

Java

CS2704: Object-Oriented Software Design and Construction

Constantinos Phanouriou
Department of Computer Science
Virginia Tech

CS2704 (Spring 2000)

1

C++ programming styles (2)

- 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

CS2704 (Spring 2000)

4

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 like Eiffel and Smalltalk.

CS2704 (Spring 2000)

2

Object-Oriented Style

- Java (like Eiffel and Smalltalk) supports only the object oriented style.
- *Libraries defining elaborate hierarchies of classes are an intrinsic part of these languages*
- Programs have a “main()” that creates the principal object (this main() may be hidden, provided by the compiler system), all subsequent control flow involve object interactions.

CS2704 (Spring 2000)

5

C++ programming styles (1)

- 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
 - 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

CS2704 (Spring 2000)

3

Design with Libraries

- You design your programs in the context of the class 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)

CS2704 (Spring 2000)

6

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

CS2704 (Spring 2000)

7

Why not C++? (1)

- 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

CS2704 (Spring 2000)

10

Why Java?

- Why a new language?
- Why didn't the originators simply provide their class libraries in C++ and work with C++?
- Let's review the origins of Java and its development

CS2704 (Spring 2000)

8

Why not C++? (2)

- 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
 - too many programming paradigms
 - easy for programmers to drop back into old procedural style habits

CS2704 (Spring 2000)

11

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.
 - cable TV decoders, VCR controllers, automobile controllers, etc.
 - limited memory, limited cpu power, lots of different architectures

CS2704 (Spring 2000)

9

Why not C++? (3)

- 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

CS2704 (Spring 2000)

12

Why not C++? (4)

- 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
 - ...

CS2704 (Spring 2000)

13

Something simpler than C++ (3)

- 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

CS2704 (Spring 2000)

16

Something simpler than C++ (1)

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

CS2704 (Spring 2000)

14

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.
- Compiling code for an “ideal” machine and running simulators on actual computers is an old trick.

CS2704 (Spring 2000)

17

Something simpler than C++ (2)

- 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.

CS2704 (Spring 2000)

15

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

CS2704 (Spring 2000)

18

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

CS2704 (Spring 2000)

19

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)

CS2704 (Spring 2000)

22

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.

CS2704 (Spring 2000)

20

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 modeller” 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?*

CS2704 (Spring 2000)

23

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

CS2704 (Spring 2000)

21

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?
- Also, the Internet isn't a safe place
 - There are people out there who write things like viruses, “Trojan Horse” programs, worms

CS2704 (Spring 2000)

24

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

CS2704 (Spring 2000)

25

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 (now 1.1.8 for bug fixes) 1997
- Version 1.2 (now 1.2.2 for bug fixes) 1998
- Version 1.3 has now been released

CS2704 (Spring 2000)

28

A niche where Java may survive?

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

CS2704 (Spring 2000)

26

JDK style development

- Use standard editor, enter Java code
- Compile with **javac**
- If “application”, run with **java**
- If Web dependent “applet”, run using a Java browser (e.g., Netscape, Explorer, or HotJava) or the simple “**appletviewer**” program

CS2704 (Spring 2000)

29

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.

CS2704 (Spring 2000)

27

Java Language

- Language features:
 - Data types
 - Variables (and constants)
 - Operators
 - Expressions (same as always)
 - Control flow
 - Functions

CS2704 (Spring 2000)

30

boolean

- `boolean` `true` `false`
- *boolean is not compatible with integer types*
- Significant tidy up from C where have automatic conversion of integer type to/from boolean (0 false, non-zero true)
 - an automatic conversion that is often source of errors

CS2704 (Spring 2000)

37

Variable declarations

- Variables can be declared as -
 - instance data members (each object created from class has its own copy)
 - class data members (one copy of variable shared by all instances of class)
 - local variables of member functions

CS2704 (Spring 2000)

40

Variables

- Different treatment of built-ins and class instances
 - built in types on stack (or “static data segment”)
 - class instances on heap
- (In C++, don't have this restriction).
- No global (or filescope) variables of any type.

CS2704 (Spring 2000)

38

Variable declarations

- Simple declarations -

```
int        counter;  
double    slope, intercept;  
char      mark;
```

 - *data members are initialized (zero numerics, false booleans)*
 - *local variables of functions are **not** automatically initialized*

CS2704 (Spring 2000)

41

Variable names

- Variable name must start with a “letter”
 - any letter from any Unicode alphabet, but stick to a..z, A..Z, _, \$
- Variable name can contain letters, digits

CS2704 (Spring 2000)

39

Variable declarations

- Declaration may provide an initial value (should do this for all local variables)

```
double        sumxsq = 0.0;  
char          hash = '#';
```

CS2704 (Spring 2000)

42

constants

- (The word const is a reserved word in Java language; currently it is not used.)
- Java has a qualifier “final” that can modify declarations
 - can specify a class as final (can’t make subclasses)
 - can specify a member function as final (subclasses cannot redefine that function)
 - can specify an initialized variable as final (initial value cannot be changed, so it is a constant)

CS2704 (Spring 2000)

43

Derived types? No

- In C/C++ you can declare variables with “derived types” like pointer types, arrays, references.
- You can’t do that in Java.
- Arrays will come soon, they are considered as being much the same as objects (class instances)

CS2704 (Spring 2000)

46

constants

- Since all data elements must be class members (or local variables in functions), constants are generally defined as final data members of a class.
- Since want only one copy of constant (not one for each instance), will make data element a class member (as in C++, this involves use of static qualifier)

CS2704 (Spring 2000)

44

operators

- Generally similar to those in C/C++.
- No operators relating to pointer types.
- An extra shift operator to resolve ambiguities that causes portability problems for C/C++ programs.

CS2704 (Spring 2000)

47

constants

- Typical example “PI”
 - in class Math

```
public final static
double PI = 3.141.....;
```
- order of qualifiers like final and static can be varied

CS2704 (Spring 2000)

45

Operator precedence table

[] array subscripting, . member access, () function call
! logical not, ~ bit not, ++, --,

* / %
+ -
<< >> >>>
<<< >>>=
= = !=
&
^
|
&&
||
?:
+= -= etc

Use parentheses on any expression complex enough to make you concerned about operator precedence!

CS2704 (Spring 2000)

48

Shift operators

- << shift left $n \ll 3$
- >> “arithmetic shift right”
- >>> “logical shift right”
- C/C++ leave meaning of >> undefined, compiler writer can chose either arithmetic or logical right shift (hence portability problems!).

CS2704 (Spring 2000)

49

Control flow

- Some oddities
“labels” and modified forms of “break” and “continue” (keyword goto is reserved but not used)

```
getRowData:  
for(int row=1; row<=nRows; row++) {  
    ...  
    if(!ValidateDataSoFar(row)){  
        System.out.println("data inconsistent!");  
        continue getRowData; }  
    ..  
}
```

CS2704 (Spring 2000)

52

operators

- operators are essentially only defined for built in (primitive types)
- Unlike C++, you can not define a ++ operator to work with instance of your classes
- One exception, Java did define operator+() for class String; can use + with String variables, it produces a new String by concatenating existing Strings (this operator will do type coercion, String + number invokes function to convert the number to a String).

CS2704 (Spring 2000)

50

Functions

- Form is generally similar to C/C++.
- But
 - all functions must be declared as member functions of a class (possibly static member functions eg. Math.sin(double), etc)

CS2704 (Spring 2000)

53

Control flow

- The usual -
 - selection
 - if
 - if ... else ...
 - switch
 - iteration
 - while
 - do ... while (repeat)
 - for(...;...;...)
 - better to use “Enumerators” (iterators) when working through collections

CS2704 (Spring 2000)

51

Functions

```
int Largest(int data[], int num)  
{  
    int largest = data[0];  
    for(int i=1; i<num; i++)  
        if(data[i]>largest) largest = data[i];  
    return largest;  
}
```

- void functions similar to C/C++

CS2704 (Spring 2000)

54

Program structure

- Superficial similarities to C/C++, but underneath there are significant differences.
- C & C++ build program by compiling source files to .o ('object') files and then linking these with libraries to get an executable.
- Java interpreter loads code for classes when it needs to, there is no real "link" step.

CS2704 (Spring 2000)

55

Runtime

- Two possibilities
 - simple applications run by the 'java' interpreter
 - applets (meant to be run by a browser, such as Netscape, which includes a Java interpreter)
- Either way,
 - byte codes of 'principal' class get loaded
 - "appropriate function" called, normally this will create one (or more) objects and get one to run
 - code for other classes loaded

CS2704 (Spring 2000)

58

Compiler

- With Java
 - a file will contain declaration of a principal class (and optionally some auxiliary classes)
 - all details of class in the one file (unlike C++, no separate header file, and no possibility of defining some member functions in one file and others in a different file)
 - name of file must match name of (principal) class (with file extension .java)
 - compiler creates files with byte codes (separate '.class' file for each class defined in the file being compiled)

CS2704 (Spring 2000)

56

Runtime

- Runtime system must 'know' where byte codes for other classes can be found
 - current directory
 - default directory (for classes supplied as part of standard Java release)
 - directories specified by CLASSPATH environment variable
 - possibly, loaded from remote site via Internet

CS2704 (Spring 2000)

59

Compiler

- A '.java' file will start with "import" statements
 - role has some similarities with #includes in C/C++
 - if you are going to be declaring and using instances of classes defined elsewhere, compiler is going to have to read descriptions of those classes so that it can verify that they support the operations that you invoke on the instances that you create

CS2704 (Spring 2000)

57

Conclusions

- Java never intended as just a fancy 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.

CS2704 (Spring 2000)

60