Java: Where Does it Fit?

Dr. Timothy C. Lethbridge

School of Information Technology & Engineering (SITE)

Computer Science

Outline

- What is Java?
- Key features
- Key problems
- Technology choice
- When should you use it?
- When should you not use it?
- Why the hype?
- What is its future?

1. What is Java

- An object-oriented programming language
- Developed by Sun Microsystems
- Open standard with many vendors providing add-on facilities, environments
- Syntactically similar to C/C++
  - With dangerous and complex elements removed
- Do not confuse it with 'Javascript'

2. Key features

- Object oriented (almost pure)
- Compiles to portable ‘bytecode’
  - Thus portable
  - Each platform has a ‘Virtual machine’
- Innovative technology for secure downloading of applets over the Internet

Key features continued

- Comes with useful class libraries:
  - Graphics
  - Networking
- Contains many of the latest ‘good ideas’ in programming language theory
  - Strongly typed
  - Threads (concurrency)
  - Exception handling

Key features continued

- Interesting add-on technologies
  - Database access (JDBC)
  - Java Beans (application components)
  - Multimedia support
  - CORBA support
  - Intelligent agents (Aglets)

3. Key problems

- Immature (growing up too fast)
  - Inconsistencies across platforms
  - Original class library had many bugs and gaps
  - Many different class libraries now being developed (standardization becoming harder)
  - Tools still underdeveloped

Key problems continued

- Internet download paradigm is oversold
  - Congestion will not go away
  - Software size and complexity always fills any increase in capacity
- Lack of experts
  - Learning curve is underestimated
  - Much junk is being produced
  - Will haunt us in future

Key problems continued

- Still too complex
  - C-like syntax is difficult to read
  - Certain common programming tasks are hard to do cleanly
    - Enumerations required (no function pointers or blocks)
    - Basic data structures weak

4. Technology Choice

- Don’t design systems around a language or technology
  - Analyse real, minimal requirements
  - Then pick whatever technology has lowest cost-benefit ratio

Technology Choice - 2

- When choosing a language, consider:
  - Cost & ability to develop initially
  - Cost & ability to maintain
- Both affected by
  - Language complexity (moderate)
  - Programming environments (ok)
  - Reusable components (+ and -)
  - Personnel skill (growing slowly)
  - Stability (watch out for new releases)
  - Interoperability, portability etc.(good)

Technology Choice - 3

- Programming environments
  - Programming and maintenance are *much* faster with:
    - Incremental compiler
    - Informative browsing and search
    - Integrated configuration management
  - Java appears to be getting better in this area

Technology Choice - 4

- Current Java programming environments
  - Code oriented tools
    - Roaster (Natural Intelligence)
    - Café (Symantec)
    - CodeWarrior (Metrowerks)
  - Tools that allow visual construction of applications
    - Visual Café
    - Mojo (penumbra)
    - Visual J++ (Microsoft)
    - Java Workshop (Sunsoft)
    - Supercede (Asymetrix)
    - VisualAge for Java - beta (IBM)

Alternatives...

- Alternative 1: Smalltalk
  - Better in many ways, except
    - Expensive and proprietary
    - hard to distribute small programs
  - Environment, flexibility and configuration management are its key strengths
  - Good for complex applications (e.g. financial analysis)
- Alternative 2: C++
  - Best for systems programming and small desktop programs
  - Far more complex than Java

Alternatives ...

- Alternative 3: Delphi
  - Not as widely accepted yet
  - A language to watch but proprietary
- Alternative 4: Visual Basic
  - Not object oriented; proprietary
  - Useful as macro language within Microsoft tools
- Others (4GL's, Cobol)
  - Most not object oriented
  - Most proprietary, uncertain future, or hard to find experts

5. When should you use it?

- Applets
  - But don't just create applets for decoration!
- Moderate-volume servers
- Small, portable applications
  - Where portability is dominant factor
- Experimental software

6. When should you not use it?

- Large data processing systems
  - Never designed for large scale processing
  - Too slow
- Desktop applications where speed is required
  - Too slow
- Be cautious about rapid prototyping
  - Syntax too finicky to code directly
    - Rely on visual tools only
  - Tools like Visual Basic, Smalltalk more stable

7. Why the hype?

- It has significant good ideas, particularly transparent loading over the Internet
- The world yearns for the 'ultimate' language
  - Truly portable
  - Minimal learning curve (based on what they already know)
- People are discovering the nightmare of C++

Why the hype ... continued

- Distribution is free
- Tremendous volume of publicity
- Not enough experience to learn about the dark side
- Business leaders feel they can't afford to be left in the cold
  - Better to fail with everybody else than fail on your own

8. What is its future?

- Will remain very popular, but hype will fade
- Will mature and be one of 3-4 leading languages (1998-2000)
- Many new versions and add-on technologies
- Will probably go through 10-year cycle like C++ and then be replaced by the next big hype

Conclusions:

- Don't jump on the bandwagon without careful analysis
- You do not always need the 'latest tech'
- Don't rely on the trade press (or academics!)
- Java may or may not be right for you
- Final tip: Focus on good people not on programming language when initiating a project