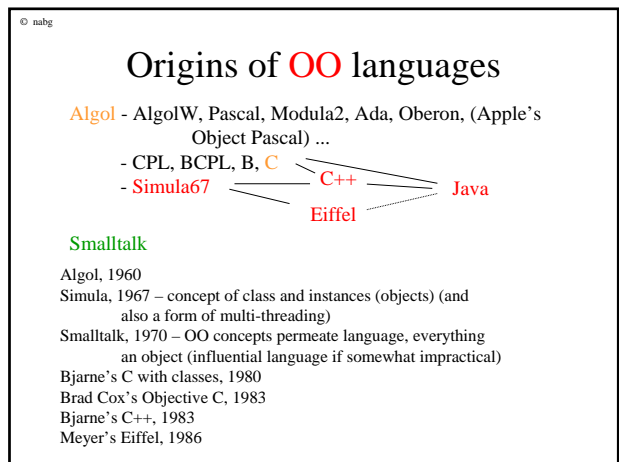


© nabg

Java

- © nabg
- Java
 - Just another OO language
 - Why Java?
 - origins

- © nabg
- ## Java language
- Java is “just another Object Oriented language”.
 - Its syntax is similar to C++ (very similar, but there are lots of small differences which result in a high frequency of syntax errors if you are using both Java and C++)
 - Java’s style is closer to “purer” object oriented languages (Eiffel, Smalltalk, ...).



- © nabg
- ## 1988 – the year of objects
- Simula-67 “Weird; guess it has some special uses; not for me though”
 - Smalltalk (1980) “That freaky language from Xerox Park, I wonder what they were smoking”
 - Brad Cox (~1983) “Software ICs” – “hey, there could be something in this Object stuff”
 - Apple Lisa & Classcal (~1983) “I see Steve Jobs has been sharing smokes with the Xerox Parc lot”
 - Classes in C (1980 ... 1983) “OK Bjarne, but it would be better if it also included ...”
 - C++ (1983 ...) “OK Bjarne, but it would be better if it also included ...”
 - Object Pascal and MacApp (1985) “Well I suppose if you must develop software for those crappy little Macintoshes, then this could be a better way”
 - Eiffel (1986) “Who is this French git? Telling us all how to develop software. Talk about conceited! Did you see how big and slow it was?”
 - C++ 1988 “Are you still using C? Pathetic! Objects are the only way to go! Have you tried Gorlen’s class library; it’s great. The new compiler generates really fast code. ...”

- © nabg
- ## Cultural revolution
- Overwhelming shift toward classes and objects ~1988
 - Everything and everyone had to be object oriented
- Well, maybe not everyone; there are still a few who don't really appreciate objects

C++

- Successful
 - Evolution rather than revolution
 - Programmers with C background could “grow” into use of classes and objects
- Efficient
- Multi-paradigm!
 - OO isn’t the only approach
 - Sometimes older procedural styles more appropriate

C++ programming styles

- C++ supports several programming styles:
 - **conventional procedural program**
 - main(), functions, global data
 - design by top down functional decomposition
 - **hybrid style with objects** (instances of simple classes)
 - objects perform well defined roles – e.g. string objects used in your C++ programs
 - overall control in “free functions” from which calls to objects are made
 - design
 - create some useful classes for component objects,
 - work out overall flow of control and put in free functions
- *(this is the style that you have been taught in first year!)*

C++ programming styles

- **fully object based**
 - main() creates single instance of principal object
 - program works through interactions among objects as defined in their member functions
 - design - “world of interacting objects”
- **object oriented**
 - object based, *plus* use of inheritance (allowing polymorphism etc) and use of elaborate class libraries
 - design in the context of the class libraries

OO style

- Java favors the OO style.[¶]
- Programs have a “main()” that creates the principal object, all subsequent control flow involve object interactions.
- *Libraries defining elaborate hierarchies of classes are an intrinsic part of OO languages (Java’s term is “package” rather than “library”).*

[¶]You can program hybrid style in Java – sometimes appropriate, e.g. little programs that submit queries to databases or which invoke services across Web

OO design with libraries (packages)

- You design your programs in the context of the class packages
 - “We need to communicate with the user, we will use
 - a scrolling (output only) text pane - *instance of class TextArea*
 - a set of standard action buttons for processing options ... - *instances of class Button*
 - a “pop-up” menu to deal with choices for ...
 - ...
 - “we also need to contact an Oracle(Access) database, we will use an instance of class ... – i.e. *a standard class Connection object that employs a driver object from the Oracle library (or a driver object from the odbc library)*

OO design with libraries

- When making the initial breakdown of a problem, you think in terms of the reusable classes from the library.
- Some of these classes define complete subsystems,
 - you may think and design in terms of using an instance of class X
 - what you get is an X object and a number of objects of other classes, these work together to achieve goals (library code defines their interactions)

Learning Java

- Language?
 - Yes, you have to learn a new language; but if you know C++ there isn't really much new to learn (just which parts of C++ to leave out)
- Libraries??
 - “learning Java” will to a larger extent involve learning about the libraries, and design ideas for using libraries, etc

Java?

- Why a new language?
- Why didn't the originators simply provide their class libraries in C++ and work with C++?
- Why Java?
 - To understand that, we will have to review its origins and development.

- Java
 - Just another OO language
 - Why Java?
 - origins

Java origins

- Java comes from Sun's research labs.
- Sun has several groups working on medium term speculative research - things that may become products in a few years, or may prove impractical.
- One project, started around 1990/1991, was to develop software for consumer durables.

Java origins

- Consumer durables?
 - Despite the jokes, not toasters.
 - Things like controllers for
 - cable TV decoders
 - video machine controllers
 - facsimile machines
 - bank teller machines
 - controllers in cars and other machinery
 - ...

Embedded controllers

- Characteristics -
 - limited memory
 - limited cpu power
 - lots of different architectures
 - relatively low computational demands
 - typically waiting for user input or external events
 - limited amount of processing for each input/event
 - nothing too time critical

Code for controllers

- Since it was the early 1990s, the new project naturally started with the assumption of object-based/object-oriented programming styles.
- The default language was C++.
- This soon proved inappropriate.

Why not C++?

- First reason was the problem of multiple architectures, some having under powered and exotic cpus.
- Too difficult to get cross-compilers (compilers that run on your main development machine and produce code for different target machine) for all these different cpus.

Why not C++?

- C++ didn't match the applications
 - too concerned with efficiency of code
 - example: C++ programmers take responsibility for allocating and releasing memory
 - you need this for efficiency (as best performance when these operations are carefully tuned)
 - but it adds to complexity of code
 - It is error prone!
 - *too many programming paradigms*
 - *easy for programmers to drop back into "bad" old procedural style habits*

Why not C++?

- C++ too complex; e.g.
 - multiple inheritance
 - C++ rules are complex
 - Complexity relates to support for specialized, *atypical* uses of multiple inheritance
 - templates
 - A sophisticated approach to generic code, - but can achieve much the same results by cruder expedients

Why not C++?

- C++ has maintained "backward compatibility" with C.
- Not required for intended applications.
- But is a source of a large number of errors
 - C permits (encourages) operations on pointers, operations that are unsafe
 - C provides a very poor, insecure model for array
 - ...

Something simpler than C++

- Cut back on the sophisticated but rarely used features:
 - multiple inheritance
 - user defined types having same support as built in types
 - operator overloading
 - ...

Something simpler than C++

- Leave out error prone features
 - no #defines, macros, ...
 - no access to pointers (no `int *p; p++;`)
 - no global data
 - no free functions
- Substitute simple expedients for things like templates.

Something simpler than C++

- Support for the programmer
 - provide more automation for free storage management
 - “garbage collection” of unused data structures
 - less efficient, more time consuming, but eliminates many errors and simplifies programming; efficiency not critical in intended language applications

Something easier to implement on multiple platforms

- *How do you get code that can run on all sorts of different machines?*
- Generate code for a single idealized computer.
- Simulate that ideal computer on all the real machines.

Ideal machines?

- Compiling code for an “ideal” machine and running simulators on actual computers is an old trick.
- Prior to Java, best known example was Wirth’s Pascal compiler (early 1970s).
- Smalltalk, a pioneer OO language, had also used this approach (Smalltalk evolved in 1970s, standardized 1980).

Conventional compiled programs & “Byte” code interpreters

- Conventional approach (C, C++ etc):
 - compiler translates program source to assembly language of target machine
 - assembly language converted to bit patterns representing actual instructions as interpreted by hardware CPU
- tightly bound to target, efficient

Conventional compiled programs & “Byte” code interpreters

- Byte code approach:
 - have an imaginary “virtual” computer with a simple instruction set (often, one byte is used to represent the instruction - i.e. 256 different instructions possible, though usually less - hence name “byte code”)
 - compiler generates code for this imaginary computer
 - simulator program models the computer

Byte code interpreter

- Virtual machine designed to make it easy to write a simple, compact simulator.
- So, write one compiler
- and lots of simulators - one for each target cpu.
- Simulator will need to invoke a function (and execute tens to hundreds of instructions) to model each interpreted virtual instruction - so relatively inefficient.

“Green” project and Oak

- So Sun started.
- Target applications: those consumer durables.
- Language: (at this stage called “Oak”, changed to Java later for copyright reasons), a simplified C++ with extras like “garbage collection” (automatic management of the freestore “heap”- memory).
- Approach: compile to byte-codes, have interpreters running on the little embedded cpus.

“Green” project and Oak

- *It was a failure.*
- Around 1993/1994, there weren't the applications
 - embedded systems in video controllers etc simply weren't elaborate enough to need anything like the scheme Sun was offering
- The Green project was doomed.

World Wide Web

- By 1994, use of the World Wide Web had started to grow substantially and capabilities of browsers were growing
 - first browsers, text and hypertext links
 - Mosaic browser, had added images etc
 - now had interactive web use
 - Web pages with forms entry,
 - User input sent to programs running on the server

Web interaction limited!

- Fancy Graphics?
 - Web pages could be set up that automatically asked server for a different picture every 15 seconds or so (very crude animation!)
 - “animated gifs” (several pictures concatenated together as if a single picture; display code shows each in turn, starts over when gets to end)

Web interaction limited!

- Forms entry:
 - clumsy, and puts a load on the server
 - request a page (network interaction with server)
 - display page and accept user input
 - create new “page request”; send it to server - this page request is actually a description of a program to be run on server computer and input to that program
 - main Web server program has to start separate program to analyse the received “page request”
 - separate program generates temporary Web page with results (may also update database etc)
 - Web program on server sends back results page

Client side computations?

- Many applications that would be nice to have on Web couldn't work this way --- too slow, too clumsy
 - e.g. imagine a “molecular modeler” that displays ball-and-stick molecules and allows them to rotate, it would be horribly slow if each 1° rotation needed an interaction with server.



But client running browser program is a computer, why shouldn't it do the work?

Client side computations?

- Client side computing?
- Nice idea, but **LOTS of problems**
- Lots of different clients - e.g. as minimum have 6 varieties of Unix, PC (with three PC op. Systems), Mac
 - do you have different downloadable code for each?

More problems - the “Web nasties”

- The Internet isn't a safe place.
- There are all those weirdos out there who write things like viruses, “Trojan Horse” programs, worms
 - they would think it a great joke if they could trick you into downloading a program that trashed your hard disk

More problems - the “Web nasties”

- Also security - if conducting commerce on web, you wouldn't like to download a program that kept some sales organization informed of all the things that you subsequently looked at.

Byte code interpreter and the Web

- But a system based on a byte code interpreter can be made:
 - platform neutral
 - “Web safe”

Byte code interpreter and the Web

- **Platform neutral:** you compile a single set of byte codes, these downloaded and run on interpreters specially written for each platform.
- **Web safe:** the byte code interpreter applies restrictions, e.g. can't access local disk (stops both the disk trashers and the commercial snoopers!)

A niche where Java may survive?

- The Web? Saved!
- Java has just what the Web needs for client side computing
 - platform independence
 - potentially, some mechanisms for security
 - easy to integrate into existing browsers

HotJava demo May 1995

- A Web browser program was written in Java
 - standard browser features (text, images, links, simple forms)
 - “applets” - provision for downloadable small Java programs that would run on client, using browser window for communication with users
- Java craze begins.

Java adoption

- Netscape committed to Java for January 1996 release.
- Java taken up by IBM, Symantec, Borland etc.
- Partly in response to its marketing as “Escape from Microsoft”, Java acquires very high profile.

IBM remains a strong supporter of Java. Its adoption of Java and Internet in mid-1990s were part of a brilliant management lead refactoring of a company that at that time appeared about to go bankrupt.

Sun’s Java release

- For Unix, Mac, and PC:
 - compiler (javac) compiles Java source to “class” files
 - interpreter (java) loads and runs class files
- these distributed free along with class library.

Sun’s Java release

- Java 1.0 came with a large number of standard libraries
 - awt a platform neutral “windowing” kit, (classes for buttons, menus, windows, dialogs etc)
 - net classes for “client-server” processing (communications across a network)
 - and more standard collection classes, utilities, etc

Sun’s Java release

- All libraries had substantial documentation (done as Web documents, so all cross references are hyperlinks)
- Sun provided on-line “Teach Yourself Java” tutorial

Sun's Java release

- Java released early 1996; v1.0.2 (minus some bugs) became the initial standard on Unix, Mac, PC.
- Version 1.1 available Jan 1997 but only for Unix and PC (Mac Javas start to lag).
- Version 1.2 been around since 1998.
- Version 1.3 ~2000.
- Version 1.4 available from Sun ~2002/3.
- Version 1.5 released late 2004
- Version 1.6, see discussion documents at Sun

Interpretation of revision level

- 0.1 -- WE GOT A REALLY GREAT NEW WAY TO DO THINGS !!!
- <0.9 -- Not ready for prime time.
- 0.9 -- We think it works, but we won't bet our lives on it.
- 1.0 -- Management is on our case; seems like a low risk.
- 1.01 -- Okay, we knew about that. All known bugs are fixed.
- 1.02 -- Fixes bugs you won't see in 27,000 years
- 1.03 -- Fixes bugs in the bug fixes.
- 1.04 -- All right, this REALLY fixes all known bugs.
- 1.05 -- Fixes bugs introduced in rev 1.04.
- 1.1 -- A new crew hired to write documentation.
- 1.11 -- From now on, no comma after "i.e." or "e.g.".
- 1.2 -- Somebody actually changed a functional feature.
- 2.0 -- New crew hired to write software. Old crew blamed for bugs.
- 2.01 -- New crew sending out resumes to placement agencies.
- 3.0 -- Re-write the software in another language, go back ten squares.
- ... -- return to line 0.1

Java versions : *confused numbering schemes*

- Java developers at Sun thought in terms of Java Development Kit - compiler + libraries
 - 1.0
 - 1.0.2
 - 1.1.1 ... 1.1.7
 - 1.2 ...
- Sales people felt that 1.2 didn't adequately reflect importance of all features in new libraries in this release.
- Sales people introduce alternative naming scheme, making **JDK-1.2** match to "**Java 2**"
- Java 2 seems to have been acceptable for 1.2...1.4,
- Java 1.5 ? Name "Java 5" is used (though not that consistently)

Java

- When approx 800 days old, about 800 different books published on Java; and now? Well Amazon has 1685 Java books in its Computer Books section.
- About half of them include a CD with Sun's Java Development Kit (JDK)
 - so if you have a PC, you had a Java for free

Java development

- Can use JDK
 - bit clumsy,
 - use some editor
 - run separate compiler
 - if compilation successful, run interpreter
- Integrated development environments available - costs \$50, ... \$thousands or free like Eclipse

Jikes

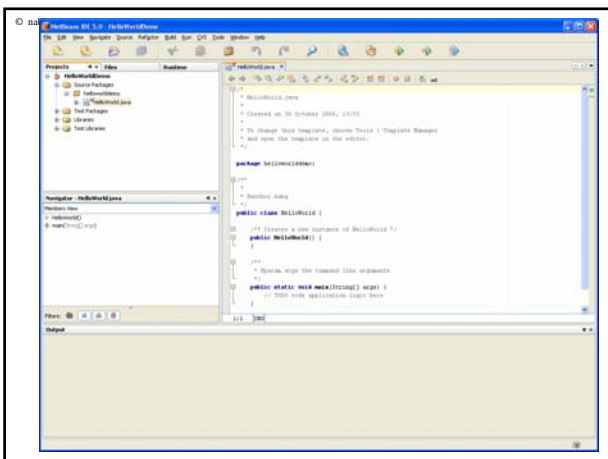
- IBM has a Java compiler interpreter system called Jikes
 - you can get it from IBM developerworks site
 - compiler is written in C and runs faster than javac
 - run-time maybe faster
- Inevitably, not as up-to-date as Sun compiler (so probably still 1.4; I've never used it, don't know)

JDK style development

- Use standard editor, enter Java code
- Compile with **javac**
- If “application”, run with **java**
- If Web dependent “applet”, run using IE, Mozilla or the simple “**appletviewer**” program

NetBeans

- This year we will be using Sun’s NetBeans Integrated Development Environment (IDE).
- NetBeans has been around for years
 - Used to be so slow it was painful
 - Modern machines so much faster, NetBeans runs reasonably!



Changing usage of Java

- Applets!!!!!!!!!!
- Er, uhm
- Maybe not.
 - General acceptance of lightweight browser client that does little work, all work done on server
 - Alternative technologies for prettying up web-pages
 - For fancy graphics – maybe “Flash”
 - For limited data validation of forms data entry use scripting (Javascript)

Real Java usage

- Very few Applets (*seems that they are used in SOLS and it the newer WebCT, also many sites load a tracker applet that monitors site usage and reports statistics*)
- Applications:
 - Multi-platform
 - Good graphical user interface
 - Much more productive development environment than C++
- Enterprise applications:
 - Java on server side of large networked applications
 - Java handles interaction with databases and enterprise systems (SAP, transaction monitors, messaging)
 - Orders of magnitude more productive than C++

NB

- Java never intended as just a cutesy language for small Web based applications.
- Original role, embedded systems, has been revived and may yet prove Java’s principal area of application.
- Java can be used for large applications, more typically done in C++.
- Java can be compiled (to real machine code, not byte code), so can be efficient.

Other Javas

- Some of you have Java on your mobile phones
- Other embedded Javas
- Smart card Java
- etc