MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall Semester, 1998

Lecture Notes, December 3 – Java

Java: From a Marketing Perspective

Java is a "fundamentally new way of computing." "It works everywhere."

Java: From a Legal Perspective

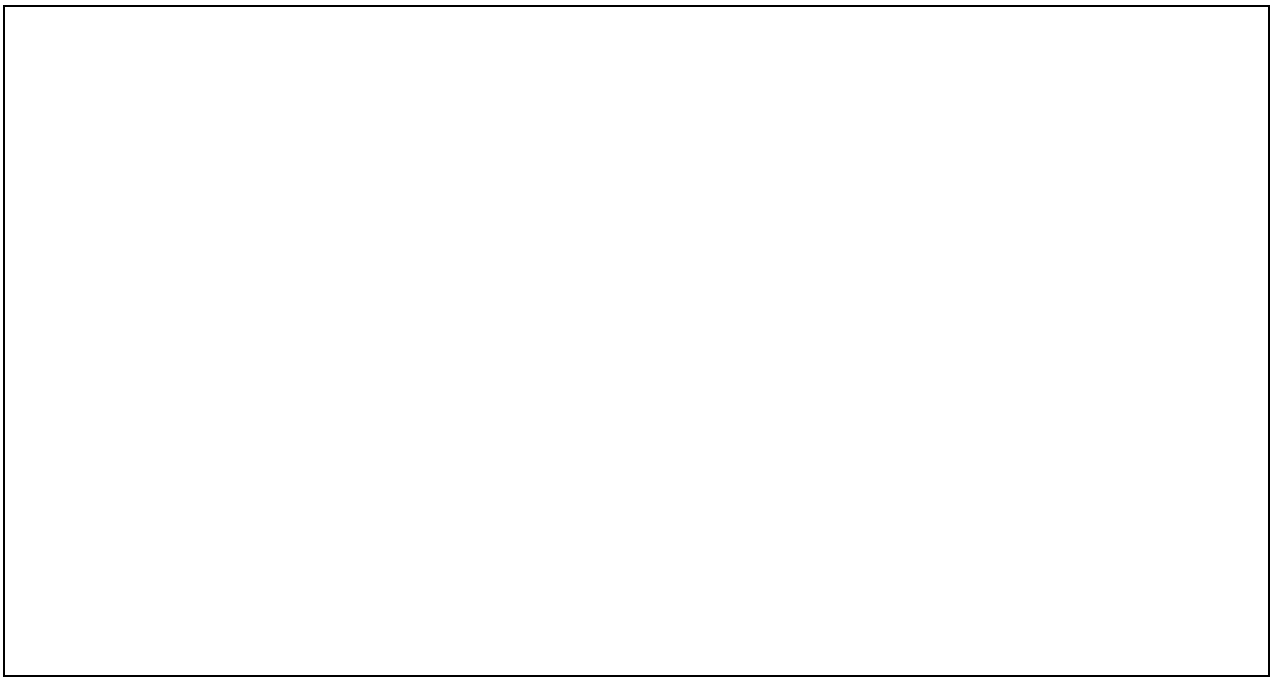
From court order in preliminary injunction, Sun v. Microsoft, Nov. 18, 1999:

"Sun's Java Technology is a collection of programming components that create a standard, platform-independent programming and runtime environment. Sun's Java Technology has two basic elements: the Java programming environment and the Java runtime environment.

"The Java programming environment allows software developers to create a single version of program code that is capable of running on any platform which possesses a compatible implementation of the Java runtime environment. The Java programming environment comprises (1) Sun's specification for the Java language, (2) Sun's specification for the Java class libraries and (3) the Java compiler.

"The Java runtime environment comprises the Java class libraries and the Java runtime interpreter. A system platform or browser program that implements the Java runtime environment can execute application programs developed using the Java programming environment. "

Java: Architecture Perspective



Jave code:

```
void spin() {
    int i;
    for (i = 0; i < 100; i++) {
        ;        // Loop body is empty
    }
}
```

Java VM instructions:

```
Method void spin()
 0 iconst_0     //Push int constant 0
 1 istore_1     //Store into local 1 (i=0)
 2 goto 8       //First time don't increment
 5 iinc 1 1     //Increment local 1 by 1 (i++)
 8 iload_1      //Push local 1 (i)
 9 bipush 100   //Push int constant (100)
11 if_icmplt 5  //Compare, loop if < (i<100)
14 return       //Return void when done
```

Java Bytecodes:

```
3 60 167 0 8 132 1 1 27 16 100 161 0 5 177
```

Java: From a Language Design Perspective

Java design goals, as summarized in "The Java Language: A White Paper," Sun Microsystems:[1]

"Java: a simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language."

- Primitives:
  - Primitive Data
    - booleans: true, false
    - integers: byte (8 bits), short (16 bits), int (32 bits), long (64 bits)
    - floating point: float (32 bits), double (64 bits)
  - Objects
- Means of Combination:
  - Arrays, e.g. int xarr[] = 1, 2, 3;
  - Method invocation
- Means of Abstraction
  - Variables, e.g. int x = 27;
  - Classes
  - Interfaces
  - Packages

[1]Material drawn from "The Java Programming Language," Ken Arnold and James Gosling, Addison-Wesley, 1996; "The Java Virtual Machine Specification," Tim Lindholm and Frank Yellin, Addison-Wesley, 1997; "Java in a Nutshell," David Flanagan, O'Reilly, 1996; and web materials at javasoft.com.

Java Class Model

A Class defines instance variables, constructors (for instance creation), methods (invoked on instances), class variables and class methods. The model provides for method Invocation, object references (where primitive data are pass-by-value and objects are pass-by-reference), and single inheritance.

Class Example 1: Class as Data Structure

```
class Body {
    public long idNum;       //instance vars
    public String nameFor;
    public Body orbits;
    public static long nextID = 0;  //class var
}

// Object (Variable) Declaration:
  Body mercury;     // declare but no creation

// Object Creation:
  Body sun = new Body();
  sun.idNum = Body.nextID++;
  sun.nameFor = "Sol";
  sun.orbits = null;

  Body earth = new Body();
  earth.idNum = Body.nextID++;
  earth.nameFor = "Earth";
  earth.orbits = sun;
```

Class Example 2: With Constructors, Instance Methods, and Class Methods

```
class Body {
    public long idNum;   //instance vars
    public String nameFor;
    public Body orbits;
    public long id() { return idNum; }
    private static long nextID = 0; //class var
    public long numBodies() {
      return nextID;
      }
    Body() { idNum = nextID++; }
    Body(String bodyName, Body orbits) {
        this(); //explicit construct invocation
        name = bodyName;
        this.orbits = orbist;
    }
}

  Body sun = new Body();
  sun.nameFor = "Sol";
```

```
  sun.orbits = null;
  Body earth = new Body("Earth", sun);
  System.out.println(earth.nameFor + " is body number" + earth.id() +
    " out of " + Body.numBodies());
==> Earth is body number 1 out of 2 bodies
```

Object References

```
  Body sun = new Body("Sol", null);
  Body earth = new Body("Earth", sun);
  Body home = earth;
  home.nameFor = "Home Sweet Home";

  System.out.println("Earth is " +  earth.nameFor);
==> Earth is Home Sweet Home
```

Garbage Collection

```
  Body sun = new Body("Sol", null);
  Body earth = new Body("Earth", sun);
  sun = new Body("Center of Universe", null);
  earth = new Body("FlatEarth", sun);
```

When are the old earth and sun objects garbage?

Garbage Collection and Finalization

Finalization provides an opportunity to perform user-defined "clean up" operations just before an object is garbage collected:

```
public class ProcessFile {
    private Stream file;
    public ProcessFile(String path) {
        File = new Stream(path);
    }
    // ...
    public void close() {
        if (File != null) {
            File.close();
            File = null;
        }
    }
    protected void finalize() throws Throwable
        super.finalize();
        close();
    }
}
```

Inheritance

```
class Moon extends Body {
  public String phase;
  Moon(String name, Body orbits, String phase) {
    super(name, orbits);
    this.phase = phase;
    orbits.addMoon(this);
  }
}
class Planet extends Body {
  final int MAX_MOONS = 10;
  public Moon[] moons = new Moon[MAX_MOONS];
  public int numMoon = 0;
  public void addMoon(Moon m) {
    moons[numMoon++] = m;
  }
}
Body sun = new Body("Sol", null);
Planet earth = new Planet("Earth", sun);
Moon luna = new Moon("Luna", earth, "Full");
```

Java "Interfaces"

```
public interface Drawable {
  public void setPosition(double x, double y);
  public void draw(DrawWindow dw);
  }
class Body implements Drawable {
    private double x_pos = 0.0, y_pos = 0.0;
    public long idNum;   //instance vars
    public String nameFor;
    public Body orbits;
    ...
    public void setPosition(double x, double y) {
      x_pos = x; y_pos = y;
    }
    public void draw(DrawWindow dw) {
      dw.drawCircle(x_pos, y_pos, 1.0);
    }
    ...
}
```

Java Packages

The Package is a means to scope names and provide/limit access to classes and methods within/outside the package. Naming provides for universally(!) unique names for classes and methods. One can use "shortened" names for classes within a package, or from "imported" packages:

```
import package;
import package.class;
```

```
import package.*;

import java.lang.*;   //implicit
```

Various rules apply about access to packages, classes, and fields within classes

- public classes are accessible within another package
- private fields in a class only accessible to methods within that class
- private protected fields in a class accessible to that class and subclasses of that class

Exception Handling

Exceptions are *thrown* by the system or by the programmer when an unexpected error condition is encountered. The exception is *caught* by surrounding code somewhere up the (dynamic) call chain designed to deal with the exception. Java exceptions are objects (instances of java.lang.Error or java.lang.Exception), and the programmer can add their own exceptions and exception handling.

```
boolean searchFor(String file, String word)
              throws StreamException
{
    Stream input = null;
    try {
        input = new Stream(file);
        while (!input.eof())
            if (input.next() == word)
                return true;
        return false;  // word not found
    } catch (FileNotFoundExcept e) {
        // Don't panic if file doesn't exist..
        return false;
    } finally {
        if (input != null)
            input.close();
    }
}
```

Threads

A thread is a process which can run concurrently with other threads. Synchronization is provided by locking on shared objects:

```
Account A;
...
synchronized (A) {  // blocks if object A is locked
  double a1 = A.getBalance();
  a1 += 200.0;
  A.setBalance(a1);
  }
```

```
synchronized (A) {  // blocks if object A is locked
  double a2 = A.getBalance();
  a2 += deposit;
  A.setBalance(a2);
  }
```

One can also declare synchronized methods:

```
class Account {
  private double balance;
  public Account(double initialDeposit) {
    balance = initialDeposit;
  }
  public synchronized double getBalance() {
    return balance;
  }
  public synchronized double deposit(double amount) {
    balance += amount;
  }
}
```

Mechanisms (wait and notify) are also provided to negotiate control and communicate between threads:

```
class Queue {
  Element head, tail;

  public synchronized void append(Element p) {
    if (tail == null)
      head = p;
    else
      tail.next = p;
    p.next = null;
    tail = p;
    notify(); // Let waiters know of arrival
  }

  public synchronized Element get() {
    while(head == null)
      wait(); // Wait for an element
    Element p = head; // Remember first element
    head = head.next; // Remove it from queue
    if (head == null) // Check for an empty Q
      tail = null;
    return p;
  }
}
```