1. Notes about this book
    1. Possible titles
        1. Practical embedded Java
           Note: A pragmatic approach to using native execution Java in real-world systems
    2. This is not a market survey
       Note: This is not a survey of all JVMs or embedded Java systems, it's an in-depth hands-
       execution Java systems. But there will be some mention of the new TINI400 and maybe
    3. CDROM contents
       Note: JEdit, ANT,

       With setups and examples for TINI/TStik, JStamp,JStik and SaJe
2. What's so great about Java?
   Note: Why is embedded Java the biggest news since the appearance of C 25 years ago?

   Benefits of using Java: 1)APIs such as TCP/IP, serial I/O and graphics are included in the lan
   extensions to it 2) Java code can be more reliable (no memory leaks) 3) all team members ca
   APIs on embedded, PC/gateway, and server side 4) at least 2X faster dev than C/assy 5) red
   costs, 70% of which is maintenance and upgrades
    1. Learn once, write anywhere
       Note: Learn java once, write code for embedded, PC/clients, servers, anywhere Java is d
    2. Robust applications
    3. Develop in half the time (or less) of C/C++
    4. Standard APIs: serial I/O, network, graphics, etc
    5. RTSJ standards for realtime control
       Note: Imagine - a standardized way to support realtime on multiple hardware! This is som
       realized with vendor specific C libraries
    6. Huge Java Community Process for extensions
    7. High quality multi-platform development tools
    8. Robust threading model built in
    9. Exception handling mechanisms built in
    10. Memory management built in
    11. Packages make code distrib and support easier
    12. Code documentation tools built in
    13. Wealth of open source code available
    14. Multi-platform, multi-vendor support
    15. Everything (well almost) is an object
        Note: The natural world can often easily be viewed through an OO paradigm. Subsump
        example - it's how most natural organisms function. Layers of behaviors. If you are hun
        predator is attacking, then all that matters is survival.
    16. True team collaboration
        Note: all members of a project team using Java can share common tools, APIs, and ca
        collaborate, from the embedded device, to the PC, to enterprise servers. Try that with th
        of C/assy (embeddedd), Visual BASIC or different C (PC/Client), and Java/database on
    17. Summary of "why Java?"
        Note: So to summarize - Java is not perfect, but I'd rather write embedded code in Java
        day. All the basic Java syntax is C, anyway, so C coders will feel at home. Get comforta
        halfway there.

        "Java is C++ done right, without pointers"
3. OK, there must be a dark side to Java

1. details of I/O drivers often hidden from you
   Note: System programmers often need to really understand what's going on in the bowel
   tends to hide exactly this sort of understanding from you.
2. can't count cycles to predict loop timing
   Note: but this only works on simple non-pipelined small micros anyway
3. garbage collection can interrupt your app
   Note: But there are ways to deal with this, especially with RTSJ. But you do need to think
4. You need to think in terms of objects
   Note: This can be difficult for the non-OOP programmer (me for example). But if I can do
5. Some Java protection comes with a price
   Note: array bounds checking for example, does slow down array access.
6. Firmware JVMs are typically big and slow...
   Note: ...but that's why we are using native execution hardware, so many of those speed

4. Where Java doesn't fit
   1. Really small devices
      Note: Really small devices such as rice cookers running on 4-bit micros. You think I exag
      sales person told me that some years ago (1995?) the largest customer of MicroChip wa
      cooker manufacturer. Millions per year.

      Although there is the Javelin Stamp and the possible new uVM
   2. DSP systems
      Note: DSP uses peculiar architectures and so far, there is not a good way to map Java o
      DSP chips.
   3. Stable legacy apps
      Note: there is no practical benefit to re-coding stable legacy apps in Java just to say they
      especially if these apps do not need major upgrades in the future. A lot of embedded dev
      never/seldom changed. If it ain't broken, don't fix it.
5. Embedded Java software & extensions
   Note: Java editions and configurations: Java Card, J2ME, CLDC and CDC, MIDP, RTSJ

   Give a quick overview, focussing on the perspective of embedded developers. References to
   1. JDK1.4.1
      Note: What's special about it (when used with embedded systems) compared to 1.3.X, s
      javadocs, regular expressions package, etc
   2. J2ME
   3. CLDC 1.0 and 1.1
   4. CDC
   5. MIDP
   6. Javaxcomm
   7. RTSJ
   8. XML and SOAP
      Note: small versions usable even on JStamp
   9. JXTA
6. Applying objects in embedded systems
   1. Typical multi-tiered projects
      1. embedded end - small systems
      2. PC or local clients
      3. Server side or company wide hosts
   2. Partitioning your project
   3. When and what to encapsulate
   4. Sharing code across a team

5. UML applied to an embedded project

7. Fundamental concepts in embedded Java
    1. Use packages to your advantage
    2. class.forName and system properties
    3. Event handlers
       Note: Much like interrupt handlers in C or assy
    4. Memory allocation & initialization
    5. Threading
    6. Initializers and finalizers
    7. Exception trapping and handling
    8. Flash file system
    9. coding with sockets and streams
        1. avoid these common pitfalls
            1. using new() in a loop
            2. polling instead of interrupts
            3. use a buffer then process the data
            4. keep interrupt handlers small and simple
            5. use javaxcomm whenever possible
    10. Keeping code as portable as possible
8. Java Controllers
   Note: Emphasize the smaller controllers, under $500. PC104 or larger systems running firmw
   OSes are not particularly interesting, at least to me.
    1. Java byte codes and class files
    2. Firmware JVM Embedded Modules
       Note: Just a brief overview of what interps are. Not in depth (this would be a book in itself
        1. Javelin Stamp (? - it's not really Java)
        2. TINI 80C400 and TStik
    3. Some other firmware JVMs
       Note: JVMs for other platforms (not embedded modules) such as the PalmOS PDAs, Poc
       Savaje.

       Typically these JVMs are not designed for only one chip or device but support a family of
    4. Native Execution Java
        1. aJile aJ80 and aJ100
        2. Systronix JStamp
        3. Systronix JStik
            1. HSIO Bus
            2. JSimm interface
        4. Rolling your own aJile native execution hardware
        5. Other possible Java controllers
           Note: Patriot Scientific (had an empty booth JavaOne2002)
           DCT (may have a chip in 2003)
           ImSys CJip (not native execution)
9. Real Time Java
    1. RTSJ spec overview, references
    2. garbage collectors
    3. writing realtime code without garbage collection, even without RTSJ support
10. Project architecture & tools
    1. JBuilder setup
        1. CLDC runtimes

        2. javaxcomm
        3. other libraries
    2. JemBuilder and Charade
    3. JSwat source debugger
    4. Arranging Java project folders
        1. source, class and doc folders
        2. separate flash and ram build output folders
    5. JEdit and ANT

11. Debugging embedded Java projects
    1. Debugging classes on other hardware
        1. command line regex debugging of classes
           Note: Debug classes on a PC, using JDK1.4 regular expressions and introspection a
        2. debugging class example - image processing CheckerBot
    2. Debugging on the embedded hardware
        1. printed output
        2. I/O bits
        3. JTAG debugging interfaces
        4. byte code debugging
        5. source level debugging
12. Interfacing to external hardware
    1. Memory mapped I/O
    2. SPI
        1. Systronix SPI address expansion
        2. XML tagging memory
        3. Master and slave
    3. I2C
    4. CAN
13. JCX - Java computer for Legos
    1. Lego 2-wire sensor interface
    2. Lego motors
14. Embedded Java networks
    Note: Only cover embedded networks with specific Java support
    1. ethernet
        1. 10 MBit to 100 MBit
        2. UDP, HTTP, TCP/IP
        3. sockets and streams
    2. CAN
        1. 128 Kbit to 1 MBit
        2. CAN Kingdom
    3. Dallas 1Wire
        1. 14 KBit to 140 KBit (approx)
        2. available devices
        3. tunneling other I/O devices over 1Wire
    4. wireless
       Note: power considerations esp with 802.11 and Bluetooth
        1. ethernet 802.11a and b
        2. Bluetooth
        3. RF modems 900 and 400 MHz
        4. JXTA protocols

    5. XML and SOAP

15. SPI masters and slaves
16. Device drivers in Java
17. Event Handlers and Interrupts
18. Http servers and Servlets
19. Threading and interrupts in a real project
    1. Typical Interrupt sources
    2. Cooperative multithreading
    3. Periodic Threads (preemptive scheduling)
20. LCD/touchscreen GUI
21. A multi-CPU embedded system
    1. Java version of Occam
22. Examples and App notes
    Note: These could all be at the end or mixed in with the body of the book as appropriate
    1. easy serial I/O using the javaxcomm event models
    2. Building a 50 nsec resolution pulse generator - in Java!
       Note: This project uses a JStamp development kit, and an LCD/touchscreen module to b
       equipment - a programmable digital pulse generator with 50 nsec resolution!
    3. Periodic threads - a simple scheduler
       Note: Write a simple wrapper to make sched threads and piano roll look like some of the
    4. Sonar rangefinder
    5. R/C Servo control
       Note: Driving multiple radio control servos from a Java chip. These servos are widely use
       scale models.

       Drive 8 servos with my bizarro idea, using one timer. Max servo pulse is 2 msec, and you
       every 20 msec, so 8 x 2 = 16 msec, leaving 4 msec of space within the 20 msec allowed.

       Use 2 timers, each in its own thread, each driving 8 servos.
    6. CMUcam vision sensor
       Note: examples would be the hockeybot and checkerbot
    7. Speech recognition and synthesis
    8. Controlling a 4WD R/C model chassis with realtime Java
       Note: Build a powerful 4WD all-terrain autonomous robot starting with a rugged chassis w
       $50.
    9. Fantazein oscillating wand sign
       Note: Driving an oscillating-wand clock from a stock JStamp development station. The m
       LED wand appear to float in space and are a popular feature in our trade show booth. Th
       code and info can be downloaded from jstamp.com It's a good example of realtime Java
       written to every 500 usec without fail or jitter.
    10. Oscillating wand stockticker/clock/weatherstation
        Note: a JStik project which maintains the wand display and also retrieves realtime stock
        time from the internet, along with weather conditions from a Dallas OneWire weather sta
        this app note is complete and is being shown in our booth in 2002 starting with JavaOne
    11. An inexpensive realtime Java robot platform
    12. A simple peer-to-peer ad-hoc RF network with JXTA
        Note: JXTA is an open-source network protocol which is media agnostic. It can work wit
        Bluetooth, or other protocols. www.jxta.org
    13. LCD and touchscreen GUIs

14. The JavaOne 2002 realtime Java sumo robots
    Note: JavaOne 2002 featured two sumo robots - one using R/C control and driven by a
    controlled by realtime Java running on a JStamp. Here's a look at the design and progra
15. University of Utah HockeyBots
16. University of Utah CheckerBot