COS 126 Lecture 21: Java

Java is

- simple
- object oriented
- statically typed
- architecture neutral
- multi-threaded
- garbage-collecting
- robust
- small
- fast
- secure
- extensible
- well-understood
- fun

- not new
- designed for toasters
- not done yet
- not as useful as C++, C, FORTRAN, ...
- slow
- unsafe
- huge
- complex

Don't believe anything on this slide!

[make up your own mind]

Java perspective

Similar to C in many ways

    int, float, double, char ...
    for, if, while, do, switch ...

Modern primitives

    Ex: Unicode
    16-bit "universal" character set (65536 chars)

Higher level of abstraction than C

    object-oriented
    no pointers
        (references to objects)
    inheritance
    language support for
        interfaces
        multiple threads
        exceptions
    system support for
        software libraries
        security
        graphic user interface
        network

Ref: Arnold, Gosling "Java Programming Language"
Addison-Wesley, 1996

http://java.sun.com

"hello, world" in C

```
#include <stdio.h>
main()
  {
    printf("hello, world ");
  }
```

COMPILE and EXECUTE the program

```
% cc hello.c
% a.out
hello, world
```

"cc" COMPILES source into machine language

argument: hello.c

output: a.out

"a.out" LOADS and EXECUTES that program

output: [result of running program]

advantage

good compilers make efficient programs

disadvantage

translated code is machine-dependent

"Hello, World" in Java

```
class HelloWorld
   {
      public static void main(String[] args)
         {
            System.out.println("Hello, World");
         }
   }
```

Make sure java is installed. For arizona:

```
% which javac
no javac in /u/rs/bin /usr/princeton/bin ...
% set path = (/opt/jdk1.1.1/bin $path)
```

TRANSLATE and INTERPRET the program

```
% javac HelloWorld.java
% java HelloWorld
Hello, world
```

"javac" TRANSLATES source into byte stream
- argument: HelloWorld.java
- output: HelloWorld.class

"java" INTERPRETS byte stream on host machine
- argument: HelloWorld.class
- output: [result of running program]

advantage: architecture neutral

disadvantage:
- interpretation slower than compiled code

good native java compilers inevitable

Square root (bisection method) in C

[see programming assignment 2]

```
#include <stdio.h>
float epsilon = .000001;
float f(float x)
  { return 2.0 - x*x; }
float root(float l, float r)
  {
    float m;
    while (r - l > epsilon)
      {
        m = (l + r)/2;
        if (f(m) > 0) l = m;
        if (f(m) < 0) r = m;
        printf("%f %f  ", l, r );
      }
    return (l + r)/2;
  }
main()
  {
    printf("%f ", root(0.0, 2.0));
  }
```

Square root (bisection method) in Java

Can mimic C code with STATIC functions
"function-oriented" programming

```
class Bisect
   {
      static final double epsilon = .000001;
      static double f(double x)
         { return 2.0 - x*x; }
      static double root(double l, double r)
         {
            double m;
            while (r - l > epsilon)
               {
                 m = (l + r)/2;
                 if (f(m) > 0) l = m;
                 if (f(m) < 0) r = m;
                 System.out.println(l + " " + r);
               }
            return (l + r)/2;
         }
      public static void main(String[] args)
         {
            System.out.println(root(0.0, 2.0));
         }
   }
```

"main" must be public and static

Byte code example

```
 % od -c HelloWorld.class
. 000 312 376 272 276  \0 003  \0   -  \0      \b  \0 024 007  \0 025
. 020 007  \0 032 007  \0 033 007  \0 034  \n  \0 004  \0  \t  \t  \0
. 040 005  \0  \n  \n  \0 003  \0 013  \f  \0 020  \0  \f  \f  \0 036
. 060  \0 027  \f  \0 037  \0  \r 001  \0 003   (   )   V 001  \0 025
. 100   (   L   j   a   v   a   /   l   a   n   g   /   S   t   r   i
. 120   n   g   ;   )   V 001  \0 026   (   [   L   j   a   v   a   /
. 140   l   a   n   g   /   S   t   r   i   n   g   ;   )   V 001  \0
. 160 021   0   0   H   e   l   l   o   W   o   r   l   d   .   j   a
. 200   v   a 001  \0 006   <   i   n   i   t   > 001  \0 004   C   o
. 220   d   e 001  \0  \r   C   o   n   s   t   a   n   t   V   a   l
. 240   u   e 001  \0  \n   E   x   c   e   p   t   i   o   n   s 001
. 260  \0  \f   H   e   l   l   o   ,       w   o   r   l   d 001  \0
. 300  \n   H   e   l   l   o   W   o   r   l   d 001  \0 017   L   i
. 320   n   e   N   u   m   b   e   r   T   a   b   l   e 001  \0 025
. 340   L   j   a   v   a   /   i   o   /   P   r   i   n   t   S   t
. 360   r   e   a   m   ; 001  \0 016   L   o   c   a   l   V   a   r
. 400   i   a   b   l   e   s 001  \0  \n   S   o   u   r   c   e   F
. 420   i   l   e 001  \0 023   j   a   v   a   /   i   o   /   P   r
. 440   i   n   t   S   t   r   e   a   m 001  \0 020   j   a   v   a
. 460   /   l   a   n   g   /   O   b   j   e   c   t 001  \0 020   j
. 500   a   v   a   /   l   a   n   g   /   S   y   s   t   e   m 001
. 520  \0 004   m   a   i   n 001  \0 003   o   u   t 001  \0 007   p
. 540   r   i   n   t   l   n  \0      \0 002  \0 004  \0  \0  \0  \0
. 560  \0 002  \0  \t  \0 035  \0 016  \0 001  \0 021  \0  \0  \0   %
. 600  \0 002  \0 001  \0  \0  \0  \t 262  \0 007 022 001 266  \0  \b
. 620 261  \0  \0  \0 001  \0 026  \0  \0  \0  \n  \0 002  \0  \0  \0
. 640 005  \0  \b  \0 003  \0  \0  \0 020  \0  \f  \0 001  \0 021  \0
. 660  \0  \0 035  \0 001  \0 001  \0  \0  \0 005   * 267  \0 006 261
. 700  \0  \0  \0 001  \0 026  \0  \0  \0 006  \0 001  \0  \0  \0 001
. 720  \0 001  \0 031  \0  \0  \0 002  \0 017
```

Names, strings, ops in bytecode

actually enough info to DECOMPILE code

www.ahpah.com

Object-oriented programming

Direct support of "data type" concept
    encapsulate functions that manipulate data
    build levels of abstraction
CLASS
    data type descriptions
    definition of associated functions (METHODS)
OBJECT
    "instance" of class
    data values describe object "state"
        [each object has its own values]
    methods describe object "behavior"
        [methods are shared with other
            objects in the class]

METHOD (function)
    instantiates objects
    invoke other methods
objects are passed by REFERENCE
strong TYPE CHECKING
    types of method args, return must match
OVERLOADING: if arg types differ, methods differ

compare with C "user-defined types"
difference:
    uniform model, supported by language/system
Java program: set of classes

Class implementation for rational numbers

Class Rational defines

- data type (two integers)
- associated methods (mul and add)
- CONSTRUCTOR (create a new object)
- "toString" method (for output)

[see programming assignment 3]

```
class Rational
  {
    private int p, q;
    Rational(int p, int q)
      { this.p = p; this.q = q; }
    public String toString()
      {
        return Integer.toString(p) + "/"
             + Integer.toString(q);
      }
    void mul(Rational x)
      { this.p *= x.p; this.q *= x.q; }
    void add(Rational x)
      {
        this.p = this.p*x.q + x.p*this.q;
        this.q *= x.q;
      }
  }
```

"+" with string args: CONCATENATES them

Example Rational client

to declare variables

of type "reference to Rational"

```
Rational t;
```

to create a new Rational

```
Rational e = new Rational(1, 1);
```

to invoke a method

```
e.add(t);
```

"this" in code refers to e’s copy of data values

if p and q were "public", could say e.p, e.q

```
class e
  {
    public static void main(String[] args)
      {
        int i;
        Rational e = new Rational(1, 1);
        Rational t = new Rational(1, 1);
        for (i = 0; i < 6; i++)
          {
            t.mul(new Rational(1, i+1));
            e.add(t);
            System.out.println(e + " " + t);
          }
      }
  }
```

Easy to OVERRIDE or add new methods

```
class RationalA extends Rational
  {
    RationalA(int p, int q)
      { super(p, q); }
    private static int gcd(int m, int n)
      {
        if (n == 0) return m;
        return gcd(n, m % n);
      }
    private static void reduce(Rational x)
      { int t = gcd(x.p, x.q);
        x.p /= t; x.q /= t;
      }
    void add(Rational x)
      {
        p = this.p*x.q + x.p*this.q;
        this.q *= x.q;
        reduce(this);
      }
  }
```

ABSTRACT CLASSES and INTERFACES

no implementations in superclass

subclasses provide implementations

hierarchy can get complicated

support client-interface-implementation paradigm

Why Java?

Basic features present in older languages

- C++
- Modula-3
- Smalltalk
- Ada
- ...

What distinguishes Java?

- free
- easy to install (?)
- works on most machines
- modern, direct full-system approach
  - to software engineering
- growing library of software
- internet applications
  - applets
  - graphical user interface
  - [stay tuned]
- it *is* fun

SECURITY

- run programs in Netscape
- can "check" that they won't crash (?)

Bumps in the road

- huge amounts of old software
- libraries too complex
- disagreement on "standards"

[check what happened to older languages!]