IN4R21
Real-Time Specification for Java (RTSJ)

Damien MASSON
http://www.esiee.fr/~massond/

January 20, 2014
References

- www.rtsj.org

- Concurrent and Real-Time Programming in Java, Andy Wellings

- Real-Time Java Programming: With Java RTS, Eric J. Bruno et Greg Bollella
Thread

- Permits concurrent execution of several sequential codes to execute within *the same process* (so share memory)
- method main = new thread, 1 process: the VM
- the programmer can start new threads, modeled by the Thread class.

```java
Thread t = new Thread();
t.start();
```

- execute method run() in a new Thread
- the method run() must be overridden

```java
Thread t = new Thread(){
    @Override public void run(){
        // hello
    }
}
t.start();
```
Runnable

But direct redefinition of `run()` in `Thread` is a *bad* idea for a lot of *good* reasons:

- no multiple heritage with Java,
- field exists in `Thread`, the name space is not empty (e.g. `Thread.name` exists),
- no semantic separation between logic to execute and shell that model it for parallelism/concurrency.

So there is the `Runnable` interface:

```java
Runnable logic = new Runnable()
    {
        @Override public void run()
        {
            // hello
        }
    }
Thread t = new Thread(logic);
t.start();
Thread t2 = new Thread(logic);
t2.start();
```
method setPriority();
Threads Gesture

- `suspend / stop() / destroy() / resume()`: deprecated (must not be use)
- `interrupt()`
- `yield()`
- `sleep()`
Timer

Permits to program executions
java.util.concurrent

Contains a lot of tools to easily handle concurrency inherent issues, i.e. without explicitly manipulate Thread: lesser chances to make mistakes.
Synchronisations

- permit to control/prevent the concurrent access to a piece of code (so to memory or other resources)
- rendez-vous

The more easy (but not recommended!)

```java
public class MyClass{
    public synchronized void myMethod(){
        // code to protect
    }
}
```

is equivalent to:

```java
public void myMethod(){
    synchronized(this){
        // code to protect
    }
}
```

(replace this by MyClass.getClass() for static methods)
Synchronizations

The good way:

```java
public class MyClass {
    private final Object lock = new Object();
    public void myMethod() {
        synchronized (lock) {
            // code to protect
        }
    }
}
```

- any Object can serve as lock
- the block structure prohibit to keep a lock beyond the method scope
- java 1.5 add the Lock object in order to address this limitation (does not concern this class)
Synchronizations

This is not magic !! Warning:

```java
public class MyClass{
    private final Object lock = new Object();
    public void myMethod(){
        // code
    }
    public void myOtherMethod(){
        synchronized(lock){
            myMethod();
        }
    }
    public void mySecondOtherMethod(){
        myMethod();
    }
}
```

"Synchronized" is only a wait to acquire a lock, it does permit to block an other piece of code to execute if this other piece of code does not ask the same lock.
rendez-vous

- `Object.wait()`
- `Object.wait(timeout)`
- `Object.notify()`
- `Object.notifyAll();`
RTSJ philosophy

- No restriction to a specific version of Java
- RTSJ VM should be able to run any Java application
- Based on J2SE
- No syntax extension
- Does not require specific hardware
- Deterministic timing behavior
Scope:
- Thread scheduling
- Memory Gesture
- Synchronization and resources sharing
- Asynchronous event handling
- Asynchronous transfer of control
- Timing and Clock
- System Functions
www.rtsj.org
one package: javax.realtime
one spec for an rt-jvm
one reference implementation: RI (tjvm)
many other implementations: Java RTS, JamaïcaVM ...
Scheduling

- (I) Runnable
  - (I) Schedulable
    - (C) AsyncEventHandler
      - (C) BoundAsyncEventHandler
    - (C) RealtimeThread
      - (C) NoHeapRealtimeThread
  - (C) Thread
Scheduling
Runnable logic = new Runnable()
    {
        @Override public void run()
        {
            System.out.println("real-time hello");
        }
    }
RealtimeThread t = new RealtimeThread(logic);
t.start();

- Priority assignment with setSchedulingParameters() (pretty much the same as setPriority(), but warning: the spec is the spec, implementations are implementations...), or directly to the constructor

new RealtimeThread(new PriorityParameters(15));

- Priority segmentation between non RT (from 1 to 10) and RT (from 11 to at least 30)
Runnable logic = new Runnable(){
    @Override public void run(){
        do{
            System.out.println("periodic hello in real-time");
        } while(waitForNextPeriod());
    }
}
RelativeTime start = new RelativeTime();
RelativeTime period = new RelativeTime(1000,0);
PeriodicParameters param = new PeriodicParameters(start, period);
RealtimeThread t = new RealtimeThread(param, logic);
t.start();

waitForNextPeriod returns false when the deadline is missed (implicit deadline is the default)
Absolute/Relative Time

- Date/duration: two subclasses of HighResolutionTime
- Permits to encode precise time (but does not mean this is the system granularity !!)
- Warning: most of these object’s methods create objects

```java
private final AbsoluteTime abs = new AbsoluteTime();
private final RelativeTime rel1 = new RelativeTime();
private final RelativeTime rel2 = new RelativeTime();
void myMethod()
{
    Clock.getRealtimeClock().getTime(abs);
    rel1.set(10, 5);
    abs.subtract(rel1, rel2);
}
```

```java
private final RelativeTime rel1 = new RelativeTime();
private final RelativeTime rel2 = new RelativeTime();
void methode()
{
    AbsoluteTime abs = Clock.getRealtimeClock().getTime();
    RelativeTime rel1 = new RelativeTime(10, 5);
    RelativeTime rel2 = abs.subtract(rel1);
}
```
Runnable logic = new Runnable(){
    @Override public void run()
    {
        do{
            System.out.println("periodic hello in real-time");
        }while(waitForNextPeriod());
    }
}
RelativeTime start = new RelativeTime();
RelativeTime period = new RelativeTime(1000,0);
RelativeTime deadline = new RelativeTime(500,0);
RelativeTime wcet = new RelativeTime(200,0);
PeriodicParameters param = new PeriodicParameters(start, period, wcet, deadline, null, null);
RealtimeThread t = new RealtimeThread(param, logic);
t.start();
Runnable logic = new Runnable()
    {
        @Override
        public void run()
        {
            System.out.println("Sporadic Hello");
        }
    }

AsyncEventHandler handler = new AsyncEventHandler(logic);
AsyncEvent event = new AsyncEvent();
event.addHandler(handler);
// ...
event.fire();
Runnable logic = new Runnable()
    {
        @Override public void run()
        {
            System.out.println("Hey");
        }
    }
AsyncEventHandler handler = new AsyncEventHandler(logic);
AsyncEvent event = new AsyncEvent();
event.addHandler(handler);
AsyncEventHandler handler2 = new AsyncEventHandler(logic);
event.addHandler(handler2);
// ...
event.fire();
Runnable logic = new Runnable() {
    @Override public void run() {
        System.out.println("Periodic Hello");
    }
};
AsyncEventHandler handler = new AsyncEventHandler(logic);
RelativeTime start = new RelativeTime();
RelativeTime period = new RelativeTime(1000, 0);
PeriodicTimer timer = new PeriodicTimer(start, period, handler);

AsyncEventHandler encapsulates Release/Scheduling Parameters, as RealtimeThread
Runnable logic = new Runnable()
{
    @Override public void run()
    {
        do{
            System.out.println("periodic\nhello");
        }while(waitForNextPeriod());
    }
}

RelativeTime start = new RelativeTime();
RelativeTime period = new RelativeTime(1000,0);
RelativeTime deadline = new RelativeTime(500,0);
RelativeTime wcet = new RelativeTime(200,0);
AsyncEventHandler wcetHand = new AsyncEventHandler(/\*/...*/\*);
AsyncEventHandler deadHand = new AsyncEventHandler(/\*/...*/\*);
PeriodicParameters param = new PeriodicParameters(start, period, wcet, deadline, wcetHand, deadHand);
RealtimeThread t = new RealtimeThread(param, logic);
t.start();

- If the VM is able to detect faults, it call fire() method of encapsulated Event
- Beware of handlers priorities...
Delegated to the Scheduler

- addToFeasibility()
- isFeasible()
- bad idea!
Asynchronous transfer of control (interruption)

- Brings the possibility to stop an other thread
- Only if this thread executes a piece of code that has been explicitly marked as interruptable
- Use Exception
  - AsynchronouslyInterruptedException
- Timed
- interruptible
- interrupt();