Java™ & Real-Time
Advantages, Limitations, & RTSJ

Christos Kloukinas

Christos.Kloukinas@imag.fr

Verimag

(http://www-verimag.imag.fr/)

Grenoble, France

Introduction

Java is a language with lots of merits
We will examine:

- Its advantages for Real-Time programming;
- Its disadvantages for Real-Time programming; and,
- The RTSJ proposal.

What's Java?

Java is:

- Object-Oriented;
- Type-Safe;
- Portable;
- Library-Rich; and,
- has automatic Garbage Collection.

Java versus C/C++

- O-O helps with structuring code and decreasing code dependencies.
- Increased type-safety, bound-checks and lack of pointer-arithmetic increase overall safety and allow for aggressive compiler optimisations.
- Built-in exceptions, threads & synchronisation means that the compiler knows how to do "*The Right Thing*".
- Built-in GC means no more dangling pointers & a lot fewer memory leaks.
- Fixed size and byte order of basic types helps with porting your software.

From C to Java

- Structures now become Classes and have associated Methods.
- Easier to tell what functions can have access to a structure.
- No need to check the kind of structure another function passed you - the O-O internal mechanisms do this automatically.

O-O: How to let your compiler write your switches. . .

From C to Java - II

Controlling access to methods, fields & classes themselves:

- **`public`**
- **`private`**
- **`protected`**
- **`package`**s (only one **`class`** can be **`public`** in a **`package`**)

Other modifiers:

- **`abstract`** / **`static`** / **`final`** / `strictfp`
- **`synchronized`** (equivalent to a **`synchronized`**`(`**`this`**`)`)

From C to Java - III

- **unions** & **enums** don't exist - use polymorphism (or compile time constants if it's not worth the abstraction)
- multiple inheritance - use interfaces or composition
- Reflection - `getClass`, `getMethods`, `getPackage`, `isInstance`

From C to Java - III

- **unions** & **enums** don't exist - use polymorphism (or compile time constants if it's not worth the abstraction)
- multiple inheritance - use interfaces or composition
- Reflection - `getClass`, `getMethods`, `getPackage`, `isInstance`
- *Unlike C++, there are no object destructors!*
  Cannot control when `finalizer`s will be executed (if they ever are. . .)

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Concurrent Programming with Java

Java has:

- Threads
- Monitors (& Condition Variables)
- Timers
- JNI

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Threads in Java

- Threads implement the `Runnable` interface, *i.e.*, provide a `run` method
- Threads have priorities: from `MIN_PRIORITY` to `MAX_PRIORITY`, default is `NORM_PRIORITY`
- Threads have lower priority than the GC (*always*)
- Resource allocation is not (necessarily) priority-based
- Stack size can be declared, but it can also be ignored. . .

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Threads in Java - II

- Dæmon threads (cannot keep an application running)
- Have methods: `holdsLock`, `interrupt`, `join`, `yield`, `sleep`, `destroy` (*not implemented*)
- *No longer have*: `resume`, `stop`, `suspend`

Interrupts:

- Threads poll for interrupts with one of: `wait`, `sleep`, `join`, `isInterrupted`
- The target thread can poll whenever it wants

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Starting a Thread

```
class ERR {
  static public void pr(String msg){
    System.err.println(Thread.
   currentThread().toString()+":␣"+msg)
   ;}}
class MyThread extends Thread {
  Thread parent;
  public MyThread() {parent = Thread.
  currentThread();}
  public void run() {
    ERR.pr("Hello");
  }
}
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Starting a Thread - II


```
class Main {
 public static void main(String[] argv)
  {
   MyThread aThread = new MyThread();
   ERR.pr("this␣is␣main");
   aThread.start(); // calls run()
   try {aThread.join();}
   catch (InterruptedException ie){ERR.pr
   ("Un-joinable"+ie);}
   finally {ERR.pr("Bye");}
  }
}
```


Interrupting a Thread

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

```
class MyThread extends Thread {
  // As before ...
  public void run() {
    ERR.pr("Hello");
    try {this.sleep(2);} catch (
    InterruptedException ie) {}
    parent.interrupt();
  }
}
```


Interrupting a Thread - II

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

```
class Main {
 public static void main(String[] argv)
 throws InterruptedException {
   MyThread aThread = new MyThread(Thread
   .currentThread());
   ERR.pr("this␣is␣main");
   aThread.start();     // calls run()
   try {
     aThread.join();
```


Interrupting a Thread - III

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

```
      if (Thread.currentThread().
      isInterrupted()) {
        ERR.pr("Was␣interrupted");
        throw new InterruptedException("
        Suicide...");}
    } catch (InterruptedException ie) {
      ERR.pr("Un-joinable:" + ie);
      throw ie;
    } finally {ERR.pr("Bye");}
    ERR.pr("Never␣printed...");
  }
}
```


Monitors

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Each Java object has its own semaphore and its own condition variable.
- We use the **`synchronized`**`(object){}` construct to denote a critical region block protected by a monitor on the semaphore of the object.
- Inside a monitor, we use `wait()` & `notify()` to wait on a condition & to flag it.
  Can also use `wait(/*timeout*/)` and `notifyAll()`.
- A **`break`**, **`continue`**, **`return`**, **`throw`** inside a monitor will exit it *after* having unlocked the object.
  Are you sure you do so in C/C++?

Monitors - II

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Entering a monitor is *not* priority-based.
- A timed wait can wait for more than the timeout specified.
- Choosing a thread to notify is *not* priority-based.
  No priority-sorted wait queues.
- *No priority-inversion avoidance protocol.*

Timers

- `java.lang.Timer` & `java.util.TimerTask`
  - One-time execution;
  - Repeated execution.
    - fixed-delay
      n^{th} execution starts *period* milliseconds after the $(n-1)^{th}$ execution
      (delays are therefore additive)
    - fixed-rate
      n^{th} execution starts $n*$*period* milliseconds after the *first* execution
      (try to keep a constant rate - may execute many times in rapid succession to catch up)

Timers - II

- No Real-Time guarantees! – Using `wait(/*timeout*/)`
- If a timer task takes too long, it can delay future executions.
- Not a dæmon thread by default
  (can keep an application from terminating)
- When should I run if no delays?
  `scheduledExecutionTime()`
- A timer can be associated with many timer tasks
- A timer task can be associated with many timers
- `cancel()` a timer or a timer task

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Timer Example

```
class MyTimerTask extends TimerTask {
 private int MAX_TARDINESS = 15;
 public void run() {
  ERR.pr(toString()+":␣Started");
  if (System.currentTimeMillis() -
  scheduledExecutionTime() >=
  MAX_TARDINESS)
    return; // Too late
  ERR.pr(toString()+":␣running␣@␣" +
  System.currentTimeMillis());
 }
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Timer Example - II

```
 public static void main(String[] argv)
  {
   MyTimerTask aTimerTask = new
   MyTimerTask();
   Timer    aTimer   = new Timer(false);
   // Not a daemon timer
   // Start=10 ms, period=30 ms
   aTimer.scheduleAtFixedRate(aTimerTask
   , 10L, 20L);
   // aTimer.cancel();
  }
}
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Native Interface Example - Java

```
class HelloWorld {
 private native void print();
 static {
  System.loadLibrary("HelloWorld");
 }
 public static void main(String[] args)
  {
  Thread.currentThread().setName("The␣
  Main␣from␣Java");
  new HelloWorld().print();
 }
}
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Native Interface

- Declaring methods as **`native`**
- Loading them up at start-up using **`static`** blocks
- Finding out signatures using `javap -s` *`class`*

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Type Signatures

void	V		
boolean	Z	**char**	C
byte	B	**short**	S
int	I	**long**	J
float	F	**double**	D

Object of class X :
`L`X`;`
Array of type X :
`[`X
ret `foo(`*args*`)` :
`(`*args*`)`*ret*

`String[] foo(Thread[] arr, `**`int`**` i, `**`long`**` j):`

?

Type Signatures

void	V		
boolean	Z	**char**	C
byte	B	**short**	S
int	I	**long**	J
float	F	**double**	D

Object of class X :
`L`X`;`
Array of type X :
`[`X
ret `foo(`*args*`)` :
`(`*args*`)`*ret*

`String[] foo(Thread[] arr, `**`int`**` i, `**`long`**` j):`

`([Ljava/lang/Thread;IJ)[Ljava/lang/String;`

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

JNI

- Can do lock/unlock, **`new`**, *etc.*
- It's heavy - all queries use strings
- Cannot optimise code written with it
- Java threads are not necessarily system threads
- May have to use a `finalizer`
  It will be run before *the memory is reused*...

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

CNI

- Used with GCJ (front-end of GCC)
- Java classes are C++ classes (not *vice versa*)
- Faster & can be optimised
- A lot simpler to write code

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Disadvantages

- Java is slow;
- Java's memory model is convoluted (Under revision);
- The GC is unpredictable;
- Scheduling is unpredictable;
- Monitors can suffer from priority-inversion;
- Notification is not priority-based;
- Timers are not accurate;
- Design for portability
  $\Rightarrow$ Cannot use any special OS mechanisms

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Speed

- It's not Java, it's the implementations;
- One can use a JIT or an AOT compiler;
- It's easier to optimise Java than C/C++
  No (wild) casts, pointer-arithmetic, etc..

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

JIT or AOT?

- JIT:
  - can do anything a JVM does;
  - works on bytecodes;
  - needs resources;
  - cannot optimise much;
  - is unpredictable.
- AOT:
  - can do aggressive optimisations;
  - no need for extra resources at the target system;
  - is predictable;
  - can always add a JVM for loading classes.

Real-Time GC

- Interruptible, low latency, Real-Time GC's exist
- Usually meant for large or non-critical systems
- But there is also the memory region technique.

RTSJ

- Thread Scheduling & Dispatching
- Memory Management
- Physical Memory Access
- Synchronisation & Resource Sharing
- Asynchronous Event Handling
- Asynchronous Transfer of Control
- Asynchronous Thread Termination

Scheduling

- Preemptive, Priority-based by default
- At least 28 different priorities
- `RealtimeThread`
  - Can access the heap
  - Lower priority than the GC
- `NoHeapRealtimeThread`
  - No heap accesses
  - Higher priority than the GC
- `AsyncEventHandler`
  - Depends on the `noheap` arg
  - Can be unbound or bound to a particular thread
  - It can block (holding resources)

Scheduling - II

- Need to have a reference to a `Scheduler` obj.
- A `Scheduler` has params for priority, importance, memory, *etc.*
- `PeriodicParameters`, `AperiodicParameters`, `SporadicParameters`
- Processing group to treat non-periodic threads as a single periodic one
- Max. Execution time, Deadline & overrun handlers
  The execution overrun handler may be ignored & use the duration only as a hint to the feasibility algorithm.

Scheduling - III

- `Scheduler.addToFeasibility(Schedulable)` to add another task in the feasibility analysis performed by the scheduler.
- `Scheduler.isFeasible()` to find out if the current system can be indeed scheduled
- Changing the parameters of a task at run-time through the
```
setIfFeasible(Schedulable,
              ReleaseParameters,
              MemoryParameters)
```
- `waitForNextPeriod()` for periodic tasks.

Scedulable Objects

```
setScheduler(
Scheduler          sched,
SchedulingParameters sched_params,
ReleaseParameters   release_params,
MemoryParameters    mem_params,
ProcessingGroupParameters group)
```

RTSJ defines the `PriorityScheduler`, with two **`final int`** fields, `MAX_PRIORITY` and `MIN_PRIORITY`.

Scheduling Parameters

RTSJ defines two kinds of Scheduling parameters:

- `PriorityParameters(`**`int`**` priority)`
- and an additional one for declaring the importance of the threads along with their priority
  `ImportanceParameters(`**`int`**` priority, `**`int`**` importance)`

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Release Parameters - Periodic

```
PeriodicParameters(
HighResolutionTime start,  /* null =
start when released */
RelativeTime  period,
RelativeTime  cost,
RelativeTime  deadline, /* use period
when null */
AsyncEventHandler overrunHandler,
AsyncEventHandler missHandler)
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Release Parameters - Aperiodic

```
AperiodicParameters(
RelativeTime  cost,
RelativeTime  deadline,
AsyncEventHandler overrunHandler,
AsyncEventHandler missHandler)
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Release Parameters - Sporadic

```
SporadicParameters(
RelativeTime  minInterarrival,
RelativeTime  cost,
RelativeTime  deadline,
AsyncEventHandler overrunHandler,
AsyncEventHandler missHandler)
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Memory Parameters

```
MemoryParameters(
long maxMemoryArea,
 // or MemoryParameters.NO_MAX (units=
 bytes)
long maxImmortal,
 // or NO_MAX (units=bytes)
long allocationRate)
 // or NO_MAX (units=bytes/sec)
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Thread Group

```
ProcessingGroupParameters(
HighResolutionTime start,
RelativeTime   period,
RelativeTime   cost,
RelativeTime   deadline,
AsyncEventHandler overrunHandler,
AsyncEventHandler missHandler)
```


Creating a R-T Thread

```
Scheduler sched= new PriorityScheduler();
PeriodicParameters pp = new
PeriodicParameters(
  new RelativeTime(0,0),//when released
  new RelativeTime(100, 10), // period
  new RelativeTime(30, 0), // cost
  new RelativeTime(60, 0), // deadline
  null,         // no Overrun Handler
  null);     // no Miss Period Handler
ImportanceParameters prio = new
  ImportanceParameters(MAX_PRIORITY, 3);
MemoryArea ma = new LTMemory(1024, 1024);
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Creating a R-T Thread - II

```
MemoryParameters mp = new
  MemoryParameters(MemoryParameters.
 NO_MAX,MemoryParameters.NO_MAX);
ProcessingGroupParameters gp = null;
RealtimeThread rt = new
  RealtimeThread(prio, pp, mp, ma, gp,
    new Runnable () {/*the run method...*/
    });
rt.setScheduler(sched);
if (!rt.getScheduler().isFeasible())
  throw new Exception("Not␣feasible");
rt.start();   // Release the thread
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Finding a New Scheduler

```
class SchedExample {
public static Scheduler findSched(String
policy) {
 String className = System.getProperty("
 javax.realtime.scheduler."+policy);
 Class clazz;
 try {
  if (null != className && null != (clazz
   = Class.forName(className)))
   return (Scheduler) clazz.getMethod("
   instance",null).invoke(null,null);
 } catch (/* lots of exceptions */) {}
 return null;
}
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Finding a New Scheduler - II

```
Scheduler sched = findSched("EDF");
if (null != sched) {
 RealtimeThread rt = new RealtimeThread(
 /*...*/);
 rt.setScheduler(sched);
 if (!rt.getScheduler().isFeasible())
   throw new Exception("Not␣feasible");
 rt.start();   // Release the thread
}
```

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Memory Management

- Execution Stack, Heap & *Memory Scopes*
- Scopes give bounds to the lifetime of objects allocated within them
- Objects of a scope collected *en masse* after its `Runnable` exits
- A scope is collected before being reused *(not when exited, so `finalizer`s still a problem)*
- Hierarchy of scopes - inner, outer (or unrelated)
- *Immortal memory* : top-most scope, collected when the JVM exits
- Objects surviving current scope need be allocated to the top-most scope which contains a reference to these objects

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

Memory Management - II

- `LTMemory` *versus* `VTMemory`
  - Allocation in `LTMemory` is *linear* with respect to object's size.
  - A `VTMemory` may be separately GC'ed and may be augmented if it's not big enough
- Assignment Rules

	Refs Heap	Immortal	Scoped
Heap	Yes	Yes	No
Immortal	Yes	Yes	No
Scoped	Yes	Yes	Yes, if same or higher scope

Memory Management - III

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- `LTPhysicalMemory`, `VTPhysicalMemory`, `ImmortalPhysicalMemory`
- `SizeEstimator` & `reserve(`**`class`**`, instances)`
- `RawMemoryAccess` - Accessing a region interpreted as **`byte`**, **`short`**, **`int`**, **`long`** or as an array of such
- `RawMemoryFloatAccess` - Same but for **`float`** & **`double`**
- Maximum preemption latency of a Real-Time thread due to the GC is given by: `RealtimeSystem.currentGC().getPreemptionLatency()` (returns a `RelativeTime`)

Synchronisation

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Priority inheritance protocol supported by default
- Given the GC it's impossible to implement PIP!
- 3 Wait-free queues for avoiding this.
  - `WaitFreeWriteQueue` (non-blocking write), `WaitFreeReadQueue` (non-blocking read), `WaitFreeDequeue` (non-blocking write & read)
- Real-Time part of queue never blocks - no need for PIP.
- If action is not possible (*e.g.*, queue is full) you decide (overwrite old value, ignore the new one)

Asynchronous Events

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- `AsyncEvent`, `AsyncEventHandler`, `handleAsyncEvent()` ≡ `run()`
- `bindTo(String)`, `addHandler()`, `fire()`
- *Data-less* events
- Can have multiple handlers per event, events per handler
- Behave like `RealtimeThread`, or like `NoHeapRealtimeThread` if given a scoped region
- Can block
- `POSIXSignalHandler` if OS supports Posix

Asynchronous Transfer of Control

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Java's `longjmp`
- A method must declare `AsynchronouslyInterruptedException`, otherwise it is postponed
  Don't break code which doesn't expect to be interrupted
- Also postponed if in a **`synchronized`** block
  Don't leave shared objects in an inconsistent state
- Use `happened()` to find out if the AIE you got is the one you expected
- `doInterruptible()` if it is, `propagate` it if not

ATC - II

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Class `Timed` for Real-Time timers (throws an AIE)
- Timer restarted after `doInterruptible()`
- `resetTime()` to use another duration
- `disable`, `enable`, `fire`, `propagate`, `isEnabled`, `happened`
- Blocking `java.io.*` classes must know of AIE (unblock, raise `IOException`, or complete normally. . .)

Asynch. Termination

- Allowing it everywhere, always is unsafe
- So we use ATC to do it
- An AIE is treated as a normal exception, *but* a **`catch`**`( ALL )` doesn't catch it, unless we specifically test for the AIE with `happened`

Other Proposals

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- RTCE & HIP- from the J-Consortium.org
- Ravenscar-Java- from the Ada community
- JTRON- from Japan
- Java 2, Micro Edition (J2ME) - from SUN, for embedded systems

Why Java?

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Easier to write correct code (threads, monitors, exceptions)
- Many libraries
- Strongly typed
- No out-of-bound accesses, dangling pointers
- Automatic GC- no more leaks
- Easier to have it optimised
- It has all the buzzwords. . .

Why Java for Real-Time?

Introduction
What's Java?
Java vs C/C++
From C to Java
Java Threads
Thread Example
Interrupts
Monitors
Timers
JNI
CNI
Disadvantages
Speed
JIT or AOT?
Real-Time GC
RTSJ
Scheduling
Memory
Synchronisation
Asynchronous
Other Proposals
Why Java?
Java for Real-Time?

- Others are already using it
- Tools are getting better (and more)
- Using C/C++ is tedious, error-prone, difficult to maintain and to have it validated

---

- Embedded systems is a huge market
- Software is replacing hardware
- Too complex for todays tools and languages
- Java promises to help us

Cannot fight a dream. . . So join it!