Introduction to Java

CS587X Lecture 4
Fall, 2002

Properties

- Object Oriented
- Garbage Collected
- Portable
- Multi-threaded
- Secure
- Internet-aware

Object Oriented

- Similar to C++, Visual Basic, Smalltalk
- Think in terms of operations (methods) on data (objects)
  - E.g., write (method) to a socket (object)
- Assists good design and implementation principles
  - Good organization
  - Data encapsulation
  - Information hiding
  - Abstraction

Garbage Collection

- Automatically reclaims memory that is no longer used
  - When references to allocated memory go out of scope
  - Or, when a reference to allocated memory is removed
- Easier for programmers to manage
- No memory leaks
- However, garbage collector can run at any time

Portability

- Java is O/S and hardware independent
  - Runs on UNIX, Windows, Mac OS
- Virtual Machine executes Java byte code
  - Abstraction over hardware
  - Performance issues
- Useful for user interfaces
  - E.g. Web applets

Threads

- Java includes syntax support for threading
  - Multiple threads of execution in single program for concurrent processing
  - Shared memory space
- May improve performance
  - Multiple clients from single server

Security

- Bounds checked
- No pointers to arbitrary memory
- Security model: sandbox
  - Limits a program's access to computer resources
  - Prevents unallowed activity outside the sandbox
- Example: Web applets
  - Can only make network connections to server from which the applet was retrieved
  - Can not access files
  - Can not start processes

Internet Awareness

- Rich support in the standard API for Internet
  - IP addresses
  - UDP packets
  - TCP streams
  - HTTP
  - Multicast
- Remote Method Invocation
- CORBA

Java Applications

- Hello World program:

```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        //Display the string.
        System.out.println("Hello World!");
    }
}
```

- C-like comments
- “class” begins a class definition block
- “public static void main(String[] args)”
  - Externally-visible, available without having to create an instance of the object, returns “void”, and is given the array of Strings from the command line

Exceptions

- Enforced handling of errors & unusual conditions
  - Unlike C, where errors can be completely ignored
  - Easier than C, where nearly every function call return value should be checked for an error respose
- Methods that can throw exceptions are marked with a “throws” clause

Exception Handling

- Implemented as try/catch/finally blocks
  - Statements in the try block are executed until an exception is thrown
  - If an exception is thrown, control jumps to the catch block most-specific to the exception
  - After the try block and any catch block, the finally block is executed

System Properties

- Like environment variables
- Can be set by command line
  - -Dpropname=value
- or by programs
  - System.setProperty(“propname”, “value”);
- Getting value
  - String value = System.getProperty(“propname”);

Packages

- Encapsulate class name spaces
- Most-used packages: java.net.*, java.io.*
- If no clashes between class names, can just use the short name
  - E.g., InetAddress instead of java.net.InetAddress

Compiling & Running

- javac ObjectName.java
  - Compiles ObjectName.java into ObjectName.class (byte code); compiles dependencies as well, if needed
  - Use the '-g' option to prepare for debugging
- java ObjectName
  - Executes the static method `main` in the class ObjectName
  - Depends on correct setting of CLASSPATH environment variable!
    - I had to “unsetenv CLASSPATH” before I could run a Java program on popeye.cs.iastate.edu

Online Documentation

- Documentation root:

http://java.sun.com/j2se/1.4/docs/index.html

- Tools (javac, java, jdb) docs:

http://java.sun.com/j2se/1.4/docs/tooldocs/tools.html#basic

- API:

http://java.sun.com/j2se/1.4/docs/api/index.html

Sample App: Producer

```
public class Producer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Producer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            cubbyhole.put(i);
            System.out.println("Producer #" + this.number
                               + " put: " + i);
            try {
                sleep((int)(Math.random() * 100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Sample App: Consumer

```
public class Consumer extends Thread {
    private CubbyHole cubbyhole;
    private int number;

    public Consumer(CubbyHole c, int number) {
        cubbyhole = c;
        this.number = number;
    }

    public void run() {
        int value = 0;
        for (int i = 0; i < 10; i++) {
            value = cubbyhole.get();
            System.out.println("Consumer #" + this.number
                                + " got: " + value);
        }
    }
}
```

Sample App: CubbyHole

```
public class CubbyHole {
    private int contents;
    private boolean available = false;
    public synchronized int get() {
        while (available == false) {
            try { wait(); } catch (InterruptedException e) { }
        }
        available = false;
        notifyAll();
        return contents;
    }
    public synchronized void put(int value) {
        while (available == true) {
            try { wait(); } catch (InterruptedException e) { }
        }
        contents = value;
        available = true;
        notifyAll();
    }
}
```

Sample App: Main

```
public class ProducerConsumerTest {
    public static void main(String[] args) {
        CubbyHole c = new CubbyHole();
        Producer p1 = new Producer(c, 1);
        Consumer c1 = new Consumer(c, 1);

        p1.start();
        c1.start();
    }
}
```