Introducción a JavaRT

Tema 8 - Programación Concurrente y de Tiempo Real

Antonio J. Tomeu[1] Manuel Francisco[2]

[1]Departamento de Ingeniería Informática
Universidad de Cádiz

[2]Alumno colaborador de la asignatura
Universidad de Cádiz

PCTR, 2016

Contenido

Principios Generales de los Sistemas RT

- Objetivo: dar respuesta a eventos procedentes del mundo real antes de un límite temporal establecido (*deadline*)
- Predictibilidad
  - Se cumplen los *deadlines* con independencia de
    - Carga de Trabajo
    - Número de Procesadores
    - Hilos y Prioridades
    - Algoritmos de Planificación
  - Determinismo
    - Funcionalidad
    - Rendimiento
    - Tiempo de Respuesta

Límites de Java Estándar en RT

- Gestión de Memoria
  - gc es impredecible. Se activa cuando quiere
  - Fragmenta la memoria (memoria no planificable)
- Planificación de Threads
  - Impredecible
  - No permite especificar cuándo un hilo ha de ejecutarse
  - Inversiones de Prioridad
- Sincronización
  - Duración de secciones críticas impredecible
  - Un hilo no sabe cuántas otras tareas compiten por un recurso ni su prioridad
- Gestión asíncrona de eventos (no está definida)
- Acceso a Memoria Física
  - Una tarea no sabe cuánta hay ni cuánta necesita

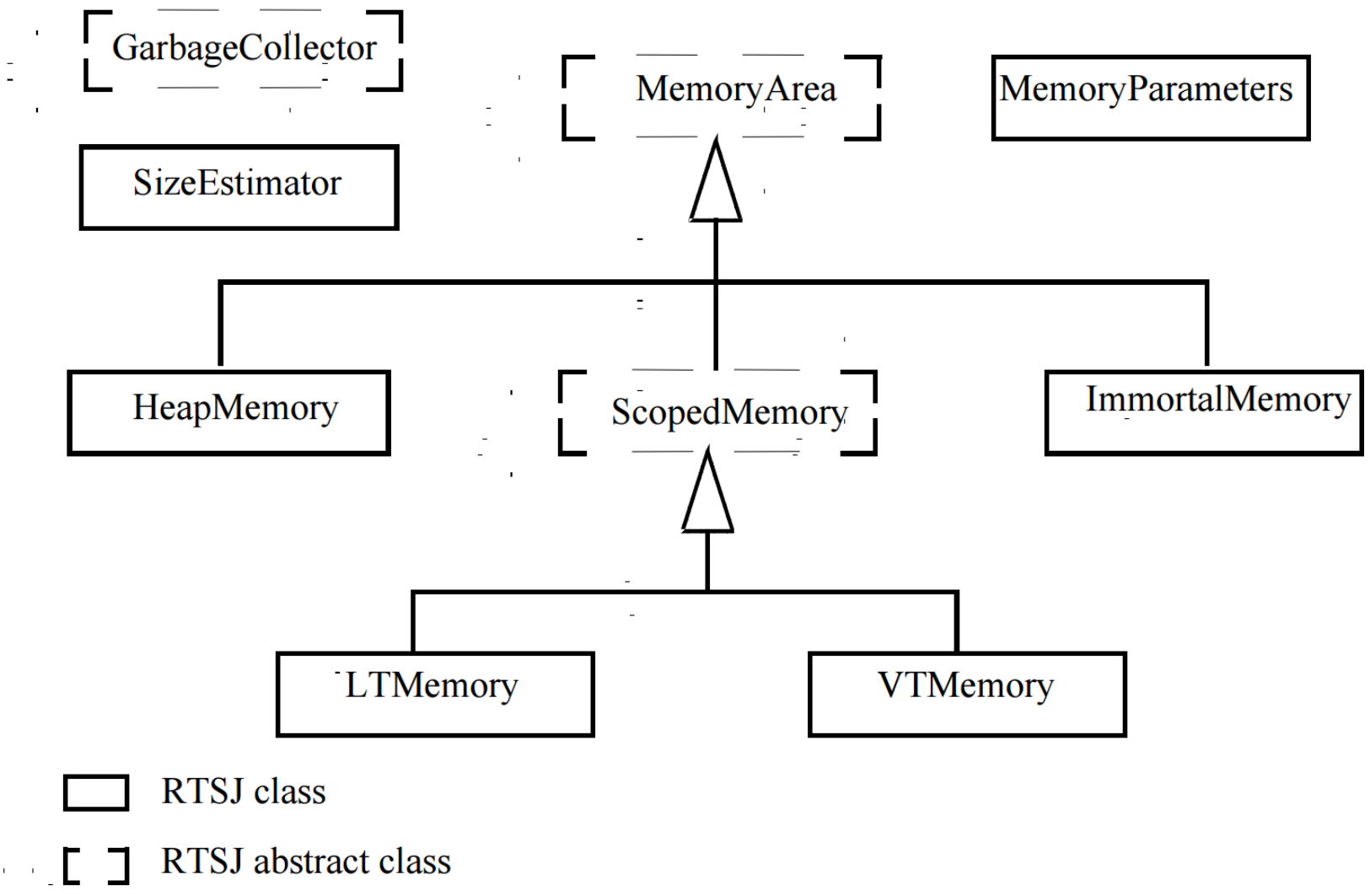
La Especificación Java-RT (RTJS)

- Cambia la JVM para que soporte RT
- No modifica el lenguaje Java; lo engloba.
- Incorpora
  - Tipos nuevos de hilos
  - Tipos nuevos de memoria
  - Gestión asíncrona de eventos
  - Predictibilidad (frente a rendimiento)
- Conservando
  - Compatibilidad hacia atrás
  - El lenguaje y el bytecode

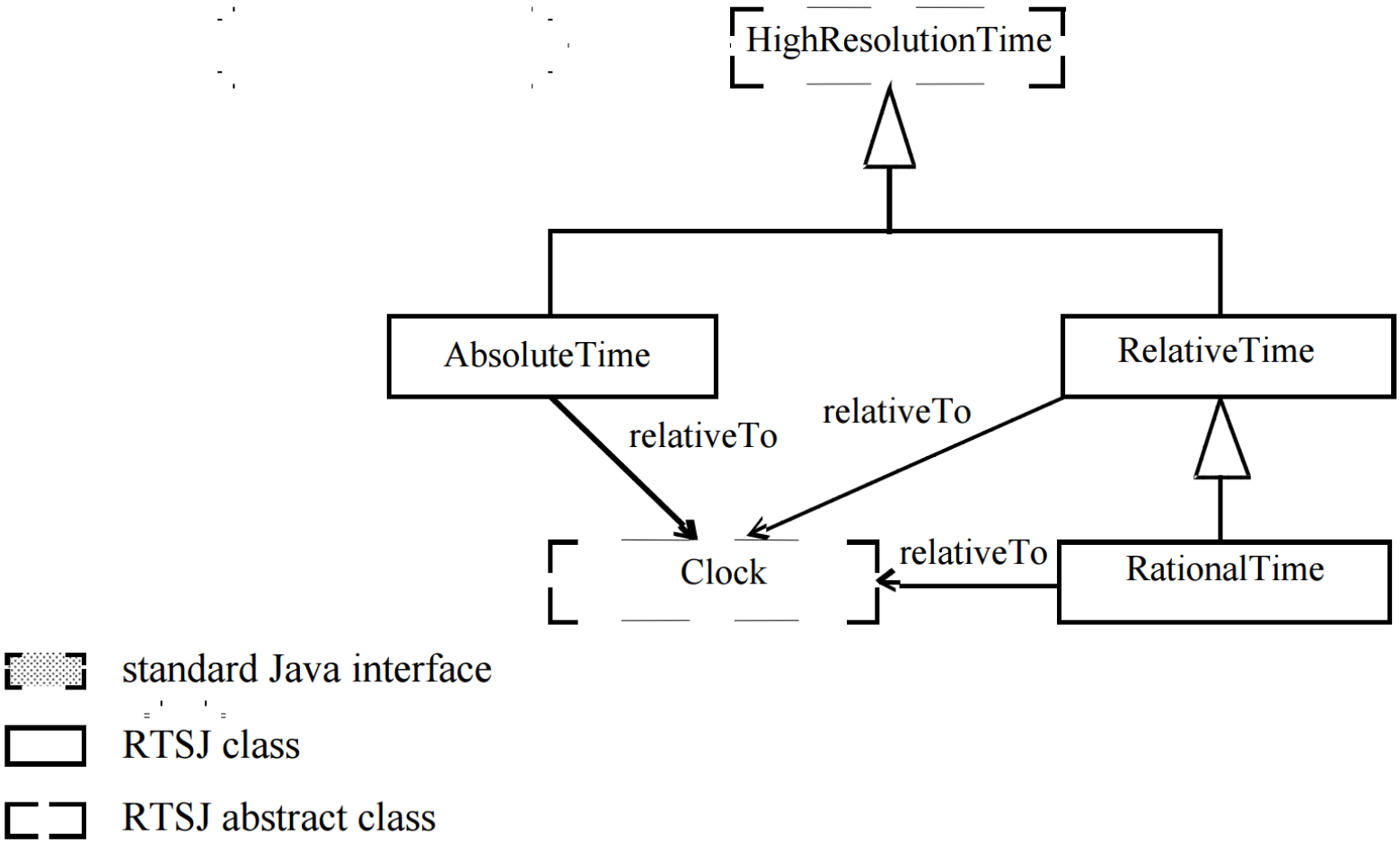
Gestión de Memoria: API

- RTJS provee áreas de memoria no afectadas por gc, al existir fuera del *heap*
- El gc puede ser expulsado por un hilo RT
- Se utiliza la clase abstracta MemoryArea y sus subclases:
  - HeapMemory
  - InmortalMemory
  - ScopedMemory
    - VTMemory
    - LTMemory

Gestión de Memoria: Clases



Relojes y Tiempo: API



Planificación en Java: Limitaciones

- Java no garantiza que los hilos de alta prioridad se ejecuten antes, debido a:
  - El *mapping* hilos java-hilos sistema
  - Uso de planificación no expulsiva
- El esquema de diez prioridades java cambia durante el *mapping* a prioridades de sistema, con superposiciones dependientes del sistema
- El concepto de planificación por prioridad es tan débil que lo hace completamente inadecuado para su uso en entornos RT
- Por tanto, los hilos de mayor prioridad, eventualmente se ejecutarán antes, pero NO hay garantías de que ello ocurra así

Planificación en Java-RT: *Gold Standard*

- Planificación por prioridades fijas, expulsiva y con 28 niveles de prioridad
- Fijas, porque los objetos planificables no cambian su prioridad (salvo si hay inversión de prioridad $\Rightarrow$ herencia de prioridad)
- Expulsiva, porque el sistema puede expeler por diferentes motivos al objeto en ejecución

Objetos Planificables (Schedulable)

- RTJS generaliza el concepto de entidades ejecutables desde el conocido *thread* a los objetos planificables (*schedulable objects*)
- Un objeto planificable implementa la interfaz `Schedulable` y puede ser
  - Hilos RT (clase `RealTimeThread` y `NoHeapRealTimeThread`)
  - Gestores de Eventos Asíncronos (clase `AsyncEventHandler`
- Cada objeto planificable debe especificar sus requerimientos temporales, indicando
  - Requerimientos de lanzamiento (*release*)
  - Requerimientos de memoria (*memory*)
  - Requerimientos de planificación (*scheduling*)

Parámetros de Lanzamiento (*release*)

- Tipos de lanzamiento
  - Periódico (activación por intervalos regulares)
  - Aperiódico (activación aleatoria)
  - Esporádico (activación irregular con un tiempo mínimo entre dos activaciones)
- Todos los tipos de lanzamiento tienen un coste y un *deadline* relativo
  - El coste es el tiempo de CPU requerido para cada activación
  - El *deadline* es el límite de tiempo dentro del cual la activación actual debe haber finalizado

Ajustes de Parámetros de Lanzamiento de un hilo-RT

```
1  RealtimeThread htr = new RealTimeThread();
2  RelativeTime miliseg = new RelativeTime(1,0);
3  ReleaseParameters relpar = new Periodic Parameters(miliseg);
4  htr.setReleaseParameters(relpar);
```

Parámetros de Planificación (*scheduling*)

- Son utilizados por el planificador para escoger qué objeto planificable ha de ejecutarse
- Se utiliza la clase abstracta `SchedulingParameters` como la raíz de una jerarquía que permite múltiples parámetros de planificación
- RTJS utiliza como criterio de planificación único la prioridad, de acuerdo al *gold standard*
- Clases:
  - PriorityParameters
  - ImportanceParameters

Ajuste de Prioridad de un hilo-RT

```
1  RealtimeThread htr = new RealTimeThread();
2  int maxPri = PriorityScheduler.instance().getMaxPriority();
3  PriorityParameters pri = new PriorityParameters(maxPri);
4  htr.setSchedulingParameters(pri);
```

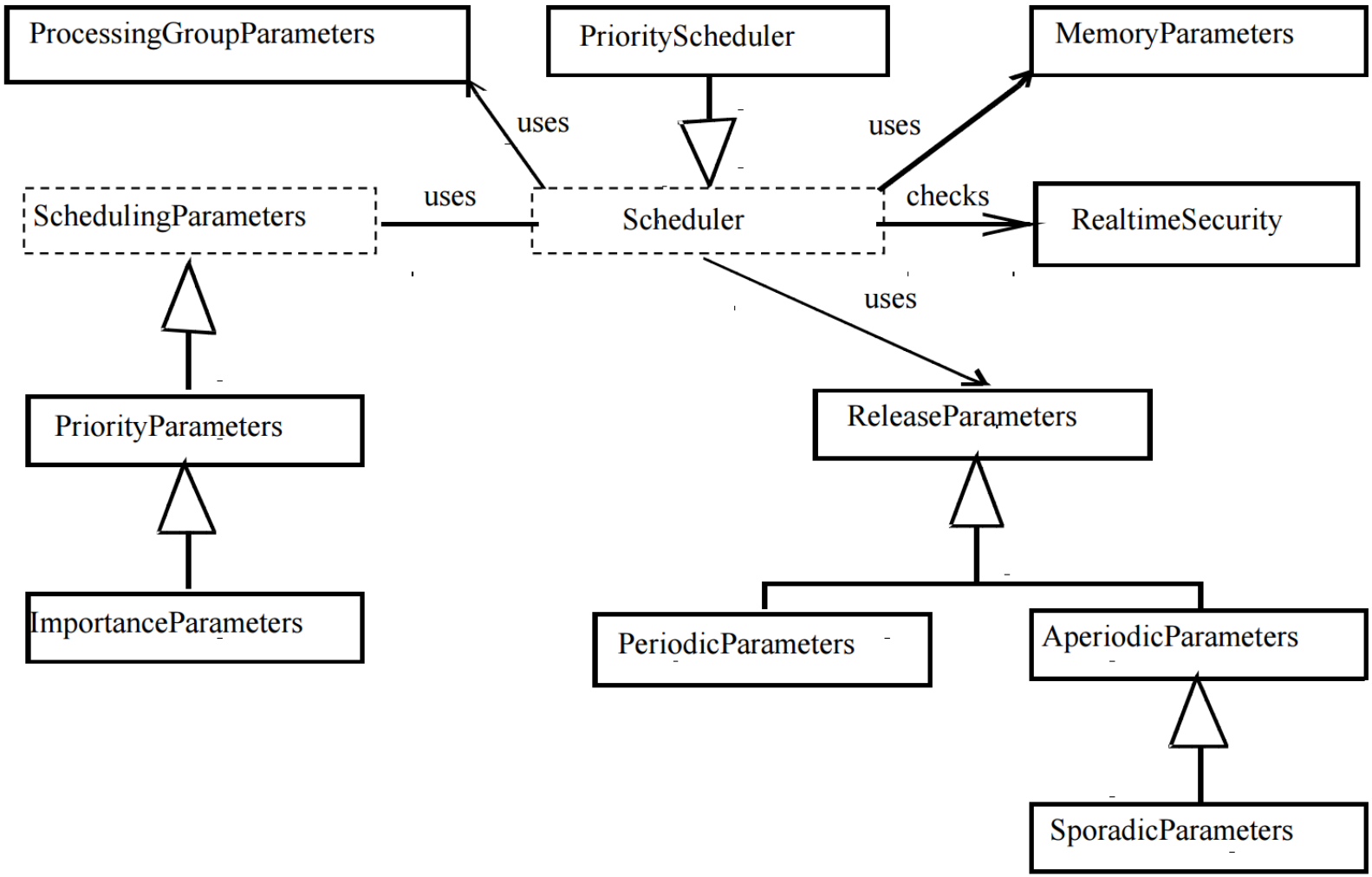
Ajuste de Prioridad Fino de un hilo-RT

```
1  RealtimeThread htr1 = new RealTimeThread();
2  int maxPri = PriorityScheduler.instance().
3  getMaxPriority();
4  ImportanceParameters ip1 = new ImportanceParameters(maxPri, 1);
5  htr.setSchedulingParameters(ip1);
6  //...
7  RealtimeThread htr2 = new RealTimeThread();
8  ImportanceParameters ip2 = new ImportanceParameters(maxPri, 2);
9  //...
10 //Se incremente la importancia de htr1 sobre htr2
11 ip1.setImportance(3);
```

Planificadores (*schedulers*)

- Son los algoritmos responsables de planificar para ejecución los objetos planificables
- RTJS soporta planificación expulsiva por prioridades de 28 niveles mediante el `PriorityScheduler`
- `Scheduler` es una clase abstracta (una instancia única por JVM y `PriorityScheduler` es una subclase
- Este esquema permite el diseño de planificadores propios mejorados

Clases para Planificación



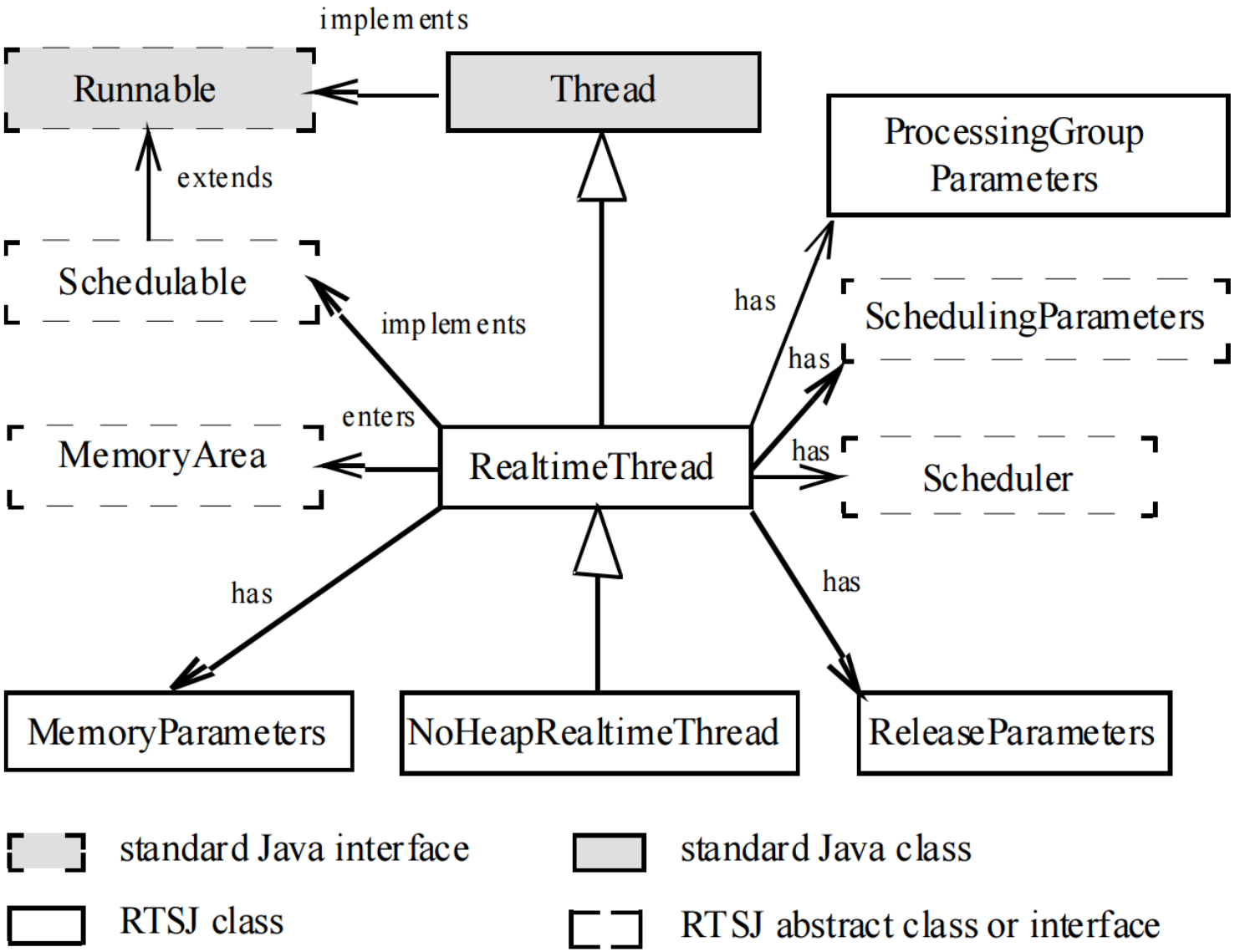
Cumplimiento de *Deadlines*

- Un sistema de tiempo real bien diseñado debe
  - Predecir si los objetos de aplicación cumplirán sus *deadlines*
  - Evitar la pérdida de *deadlines* y de sobrecargas de ejecución
- Algunos sistemas pueden ser verificados *offline*
- Otros requieren un análisis *online*
- RTJS provee herramientas para el análisis *online*

Threads Real-Time: Generalidades

- Son objetos *schedulable* y heredan de la clase `Thread`
- Son mucho más que una simple extensión de la clase
- Existe además una versión *no-heap* independiente del recolector de basura y que no usa memoria del *heap*

Threads Real-Time: Clases



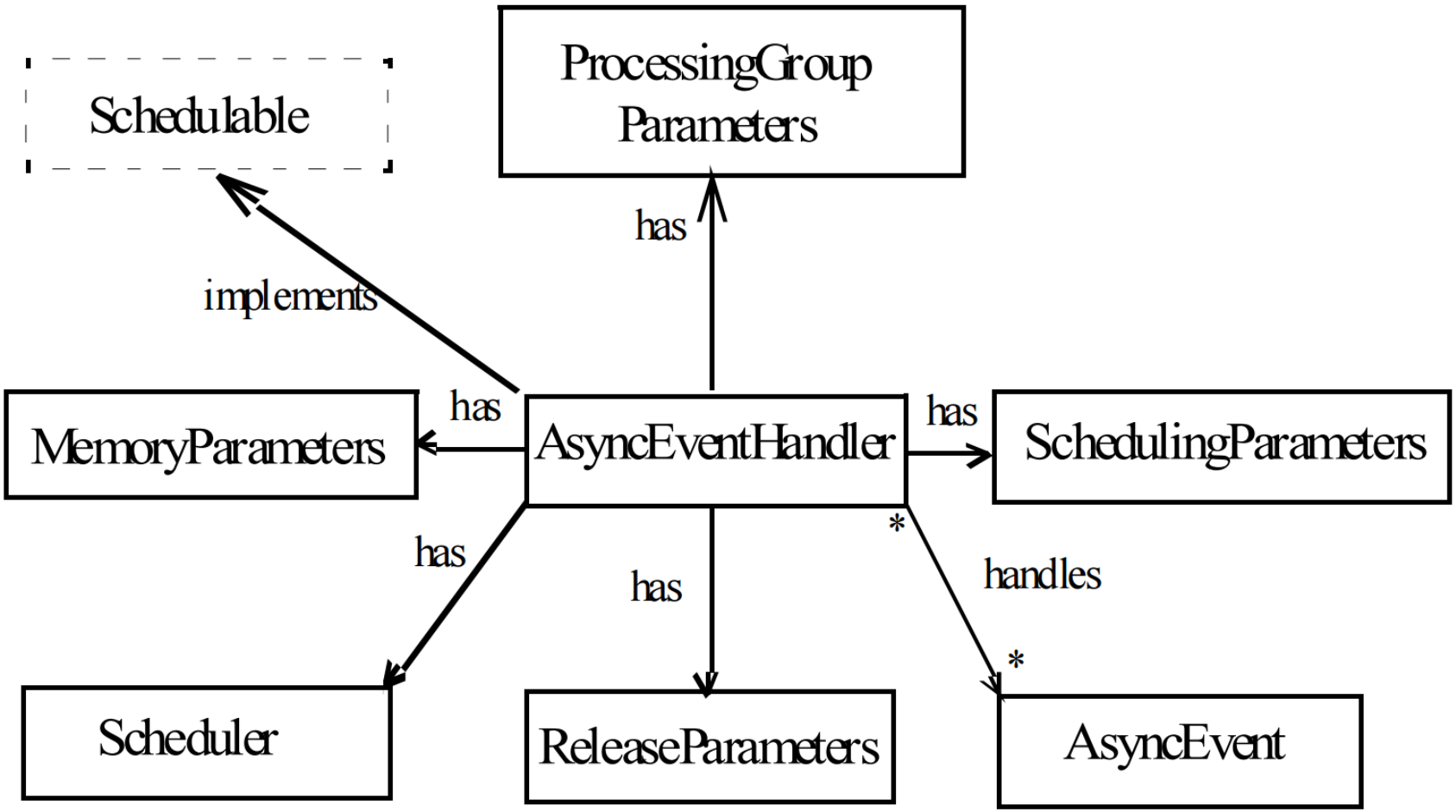
Threads Real-Time: API (Constructores)

```
1  public class RealtimeThread extends java.lang.Thread
2  implements Schedulable {
3   // constructores
4   public RealtimeThread();
5   public RealtimeThread(SchedulingParameters scheduling);
6   public RealtimeThread(SchedulingParameters scheduling,
7    ReleaseParameters release);
8   public RealtimeThread(SchedulingParameters scheduling,
9    ReleaseParameters release, MemoryParameters memory,
10   MemoryArea area, ProcessingGroupParameters group,
11   Runnable logic);
12 }
```

Gestión de Eventos Asíncronos: Generalidades

- Threads estándares o RT modelan bien tareas concurrentes con un ciclo de vida bien definido
- Se requiere modelar eventos que ocurren asíncronamente durante la actividad de un hilo
  - Procedentes del entorno
  - Generados por la lógica del programa
- Podríamos tener hilos en espera (p. e. en un *pool*) para tratarlos, pero sería ineficiente
- Desde un enfoque *real-time*, estos eventos requieren gestores que respondan bajo un *deadline*
- RTJS generaliza los gestores de eventos a objetos *schedulable*

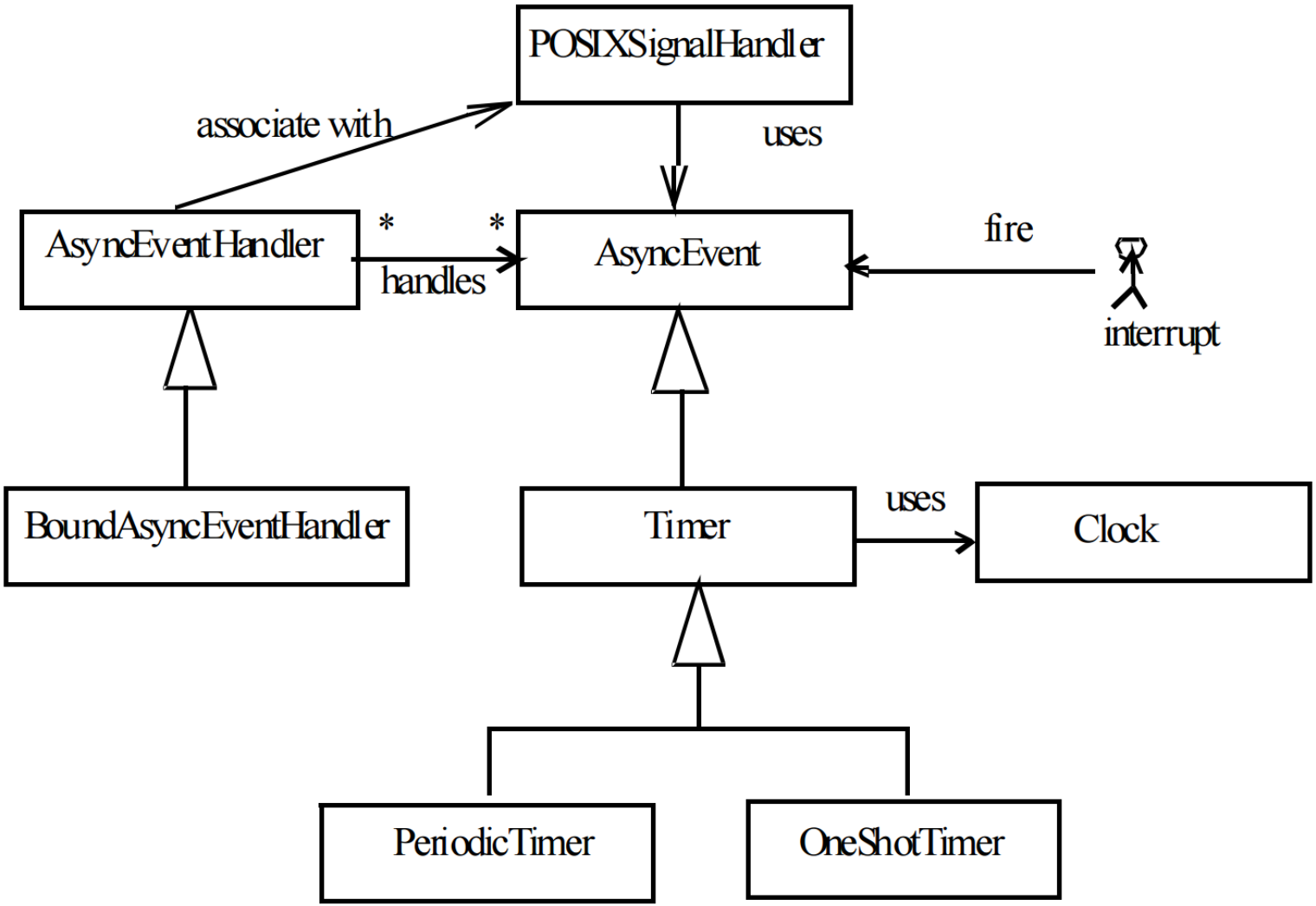
Manejadores de Eventos Asíncronos



Hilos-RT y Manejadores de Eventos

- En la práctica, la JVM-RT liga dinámicamente un gestor de eventos con un hilo-rt
- Es posible ligar un gestor de eventos a un hilo-rt de forma permanente
- Cada instancia de AsyncEvent puede tener más de un gestor de eventos
- Cuando el evento se produce, su gestor de eventos es activado para ejecución según sus parámetros de planificación

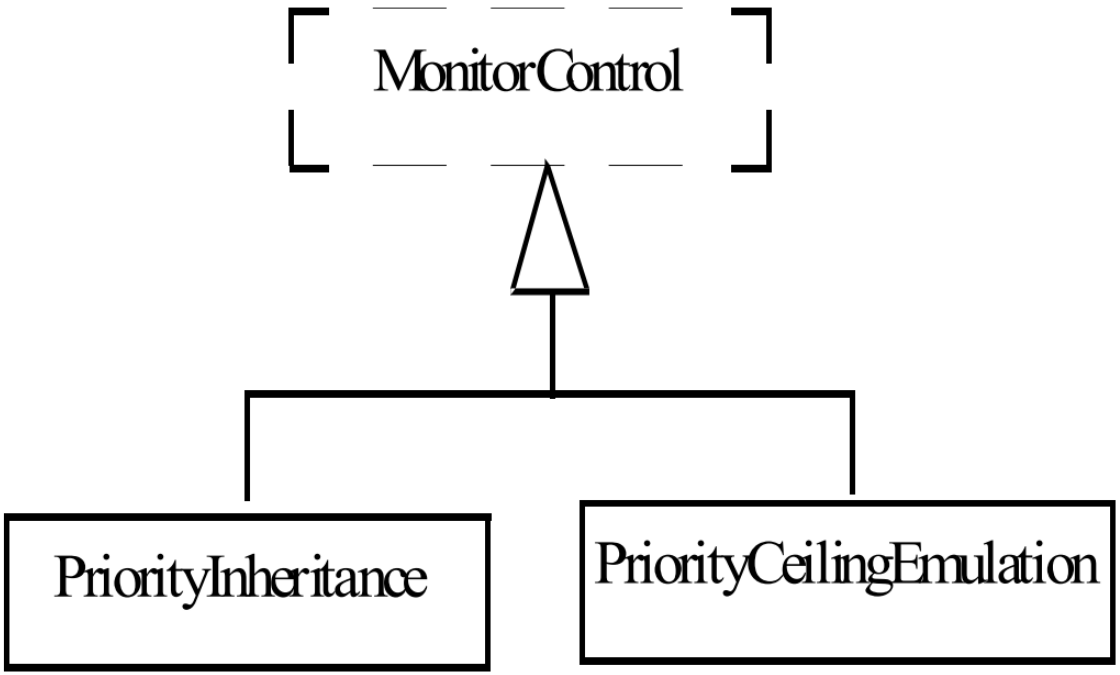
Eventos Asíncronos: Clases



Control de la Sincronización

- Los objetos planificables (incluidos hilos-rt) deben poderse comunicar y sincronizar
- Sabemos que Java proporciona control de acceso tipo monitor a objetos para control de la exclusión mutua
- Sin embargo, todas las técnicas de sincronización basadas en la exclusión mutua sufren inversión de prioridades
- Java-RT lo soluciona mediante la técnica de herencia de prioridad

Herencia de Prioridad en Java-RT: Clases



Bibliografía


Bollella, G. & Bruno, E.
Real Time Java Programming With Java RTS
SunMicrosystems, 2009


Wiley, J.
Concurrent and Real Time Programming in Java
2004