Java white paper description

- simple
- object-oriented
- distributed
- interpreted
- robust
- secure

Java white paper description, cont'd

- architecture neutral
- portable
- high-performance
- multithreaded
- dynamic

Similarities with C++

- Syntax
- primitive data types (except that char is Unicode)
- control flow constructs and operators
- // and /*...*/ for comments
  /**...*/ for javadoc comments
Differences from C++

- not just a language, but an entire execution environment

- has:
  
  separate interface and implementation inheritance, threads, exception handling

- does not have:
  
  separate class declaration, pointers, globals, structures, unions, enumerated types, typedefs, templates, operator overloading

- Single implementation inheritance
  
  multiple interface inheritance

References (object and array)

- object and array variables are references
  
  - no & or *
  
  - no pointer arithmetic or sizeof
  
  - field access is always via "."
  
  - assignment: use clone() instead of =
  
  - equality: use equals() instead of ==

- a variable equal to null doesn’t refer to any object

- object and array parameters passed by reference, e.g.,

  public static void doublebuffer (StringBuffer s) {
      s.append (s); // side-effect: modifies s
  }

Differences from C++, cont’d

- no preprocessor

  no macros, no #define’d constants, no #include

- methods are not explicitly declared virtual

  public and non-final methods are virtual

- abstract instead of = 0 for pure virtual methods

- limited access to underlying system (through System Properties)

  no getenv()

Java memory management

- must dynamically allocate instances via operator new

  except of primitive (not object or array) types

- no allocation off stack

- no explicit deallocation
  
  - all objects are reference counted
  
  - when its reference count goes to 0, garbage collector can deallocate an object

- can manually invoke garbage collector (System.gc())
Defining Java classes

```java
/*
 * Example class with both implementation and interface inheritance.
 * @author Wild Hacker
 * @version 0.1
 */
public class Foo extends Bar implements Baz {
    /*
     * constructor: accesses private instance member
     * @param x argument of primitive type is passed by value
     */
    public Foo (int x) {
        super (x); // calls Bar constructor
        x = foo_int; // no side-effect
    }

    /*
     * static method: does not access any instance fields
     * @param s argument of object type is passed by reference
     */
    public static void doubleBuffer (StringBuffer s) {
        s.append (s); // side-effect
    }

    // the class instance is allocated at class load time . . .
    private static Foo foo_int = new Foo ();
    private int foo_int_ = 21;
}
```

Application Programming Interface (API)

- **package lang**
  - Object, Class, Thread, Math
  - System, Runtime, Process
  - Throwable (Exceptions, Errors)
  - shadows of primitive data types
  - String and StringBuffer

- **package io**
  - streams and files

- **package net**
  - sockets, URLs, Internet addresses

java.io package

- System.out.println ("hello");
- **file output**
  - FileOutputStream
  - PrintStream

- **file input**
  - FileInputStream
  - DataInputStream

- **layer streams** (System.in is an InputStream)
  DataInputStream in = new DataInputStream (System.in);
  String input = in.readLine();
java.Lang.Thread class

• states
  new, runnable, blocked, dead

• java.lang.thread package

• always construct with String name

• always call start()
  in turn calls run()

Thread synchronization

• synchronized keyword
  – synchronized method
    monitor grants thread exclusive access for invoking object
  – synchronized class (static) method
    monitor grants thread exclusive access for all class static objects
  – synchronized statement
    critical section around object or array

• java.lang.object methods
  – wait()
  – timed wait()
    no indication of timeout
  – notify() or notifyAll()

Thread scheduling

• scheduling is implementation dependent
  – cooperative on Solaris 2.x
  – time-sliced on Windows

• priorities
  – MIN_PRIORITY = 1
  – NORM_PRIORITY = 5
  – MAX_PRIORITY = 10

  ThreadLister utility

Thread groups

• a Thread is always constructed in a ThreadGroup

  • can specify other than the (default) main Thread

  • can perform operations on all Threads in a ThreadGroup
    daemonize, suspend, resume, stop
Example: Debate

```
import java.util.Hashtable;

/**
 * example threaded program
 */
public class Debate {
  /**
   * entry point
   * @param argv currently unused
   */
  public static void main(String[] argv) {
    Moderator moderator = Moderator.getModerator();
    moderator.addDebater("Clinton", "ten million new jobs");
    moderator.addDebater("Dole", "liberal!");
    moderator.start();
  }
}
```

Example: Moderator Thread

```
import java.util.Enumeration;
import java.util.Hashtable;
import java.io.*;

/**
 * Debate moderator
 */
public class Moderator extends Thread {
  private Moderator(String name) {
    super(name);
  }

  /**
   * accessor for the Singleton moderator
   */
  public static Moderator getModerator() {
    return _theModerator;
  }

  /**
   * add a debater
   * @param name the debater’s name
   * @param toSay the debater’s (scripted) text
   */
  public void addDebater(String name, String toSay) {
    Redemocrat debater = new Redemocrat(name, toSay);
    _debaters.put("Mr. " + name, debater);
  }

  static private Moderator _theModerator = new Moderator("Lehrer");
  static private Hashtable _debaters = new Hashtable();
}
```

Example: Debater

```
import java.io.*;

public class Redemocrat extends Thread {
  public Redemocrat(String name, String toSay) {
    super(name);
    setDaemon(true); // don’t wait for me when debate ends!
    _sez = toSay;
  }

  public void run() {
    while (true) {
      waitForModerator();
      System.out.println(getName() + "": ": " + _sez);
    }
  }

  private synchronized void waitForModerator() {
    notify();
  }

  private synchronized void go() {
    try {
      wait();
    } catch (InterruptedException interruption) {
      // ignore interruption
    }
  }

  private String _sez;
}
```
Exceptions and Errors

- Exceptions must be caught or thrown
- Errors and RuntimeExceptions need not be handled: they get passed up the call stack
- All have getMessage() method for retrieving message String
- All have printStackTrace() method

java.net package

- Provides passive ServerSocket and active Socket classes
- Transparent hostname resolution
- java.io streams can be layered on top of socket’s InputStream and OutputStreams
- All socket operations are blocking
- There is no select()

Exceptions and Errors, cont’d

- Example of user-defined Exception:

```java
public class InvalidDebaterException extends Exception {
    /**
     * Constructor.
     *
     * @param debater the name of the invalid debater
     */
    public InvalidDebaterException (String debater) {
        super (debater);
    }
}
```

Idioms

- Can have main() for each class, for testing
- toString() method permits implicit conversion
- String, Vector, Hashtable
- Threads
  - Avoid sleep() to avoid polling
  - Avoid priorities to not depend on scheduler
  - Match every wait() with a notify()
- To pass primitive types by reference, put into array and pass that
Performance

- declare classes or methods **final** to enable inlining

- use StringBuffer for Strings that need to be modified

- classes are loaded dynamically on demand: timing should be measured after all classes are loaded

- just-in-time compilers

- **native** methods are supported

Using Java

- only one (public) class per file

- `javac foo.java`

- `java foo`
  - entry point in class foo:
    ```java
    public static void main (String[] argv) {...}
    ```

- disassembler: `javap -c foo.class`

- debugger: `jdb foo`

CLASSPATH

- environment variable or command line argument

- contains any number of directories
  - colon separated on UNIX, semicolon on Windows

- directories searched for root of a package hierarchy
  
  if package **EDU.wustl.cs.544.user.utils** is rooted at `/home/user/classes/cs544/java`, then `CLASSPATH` should contain:
  ```
  :./home/user/classes/cs544/java
  ```

Packages

- name with **package** statement

- naming convention: **EDU.wustl.cs.544.<...>**

- access other packages with **import** statement
  ```
  import EDU.wustl.cs.544.user.utils::*;
  ```
Javadoc

- for documenting public classes, methods, and data members
  - package listing
  - package tree
  - name index
  - detailed descriptions

- `javadoc [-d directory] package/file names`

- start javadoc comment: /***

- end javadoc comment: */

- can contain most HTML tags and special tags

Javadoc special tags

- `@version` (does not seem to work?)

- `@author` (does not seem to work?)

- `@see` classname

- `@param` name description

- `@exception` name description

- `@return` description

Limitations

- Thread interruption not supported by current implementations

- ThreadDeath doesn’t appear to always be caught

- ThreadGroups may be leaked

- on our systems, /usr/bin must be ahead of /pkg/gnu/bin in path

Resources

- http://java.sun.com
  - language reference
  - virtual machine reference

- http://www.cs.wustl.edu/~schmidt/cs544/