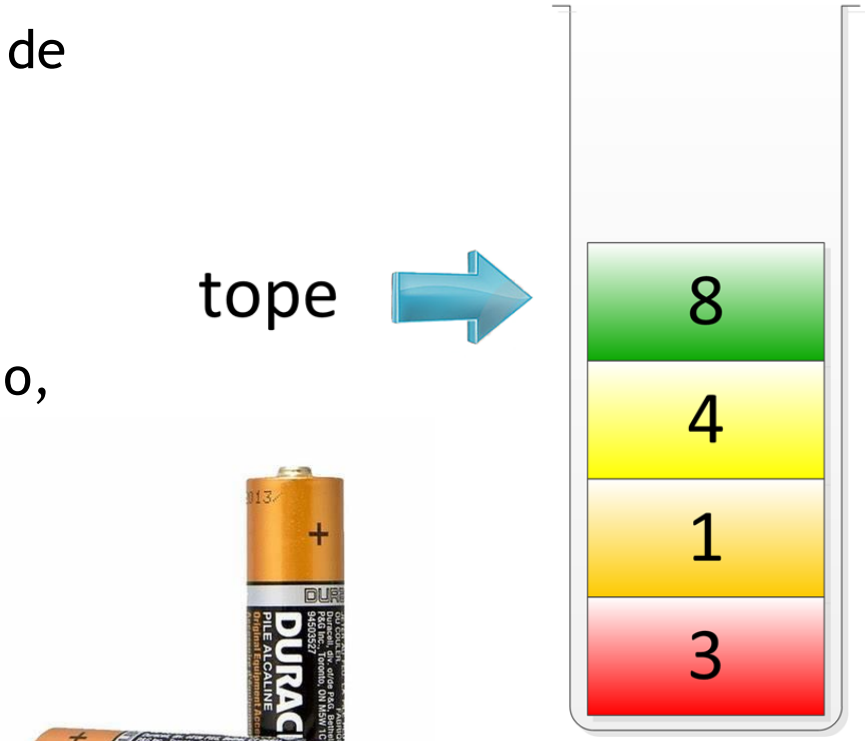


# Object locking

- Un subproceso puede bloquear un objeto varias veces
- La MVJ lleva un conteo de cuantas veces se ha bloqueado un objeto
- Cada bloqueo incrementa en 1 el contador
- Cuando se libera un bloqueo, el contador decrementa en 1
- Cuando el contador vale 0, el bloqueo se libera

# Instrucciones sincronizadas

- La MVJ es una MV orientada a la pila
- Casi todas las operaciones toman operandos de la pila de operandos
- O mandan resultados a la pila de operandos
- De acuerdo a la trama actual
- Una nueva trama se crea cada que se invoca un método, creando una nueva pila de operandos



# Instrucciones sincronizadas

- Toda instrucción de la MVJ generado por el JDK Oracle tiene la forma:

```
<índice> <opcode> [ <operando1> [ <operando2>... ]] [<comentario>]
```

- <*índice*> Contiene los bytes de código de la MVJ para un método. Es el mnemotécnico para el <*Opcode*> de la instrucción, y esta instrucción puede tener 0 o más <*operandos*>.
- Opcionalmente se tiene un comentario.

# Instrucciones sincronizadas

La sincronización se implementa mediante la entrada y salida al monitor

*monitorenter*

- Adquiere el bloqueo asociado al objeto

monitorexit

- Libera el bloqueo asociado al objeto

# Instrucciones sincronizadas

- Si se tiene un método como el siguiente:

```
void metodo (Foo f){
   synchronized (f) {
      HazAlgo();
   }
}
```

- La compilación para la MVJ queda de la siguiente forma:

```
Method void metodo(Foo)
0   aload_1              // Empuja a f
1   dup                  // Se duplica en la pila
2   astore_2             // Se guarda el duplicado en la variable local 2
3   monitorenter         // Se entra al monitor asociado con el objeto f
4   aload_0              // Con el monitor adquirido, se sigue a la siguiente instrucción
5   invokevirtual #5     // Se llama a HazAlgo ();
8   aload_2              // Se empuja la variable local 2
9   monitorexit          // Se libera el monitor asociado al objeto f
10  goto 18              // Se completa el método normalmente, y se salta a la línea 18
13  astore_3             // En caso de error, terminar aquí
14  aload_2              // Empujar la variable local 2
15  monitorexit          // Con error o no, se debe de liberar el monitor
16  aload_3              // Empujar el valor arrojado
17  athrow               // ...y volver a lanzar el valor al invocador
18  return               // Termina el método en el caso normal, o erróneo
```

# Métodos sincronizados

- Si un método esta sincronizado, la MVJ adquiere un mutex para el objeto antes de invocar a un método
- Para declarar un método sincronizado, se usa:

```
class nombre_clase{
    synchronized void nombre_método() {
        [Instrucciones]
    }
}
```


MUCHAS GRACIAS
POR SU ATENCION
PERO NO SE
PERMITEN PREGUNTAS