

Sistemas de Tiempo Real

La Especificación JRTS

Tema 8 - Programación Concurrente y de Tiempo Real

Antonio J. Tomeu¹

¹Departamento de Ingeniería Informática
Universidad de Cádiz

PCTR, 2019

1. Principios Generales de los Sistemas RT
2. Definición, Elementos y Tipos de un Sistema RT
3. Tipos de Tarea RT
4. Planificación de Tareas RT
5. La Especificación RTJS (Real Time Java Specification)
6. Gestión de Memoria
7. Relojes y Tiempo
8. Planificación
9. Threads RT
10. Eventos Asíncronos

Parte 1

Principios Generales de los Sistemas de Tiempo Real

Definición de Sistema de Tiempo Real

- ▶ Es un sistema cuyo funcionamiento correcto depende de los resultados lógicos producidos, y también del **instante de tiempo** en que se producen esos resultados
- ▶ El plazo del tiempo dentro del cuál los resultados deben producirse para ser considerados válidos se denomina **ligadura temporal**
- ▶ El incumplimiento de la ligadura temporal conlleva aparejado un fallo del sistema

- ▶ 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

Ejemplo de Sistema de Tiempo Real

Radar System

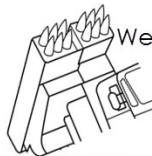


REAL-TIME SYSTEM

Incoming Missile



Command &
Decision System



Weapon Firing
System



Elementos de un Sistema de Tiempo Real

- ▶ En general, un STR incluye los siguientes elementos: sensores (antena de radar), sistema de control (hardware, software o combinación de ambos) que lee información procedente de los sensores y efectúa un cálculo con ellos (dentro de la ligadura temporal); en el ejemplo, el software de la dirección de tiro, actuadores (conjunto de dispositivos de diversa naturaleza que reacciones acciones sobre el entorno físico) y sistema controlado (sistema físico que los actuadores controlan), como es el sistema de defensa antimisiles
- ▶ El ciclo habitual de funcionamiento de un STR sigue de forma cíclica las etapas: lecturas de sensores → computación → control de actuadores → actualización del sistema controlado
- ▶ Encontramos STR en automoción, equipamiento médico, sistemas defensivos, aeronáutica, etc.

Propiedades de los Sistemas RT

- ▶ Reactividad: el sistema interacciona con su entorno, y responde correctamente dentro de su ligadura temporal a los estímulos que los sensores le proporcionan
- ▶ Predecibilidad: la ejecución del sistema es **determinista**, lo cuál implica responder en el plazo adecuado, conocer los componentes software y hardware que se utilizan, y definir marcos temporales acotados cuando no se dispone de un conocimiento temporal exacto
- ▶ Fiabilidad: grado de confianza en el sistema, dependiente de:
 - ▶ Disponibilidad: capacidad de proporcionar servicios cuando se solicitan
 - ▶ Tolerancia a fallos: capacidad de operar en situaciones excepcionales sin causar fallos
 - ▶ Fiabilidad: capacidad de responder siempre igual a la misma situación
 - ▶ Seguridad: capacidad de protegerse frente a ataques o fallos accidentales o deliberados

Tipos de Sistemas RT

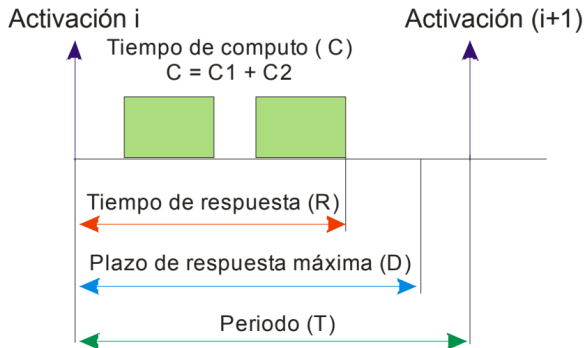
| Tipo | Características | Ejemplos |
|-------------------------------------|---|--|
| Estrictos (hard) | Respuesta dentro de la ligadura temporal | Control de vuelo de un avión sistema ABS de frenado |
| No estrictos Flexibles (soft) | Cumplir la ligadura temporal es el objetivo, pero el sistema puede funcionar bien aunque haya respuestas que no se den en plazo | Sistema de adquisición de datos meteorológicos |
| Firmes | Se permiten algunos incumplimientos de la ligadura temporal, pero si son excesivos el sistema falla de forma total | Sistema de control de reserva de vuelos sistema de video bajo demanda |

Un sistema de tiempo real se estructura en tareas que acceden a los recursos del sistema

- ▶ Tarea: Conjunto de acciones que describe el comportamiento del sistema o parte ejecutando código secuencial (proceso o hebra)
 - ▶ Las tareas satisfacen necesidades funcionales concretas
 - ▶ Las tareas tienen definida una ligadura temporal mediante atributos temporales
- ▶ Recurso: Elementos disponibles para la ejecución de tareas
 - ▶ Activos: Procesador, cores, red
 - ▶ Pasivos: Datos, memoria, dispositivos E/S

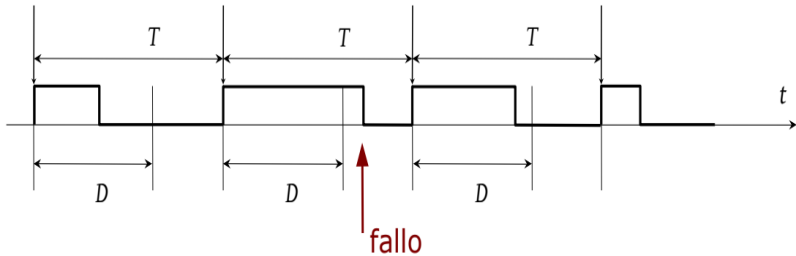
- ▶ Tiempo de ejecución (C): tiempo necesario para ejecutar la tarea íntegramente
- ▶ Tiempo de respuesta (R): tiempo que la tarea ha necesitado para ejecutarse íntegramente
- ▶ Ligadura temporal (*deadline*) (D): máximo tiempo de respuesta posible
- ▶ Periodo (T): intervalo temporal entre activaciones sucesivas de tareas periódicas

Atributos Temporales de Tareas RT

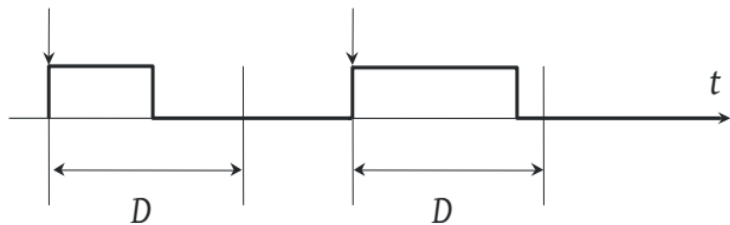


Tareas RT Periódicas

Periódicas: T es el periodo de activación de la tarea

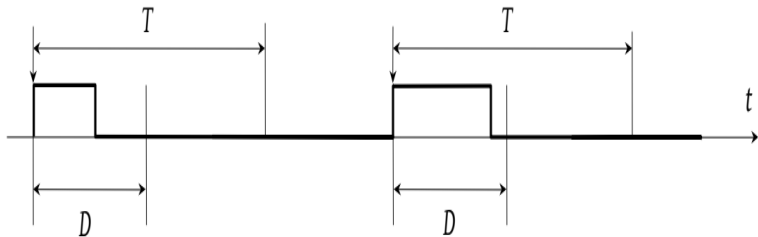


Tareas RT Aperiódicas



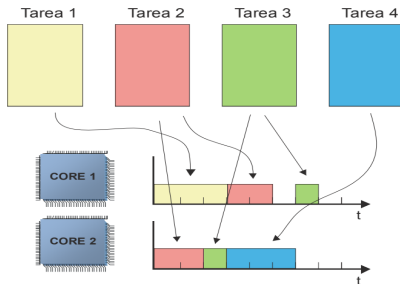
Tareas RT Esporádicas

Esporádicas: T es la separación mínima entre eventos



Planificación de Tareas RT

La planificación de tareas se encarga de asignar las tareas a los recursos activos de un sistema RT (en especial el procesador o los cores de ejecución), garantizando la ejecución de todas las tareas de acuerdo a las restricciones establecidas. En sistemas RT, estas restricciones aparecen principalmente en forma de ligaduras temporales



¿Y qué se necesita para planificar tareas?

- ▶ Determinar los procesadores (o cores) disponibles a los que podemos asociar tareas
- ▶ Determinar las relaciones de dependencia entre tareas:
 - ▶ Relaciones de precedencia entre tareas
 - ▶ Determinar los recursos comunes a que distintas tareas acceden
- ▶ Determinar el orden de ejecución de las tareas para garantizar el cumplimiento de las restricciones

Esquema de Planificación de Tareas RT

Determinar la planificabilidad de un conjunto de tareas requiere de un esquema (método) de planificación que incluya los siguientes elementos:

- ▶ Un algoritmo de planificación, que define una política de planificación para determinar el orden de acceso de las tareas a los procesadores
- ▶ Un método de análisis (test de planificabilidad) para predecir el comportamiento temporal del sistema, y de termina si la planificabilidad es posible bajo las condiciones o restricciones presentes
- ▶ Una comprobación factible de que las ligaduras temporales se cumplen en todos los casos posibles
- ▶ En general, se estudia el peor caso posible o WCET (Worst Case Execution Time)

Construir un test de planificabilidad para un conjunto de tareas exige conocer el WECT (C) para cada una de ellas. Es posible determinar este parámetro de dos formas diferentes:

- ▶ Medimos el tiempo de ejecución en el peor de los casos sobre la plataforma hardware en varias ocasiones y se construye una estadística
- ▶ Se efectúa un análisis del código ejecutable
 - ▶ descomponiendo el código en un grafo de bloques secuenciales
 - ▶ calculando el tiempo de ejecución de cada bloque
 - ▶ y buscando el camino más largo en el grafo

Esquemas de Planificación (sistema monoprocesador)

- ▶ Planificación estática *off line* sin prioridades
 - ▶ Planificación cíclica (ejecutivo cíclico)
- ▶ Planificación basada en prioridades
 - ▶ Prioridades estáticas
 - ▶ Prioridad al más frecuente (RMS, Rate Monotonic Scheduling)
 - ▶ Prioridad al más urgente (DMS, Deadline Monotonic Scheduling)
 - ▶ Prioridades dinámicas
 - ▶ Proximidad del plazo de respuesta (EDF, Earliest Deadline First)
 - ▶ Prioridad al de menor holgura (LLF, Least Laxity First)

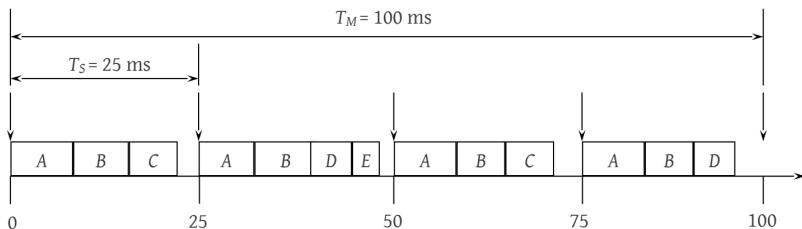
Ejemplo de Planificación Cíclica

- ▶ Construye una planificación explícita y fuera de línea de las tareas que especifica el entrelazado de las mismas de forma que su ejecución cíclica garantiza que se cumplen las ligaduras temporales
- ▶ La estructura de control se denomina ejecutivo cíclico
- ▶ El entrelazado de tareas es fijo, y conocido antes de poner en marcha el sistema
- ▶ La duración del ciclo principal es igual al hiperperiodo T_M
 - ▶ $T_M = mcm(T_1, T_2, \dots, T_m)$
 - ▶ se supone tiempo entero
 - ▶ el comportamiento temporal del sistema se repite cada ciclo principal

Planificación Cíclica

| Tarea | T | C |
|-------|-----|----|
| A | 25 | 10 |
| B | 25 | 8 |
| C | 50 | 5 |
| D | 50 | 4 |
| E | 100 | 2 |

- ▶ El ciclo principal dura $T_M = 100ms$
- ▶ Se compone de 4 ciclos secundarios de 25 ms



- ▶ No hay concurrencia en la ejecución
 - ▶ Cada ciclo secundario en una secuencia de llamadas a método
 - ▶ No se necesita un núcleo de ejecución multitarea
- ▶ Las tareas pueden compartir datos (y no hace falta control de exclusión mutua)
- ▶ No hace falta analizar el comportamiento temporal

Inconvenientes de la Planificación Cíclica

- ▶ Dificultad para incorporar tareas con periodos largos
- ▶ Las tareas esporádicas dificultan el tratamiento
- ▶ El plan del ciclo es difícil de construir
- ▶ Es poco flexible y de difícil mantenimiento

Planificación con Prioridades

- ▶ La prioridad es un atributo de las tareas que mide su importancia relativa respecto a las demás
- ▶ Se codifica con números enteros crecientes según la importancia de la tarea (o al revés)
- ▶ La prioridad de una tarea determina sus necesidades temporales
- ▶ Idealmente, las tareas se despachan para ejecución según su prioridad

- ▶ Método de planificación estático on line con asignación de prioridades a las tareas más frecuentes
- ▶ Cada tarea recibe una prioridad única basada en su periodo; cuanto menor sea el periodo (mayor frecuencia), mayor prioridad $\forall i, j : T_i < T_j \rightarrow P_i < P_j$
- ▶ Aquí asignamos prioridades decrecientes para procesos más frecuentes

- ▶ Da mayor prioridad a la tarea más próxima a su ligadura temporal (deadline). En caso de igualdad, se hace una elección no determinista
- ▶ Es un algoritmo de planificación dinámico, donde las prioridades de las tareas cambian en el tiempo
- ▶ No necesita que las tareas sean periódicas

Parte 2

Java Real Time Specification (JRTS)

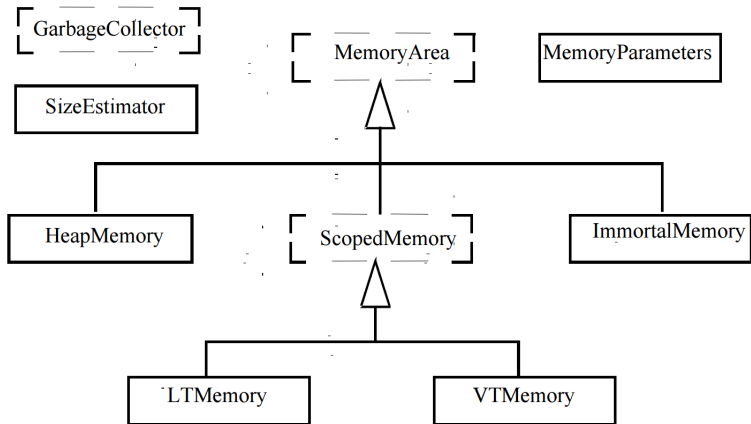
- ▶ Gestión de Memoria
 - ▶ gc es impredecible. Se activa cuando quiere
 - ▶ Fragmenta la memoria (memoria no planificable)
- ▶ Planificación de Threads
 - ▶ Impredecible
 - ▶ No permite especificar cuándo un hilo ha de ejecutarse
 - ▶ Inversiones de Prioridad
- ▶ Sincronización
 - ▶ Duración de secciones 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

- ▶ 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

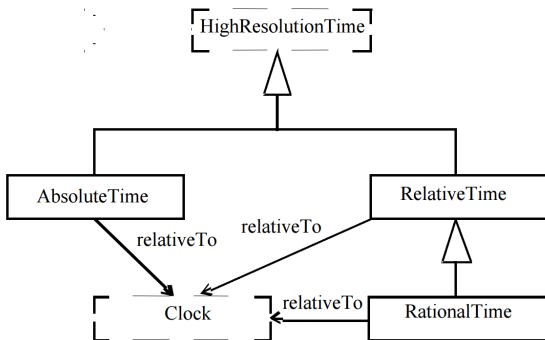



RTSJ class




RTSJ abstract class

Relojes y Tiempo: API



 standard Java interface

 RTSJ class

 RTSJ abstract class

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 por prioridades fijas, **preemptive (expulsivo)** 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);
```

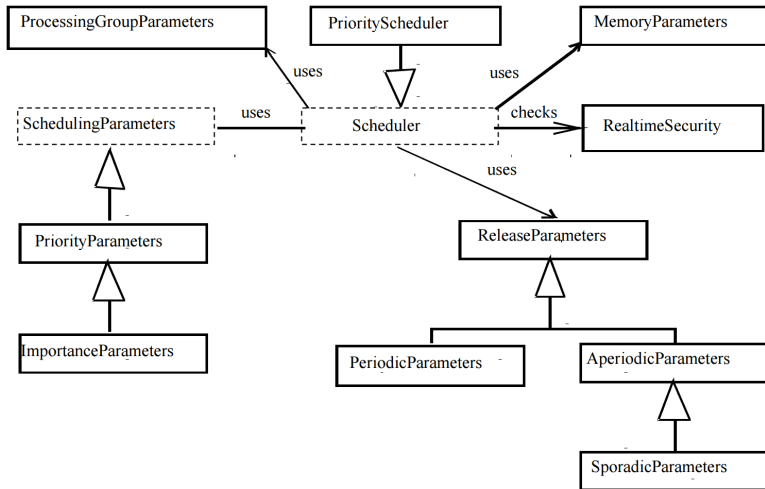

Ajuste de Prioridad Fino de un hilo-RT

```
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

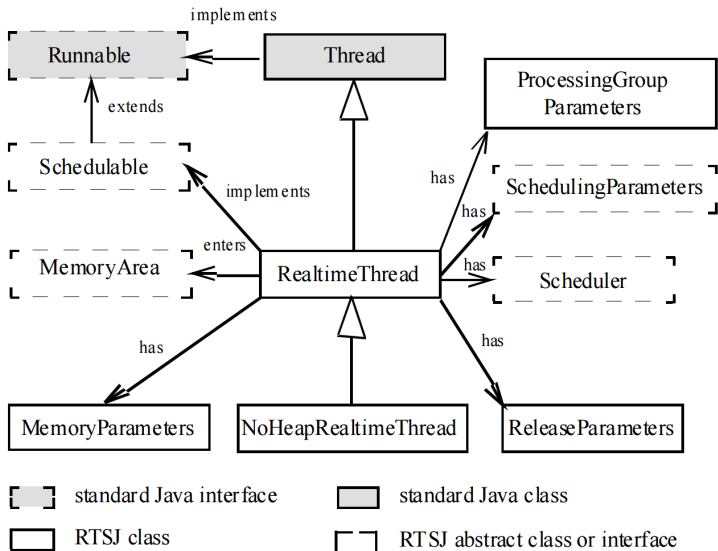


Cumplimiento de *Deadlines*

- ▶ 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*

- ▶ 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: Classes



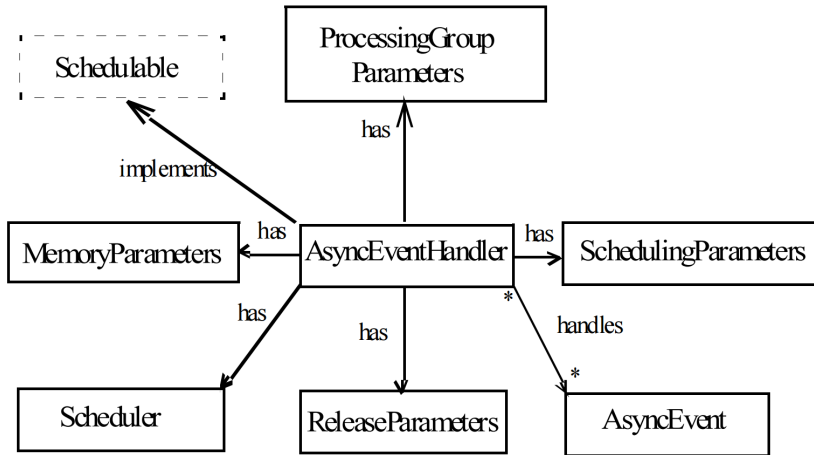
Threads Real-Time: API (Constructores)

```
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

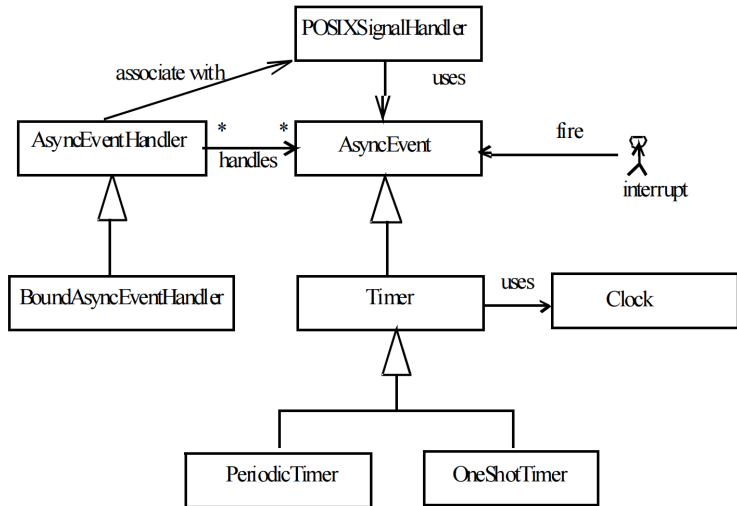
- ▶ Threads estándares o RT modelan bien tareas concurrentes con un ciclo de vida bien definido
- ▶ Se requiere modelar eventos que ocurren asíncronamente durante la actividad de un hilo
 - ▶ Procedentes del entorno
 - ▶ Generados por la lógica del programa
- ▶ Podríamos tener hilos en espera (p. e. 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



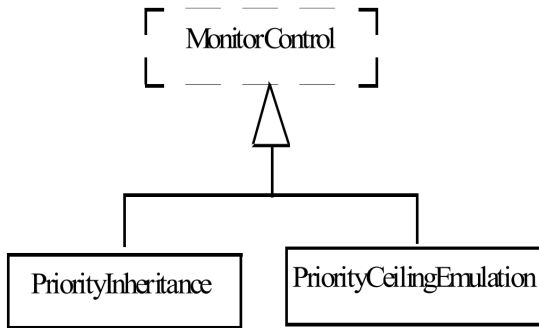
- ▶ En la práctica, la JVM-RT liga dinámicamente un gestor de eventos con un hilo-rt
- ▶ Es posible ligar un gestor de eventos a un hilo-rt de forma permanente
- ▶ Cada instancia de `AsyncEvent` puede tener más de un gestor de eventos
- ▶ Cuando el evento se produce, su gestor de eventos es activado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




Eventos Asíncronos: Clases



- ▶ 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



-  Bollella, G. & Bruno, E.
Real Time Java Programming With Java RTS
Sun Microsystems, 2009
-  Capel, M. & Rodríguez, S.
Sistemas Concurrente y Distribuidos
Editorial Copicentro, 2012
-  Wiley, J.
Concurrent and Real Time Programming in Java
2004