Java
The Language

History

- 1990 - Started by James Gosling
- 1991 - Consumer electronics
- 1993 - Interactive TV
- 1995 - Internet & HotJava

Motivation

- Simple
- Efficient
- Small
- Architecture Neutral
- Secure
- Robust
- Multi-Threaded

Syntax

- Very Similar to C/C++
- Easy to Learn
- Fewer Concepts
  - No pointer arithmetic
  - No structs
  - No typedefs
  - No #define
  - No need to free memory
  - No core dumps

Example

```
int sum(int arr[]) {
   int result = 0;
   for (int i= arr.length ; --i >= 0 ; ) {
      results += arr[i];
   }
   return result;
}
```


Simple Types

- **boolean** (1 bit)
- **byte** (8 bits, signed)
- **char** (16 bits, unsigned, UNICODE)
- **short** (16 bits, signed)
- **int** (32 bits, signed)
- **long** (64 bits, signed)
- **float** (32 bits, IEEE 754)
- **double** (64 bits, IEEE 754)

More on Types

- **boolean** is a real type, it is not an integer!
- **null** is a keyword
- Integers are not pointers
- Additional type checking

```
boolean b = obj != null;

if (foo == null) {
   ...
}

// oops
if (a = b) {
}
```


Control Flow

- **if** (*expr*) *statement* **else** *statement*
- **for** (*expr* ; *expr* ; *expr*) *statement*
- **while** (*expr*) *statement*
- **do** *statement* **while** (*expr*);
- **switch** (*expr*) { ... }

Factorial

```
// Iterative
long faci(int n) {
   long result = 1L;
   for (int i = 2; i <= n ; i++) {
      result *= n;
   }
   return result;
}

// Recursive
long facr(int n) {
   if (n <= 1) {
      return 1L;
   }
   return facr(n - 1) * n;
}
```


Strings

- Strings are objects
- Strings are 16 bit UNICODE

```
System.out.println(“Hello World”);

String hello = “Hello”;
String world = “World”;
System.out.println(hello + “ ” + world);

int len = hello.length();
String str = hello.substring(1, len);
System.out.println(“j” + str);

char ch = hello.charAt(0);
```


Arrays

- Arrays are objects
- Arrays are dynamic
- Arrays are passed by reference
- Arrays are bounds checked

```
int rgb[] = new int[3];
rgb[0] = 255;
rgb[1] = 100;
rgb[2] = 100;

int avg = sum(rgb) / rgb.length;

// oops
int err = rgb[4];
```


Multi Dimensional Arrays

- Arrays of arrays

```
int mx[][] = new int[3][4];

int row[] = mx[0];
int rowsum = sum(mx[1]);

int total = 0;
for (int r=0; r < mx.length; r++) {
   for (int c=0; c < mx[r].length; c++) {
      total = mx[r][c];
   }
}

System.out.println(“total=” + total);
```

Garbage Collection

- Never have to free anything!
- No more memory leaks
- Conservative mark+sweep gc
- Most gc is done when idle
- Typical delay ~200ms for 4Mb

Classes

- Single implementation inheritance
- Multiple interface inheritance
- Runtime type information available
- All methods are virtual or static
- There are no functions - only methods

Class Example

```
class Car{
   void beep() {
      System.out.println(“beep”);
   }
   int maxSpeed() {
      return 55;
   }
}

class Porsche extends Car {
   int maxSpeed() {
      return 200;
   }
}
```


Class Example (continued)

```
Car c = new Car();
Porsche p = new Porsche();

c.beep();
p.beep();

int s1 = c.maxSpeed();
int s2 = p.maxSpeed();
```


Interfaces

- Defines an interface that a class must implement
- A class can implement multiple interfaces
- Very similar to IDL interfaces

```
interface Printable {
   void print();
}

class Foo implements Printable {
   void print() {
      System.out.println(“foo”);
   }
}
```


Printable List

```
void printAll(Printable list[]) {
   for (int i = 0; i < list.length; i++) {
      list[i++].print();
   }
}


Printable list[] = {
   new Foo(),
   new Bar()
}

printAll(list);
```

Dynamic Types

- Access to runtime type information

```
Car c = new Porsche();
String nm = c.getClass().getName();

if (c instanceof Porsche) {
   System.out.println("It's a Porsche!")
}

Porsche p = (Porsche)c;

// oops
Mercedes m = (Mercedes)c;
```


Access Modifiers

- **public** -Anyone can access
- **protected** - Access by subclasses
- **private** - Only access by the class itself
- The default is access by any class in the same package

Restricting Access to Variables

```
class Point {
   private int x, y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return x;
   }
   public int getY() {
      return y;
   }
}
```


Packages

- Organize class name space
- Default access boundary

```
package java.util;

public class Date {
   int now;
   ...
   public Date() {
   }
}


System.out.println(
   new java.util.Date());
```


Exceptions

- C++ exception model

```
try {
   int result = a / b;
   if (result > 5) {
      throw new ResultTooLarge(sum);
   }
   return result;

} catch (ArithmeticException e) {
   System.out.println("divide by zero");
   return -1;
}
```


Finally

```
InputStream in =
   new FileInputStream("index.html");

try {
   parse(in);

} catch (SyntaxError e) {
   error("syntax error: " + e.msg);
   throw e;

} catch (IOException e) {
   error("I/O Exception");

} finally {
   in.close();
}
```

Multi-Threaded

- Preemptive multi-threading
- No explicit locks
- Built-in primitives for synchronization
- Condition variables
- Better code structure

Multi-Threading Example

```
class EchoThread extends Thread {
   String msg;
   int delay;
   EchoThread(String msg, int delay) {
      this.msg = msg;
      this.delay = delay;
      start();
   }
   public void run() {
      while (true) {
         System.out.println(msg);
         sleep(delay);
      }
   }
}

new EchoThread("ping", 500);
new EchoThread("pong", 600);
```


Synchronized Methods

- Automatically lock instance or class
- Can be called recursively

```
class Stack {
   int sp;
   Object data[];
   ...
   synchronized void push(Object elem) {
      if (sp >= data.length) {
         throw new StackOverFlow();
      }
      data[sp++] = elem
   }
}
```


Native Methods

- Extend the Java runtime using C
- Dynamically load shared libraries

```
public class Date {
   int now;

   public Date() {
      now = time();
   }

   private native int time();

   static {
      Linker.loadLibrary("time");
   }
}
```


Libraries

- **java.lang** - Object, Thread, Exception, String, System, Number, ...
- **java.util** - Hashtable, Vector, BitSet, Regexp, Date, Cache, ...
- **java.io** - InputStream, OutputStream, BufferedInputStream, PrintStream, ...
- **java.tools** - Compilation, Debugging, Documentation, Profiling, ...
- Networking
- Abstract window toolkit

Compilation

- Compile Java source to Java bytecodes
- Compiler is written in Java
- No header files!

```
prompt> cat > HelloInternet.java
class HelloInternet {
   public static void main(String argv[]) {
      System.out.println("Hi internet!");
   }
}
^D
prompt> javac HelloInternet.java
prompt> java HelloInternet
Hi internet!
```

ByteCodes

- Simple instruction set for a virtual machine
- Register based
- Small stack for intermediate results
- Completely architecture neutral
- Byte order independent
- Easy to interpret
- Easy to generate machine instructions
- Type information included
- Deterministic behavior

Bytecode Listing

```
Method int sum(int [])
   0 iconst 0     ;; result = 0
   1 istore R1
   2 aload R0     ;; i = arr.length;
   3 arraylength
   4 istore R2
   5 goto 14
   8 iload R1     ;; result += arr[i]
   9 aload R0
  10 iload R2
  11 iaload
  12 iadd
  13 istore R1
  14 iinc R2 -1    ;; if cond goto 8
  17 iload R2
  18 ifge 8
  21 iload R1     ;; return result
  22 ireturn
```


Late Binding

- All method and variable offsets are determined at link time
- Bytecodes are rewritten the first time they are executed

```
getfield       Foo.stack
aload          R2
invoke         Stack.push(Object)

quickgetfield  #3
aload          R2
quickinvoke    #1
```


Performance

- Interpreted, 15-20 x compiled C
- Just in time compiled, about same as C

Small amount of extra overhead to do null pointer checking and array bounds checking.

Security

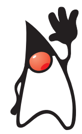
- Bytecode verifier
- Enforce language constraints at link time
- Untrusted code in a trusted environment
- Complete control over class loading
- Ability to sign classes using RSA

Robustness

- Null pointer checking
- Array bounds checking
- Garbage collector
- Exceptions
- Multi-threading
- Bytecode verification

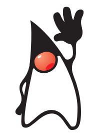
Internet Extension Language

- Robust
- Architecture Neutral
- Secure
- Fast



Status

- Alpha release available for Solaris and NT
- Port to SunOS and PowerMac underway
- Many other ports underway inside and outside Sun: Solaris x86, Windows 3.1, NextStep, SGI, Linux, ...
- Free for use on the Internet, restrictions on commercial redistribution
- Source available
- Beta release this summer



Questions