Seminarvortrag

The Real-Time Specification for Java

Erik Burger

Overview

- History of RTSJ / development process
- Standard JDK architecture
- Requirements on real-time systems
- RTSJ specification
- Code examples
- Implementations

History of RTSJ

History of RTSJ

- Java is generally believed as unsuitable for real-time applications
- Common handling of real-time components in Java: Use C for real-time parts, Java for the application

History of RTSJ

- Java Specification Request (JSR) number 1
- Specification Document: 2001
- various companies were involved during the spec process (Sun, IBM, TimeSys)

Design of Java

Design of Java

Write
Once
Run
Anywhere

Virtual Machine (JVM)

- abstraction layer for underlying hardware
- bytecode interpreter = hand-optimized assembler code
- Just-In-Time (JIT) compiler
- does class loading, scheduling, memory management

Garbage Collection (GC)

- not in the Java Specification
- part of the JVM
- collects unused memory
- GC is the only possibility to free memory in Java

Design of Java

GC is executed

- on request: `System.gc` suggests GC, but program can prevent execution
- on demand: only when `new()` is invoked and there is not enough memory on the heap
- in background: when system thinks that program is idle; can be disabled

Requirements on a Real-Time System

Requirements on a RT System

real time ≠ really fast

Requirements on a RT System

- But then, what does Real-Time mean?
- „*the ability to reliably and predictably reason about and control the temporal behavior of program logic*” (Bollela)
- consistent performance over speed

Requirements on a RT system

according to Alan Burns, a real-time system is required to:

- specify times at which actions are to be performed
- specify times at which actions are to be completed
- respond to situations where all timing requirements cannot be met
- respond to situations where the timing requirements are changed dynamically

Requirements on a RT system

How does Java meet these requirements?

- GC makes predictions on execution times complicated
- possible solution: force GC to become predictable (e.g. Sun Java RTS 1.0)
- JVM performance depends on caching, paging, DMA
- problem: uncontrolled asynchronous events

The Real-Time Specification for Java

Specification

- enhancement of API, no syntactical extension
- any compiler can make classes that use RTSJ
- correct non-real-time code will execute on any real-time implementation
- for real-time applications, a real-time JVM is needed

Specification

The RTSJ enhances Java in six fields:

- RT threads with more carefully defined scheduling attributes
- tools and mechanisms that help writing code that does not need GC
- asynchronous event handlers
- asynchronous transfer of control
- control over memory allocation of objects
- direct memory access

High-Resolution Time

- 0.1µs granularity is sufficient for almost any software purpose
- RTSJ: 84bit time
- nanosecond granularity,
  range of ~292 years
- **`HighResolutionTime`** Base Class
- Rational Time for periodic events

Real-Time Threads

- extended priorities (28 levels)
- scoped memory
- `AsynchronouslyInterruptedExceptions`
- periodic scheduling

Memory Allocation

- no “magic” real-time GC
- three types of memory:
  - immortal memory: never GC'ed, shared among all threads, remains until JVM terminates
  - scoped memory: defined lifetime
  - heap: GC frees object only if it cannot be reached

Specification

- **`NoHeapRealTimeThread`**
  - is not allowed to access elements on the heap
  - can preempt GC
- Reuse of immortal memory is encouraged
- re-using of threads is possible

Specification

- Allowed memory assignment:

	➔ Heap	➔ Immortal	➔ Scoped
Heap	✓	✓	✘
Immortal	✓	✓	✘
Scoped	✓	✓	”Available” Scopes
Local Variable	✓	✓	”Available” Scopes

Asynchronous Events

- events in Java – *happenings* in RTSJ
- **`AsynchronousEventHandler`**

Specification

Asynchronous Transfer of Control

- deprecated Java methods:
  ~~`Thread.suspend()`~~, ~~`Thread.resume()`~~, ~~`Thread.stop()`~~
  `throws ThreadDeathException`
- but TDE is catchable -> possible corruption of data structures

Raw Memory Access

- in standard Java, no memory adresses can be accessed (security)
- thread may not touch memory other than its own
- but: control of memory-mapped devices requires raw memory access
- **`class RawMemoryAccess`**

Specification

Write

O nce

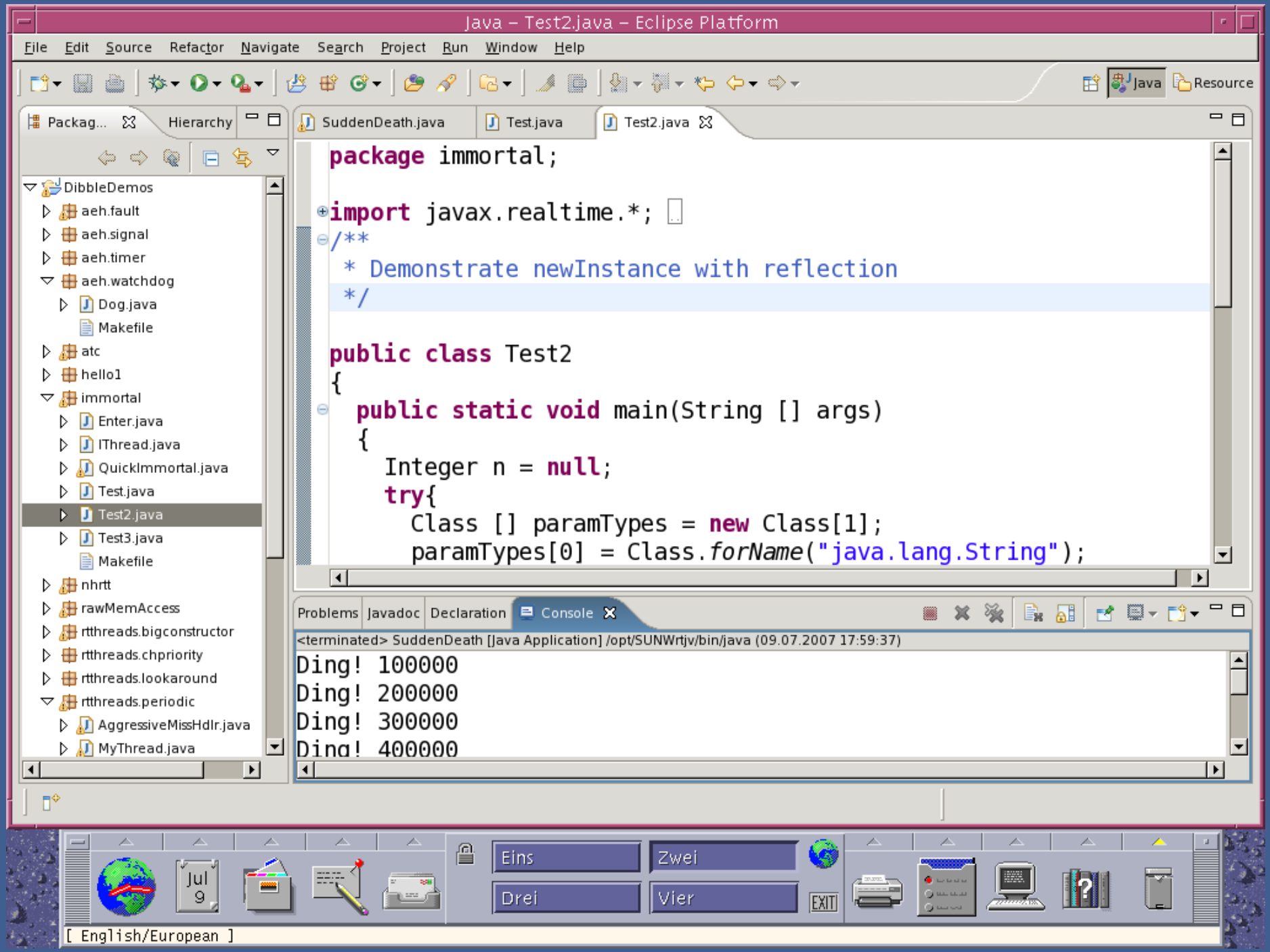
C arefully

R un

A nywhere

C onditionally

Code Example


Java – Test2.java – Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Resource
Packag...
Hierarchy
SuddenDeath.java
Test.java
Test2.java
DibbleDemos
aeh.fault
aeh.signal
aeh.timer
aeh.watchdog
Dog.java
Makefile
atc
hello1
immortal
Enter.java
IThread.java
QuickImmortal.java
Test.java
Test2.java
Test3.java
Makefile
nhrtt
rawMemAccess
rtthreads.bigconstructor
rtthreads.chpriority
rtthreads.lookaround
rtthreads.periodic
AggressiveMissHdlr.java
MyThread.java
package immortal;
import javax.realtime.*;
/**
* Demonstrate newInstance with reflection
*/
public class Test2
{
public static void main(String [] args)
{
Integer n = null;
try{
Class [] paramTypes = new Class[1];
paramTypes[0] = Class.forName("java.lang.String");
Problems
Javadoc
Declaration
Console
<terminated> SuddenDeath [Java Application] /opt/SUNWrtjv/bin/java (09.07.2007 17:59:37)
Ding! 100000
Ding! 200000
Ding! 300000
Ding! 400000
Eins
Zwei
Drei
Vier
EXIT
Jul
9
[English/European]

SuddenDeath.java

```
public static class OverrunHandler extends AsyncEventHandler  {
  RealtimeThread th;

  public void setThread(RealtimeThread th) {
    this.th = th;
  }

  OverrunHandler() {
    super(
    new PriorityParameters(
        PriorityScheduler.instance().getMinPriority() + AEH_PRIORITY),
        null, null, null, null, null);
  }

  public void handleAsyncEvent() {
    System.out.println(
    "Zapping thread that's over budget");
    th.interrupt(); // Throw an AIE into the thread
  }
}
```

SuddenDeath.java

```
/**
 * Define a RT thread that stops quickly in case
 * of an async interrupt.
 */
public static class PeriodicThread  extends RealtimeThread  {
 private void doWork() throws
  AsynchronouslyInterruptedException {
    int bound=0;

    while(true){
      do{
        // Use some time
        for(f=0.0;f<bound;f+=1.0) ;
        bound += 100000; // Use more next time
        System.out.println("Ding! " + bound);
      }while(waitForNextPeriod()) ;
      // Recover from miss
      System.out.println("Scheduling error");
      bound -= 150000; // Lighten load
      while(!waitForNextPeriod()) // Eat errors
        System.out.print(".");
      System.out.println();
    }
  }
```

SuddenDeath.java

```
public static void main(String [] args) {

  // Build parameters for construction
  // of RT thread
  OverrunHandler overrunHandler = new OverrunHandler();
  ReleaseParameters release = new PeriodicParameters(
      new RelativeTime(), // Start at .start()
      new RelativeTime(1000, 0), // 1 second period
      null, // cost
      new RelativeTime(100,0),// deadline=period/10
      overrunHandler, // overrun handler
      null); // no miss handler

  SchedulingParameters scheduling = new PriorityParameters(
      PriorityScheduler.instance().getMinPriority() +
      THREAD_PRIORITY);
```

SuddenDeath.java

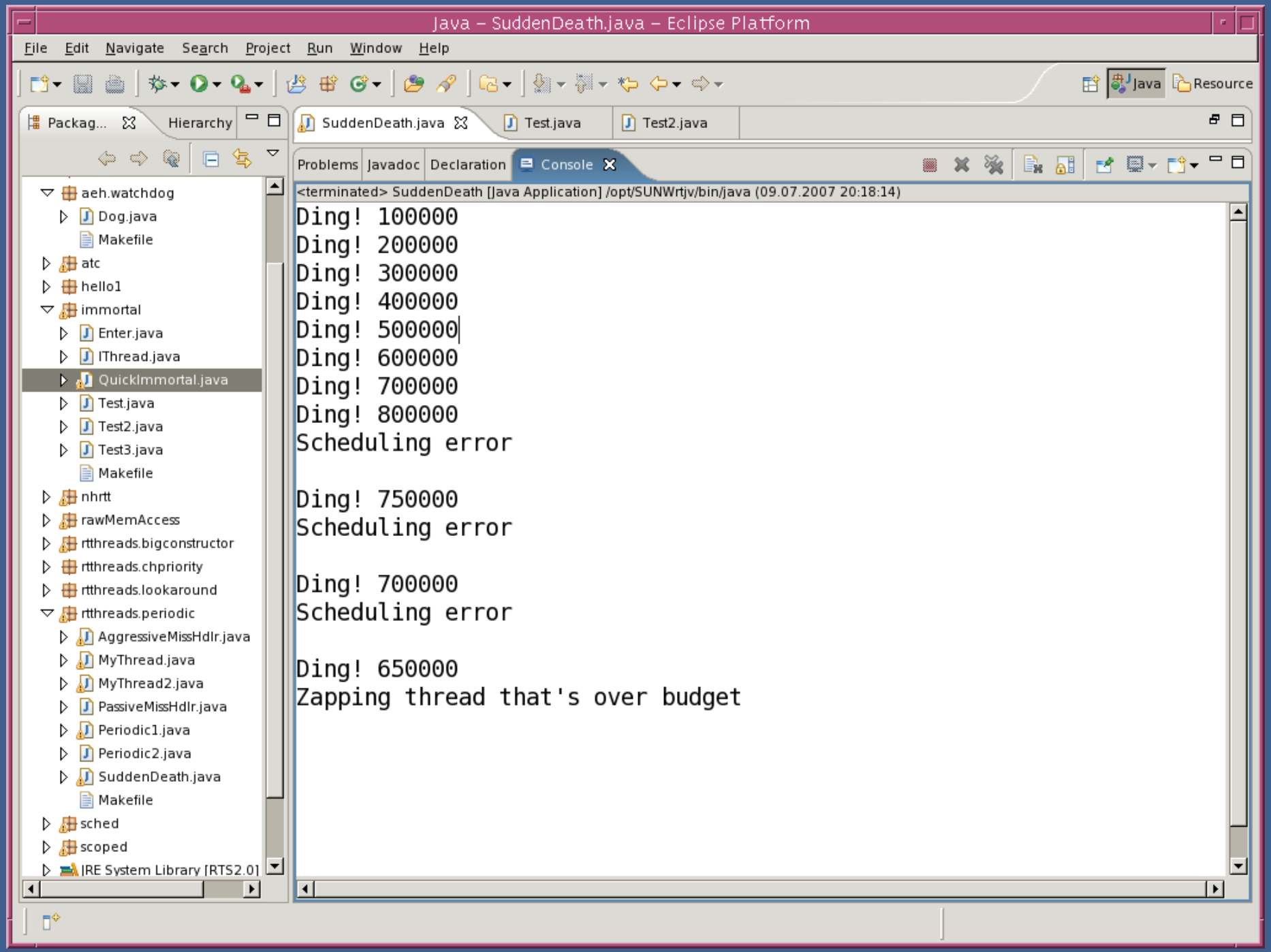
```
    RealtimeThread rt= new PeriodicThread(scheduling, release);

    // Give the overrun handler a reference to
    // the thread it is managing.
    overrunHandler.setThread(rt);

    rt.start(); // Start the periodic thread
    try{
      Thread.sleep(10000);
    }catch(InterruptedException e){}

    AsyncEvent evt = new AsyncEvent();
    evt.addHandler(overrunHandler);
    evt.fire(); // pretend we caught an overrun

    try{
      rt.join(); // Wait for the thread to end
    }catch(Exception e){};
  }
}
```


Java – SuddenDeath.java – Eclipse Platform
File Edit Navigate Search Project Run Window Help
Java Resource
Packag... Hierarchy
SuddenDeath.java Test.java Test2.java
Problems Javadoc Declaration Console
<terminated> SuddenDeath [Java Application] /opt/SUNWrtjv/bin/java (09.07.2007 20:18:14)
Ding! 100000
Ding! 200000
Ding! 300000
Ding! 400000
Ding! 500000
Ding! 600000
Ding! 700000
Ding! 800000
Scheduling error
Ding! 750000
Scheduling error
Ding! 700000
Scheduling error
Ding! 650000
Zapping thread that's over budget
aeh.watchdog
Dog.java
Makefile
atc
hello1
immortal
Enter.java
IThread.java
QuickImmortal.java
Test.java
Test2.java
Test3.java
Makefile
nhrtt
rawMemAccess
rtthreads.bigconstructor
rtthreads.chpriority
rtthreads.lookaround
rtthreads.periodic
AggressiveMissHdlr.java
MyThread.java
MyThread2.java
PassiveMissHdlr.java
Periodic1.java
Periodic2.java
SuddenDeath.java
Makefile
sched
scoped
JRE System Library [RTS2.0]

Implementations

Timesys

- RTSJ reference implementation
- x86-Linux (Fedora)
- freely available
- version 1.1alpha

Sun Java Real Time System (RTS)

- Solaris (x86 and SPARC)
- version 2.0 (May 2007)
- available for academic use

Implementations

jRate

- extension of the Gnu GCJ compiler
- Washington University, St. Louis
- GPL
- version 0.3.7.2

Javolution

- real-time library
- BSD licensed
- for J2ME, J2SE, GCJ

References

References

- www.rtsj.org
- **[Bol00] Greg Bollela** et al: *The Real-time Specification for Java*, Addison-Wesley, 2000
- **[Dib02] Peter Dibble**: *Real-Time Java Platform Programming*, Sun Microsystems Press, 2002
- **[Bur01] Alan Burns**: *Real-Time Systems and Programming Languages*, Addison-Wesley, 2001

Summary

Summary

- RTSJ enhances Java with real-time capabilities
- offers platform-independent real-time software development
- keeps compatibility with legacy Java applications
- still under development
- various implementations