

Java 1

Java Online

<http://java.sun.com/docs/index.htm>

1

the front of the site is somewhat cluttered with Sun marketing stuff, but the useful content is there too. Check out the tutorials at

<http://java.sun.com/docs/books/tutorial/index.html>. The "API" section is a reference for all of the library classes :

<http://java.sun.com/products/jdk/1.2/docs/api/index.html>

<http://www.afu.com/javafaq.html>

The comp.lang.java FAQ -- maintained by Peter van der Linden -- great!

<http://www.bruceeckel.com/>

Bruce Eckel's Book, Thinking in Java, in a free online form

<http://www.javaworld.com/>

lots of Java articles

Java Qualities

Interpreted

Slow

Interpreters are slower than compiled code, but they enable true portability

Possible technical fixes -- Just In Time compilers

The Sun Hot Spot dynamic compiler project

Portable

Runs the same everywhere without even a re-compile. Very true for the core language. Less true for the GUI parts such as AWT, although Swing has mostly fixed this.

Robust/Safe

Hard to crash, hard to write viruses

C/C++ Syntax

Fool the C/C++ programmers into thinking this is not much of a change

OOP

The cool way to structure code for re-use

"Blessed" libraries

Common classes: string, collection, ...
GUI libs (AWT, JFC)

Threaded

Supports multiple threads of control in the language

Dynamic

Types/behaviors are determined at run-time (most flexible, at the cost of some efficiency).

Automatic garbage collector takes care of memory management.

**Point: Inefficient + portable +
programmer efficiency features**

Programmer Efficiency

Faster Development

Building an application in Java takes about 30% less time than in C or C++

Libraries

Code re-use at last -- String, ArrayList, ... (C++ can also do this to an extent)

Memory errors

Which you no longer get -- I suspect the lack of memory errors accounts for much of the increased programmer productivity.

Procedural vs. OOP

Class

State + behavior

Objects

Object/Instance

Stores/represents state

Of some class

Object always remember their class

Java objects are always heap allocated with a call to "new", and always referred to by pointers

Constructor

Special code in the class which initializes new instances

Message

Sent to Object -- Request

Receiver

The object receiving the message.

obj.var++ NO

The receiver should operate on its state, not the client

obj.Increment() YES

Send a message to the obj, maps to a method (below), that method is code that actually changes the receiver state.

Student Example

units

getUnits(), setUnits()

getStress()

Example Client Side Code

```
// Make two students
Student a = new Student(10);
Student b = new Student(12);

// They respond to getUnits() and getStress()
System.out.println("a units:" + a.getUnits() +
    " stress:" + a.getStress());

System.out.println("b units:" + b.getUnits() +
    " stress:" + b.getStress());

a.setUnits(a.getUnits() - 3); // drop that class!

System.out.println("a units:" + a.getUnits() +
    " stress:" + a.getStress());

// Now "b" points to the same object as "a"
b = a;
b.setUnits(10);
```

```
// So the "a" units have been changed
System.out.println("a units:" + a.getUnits() +
    " stress:" + a.getStress());

/*
OUTPUT...
a units:10 stress:100
b units:12 stress:120
a units:7 stress:70
a units:10 stress:100
*/
```

Objects

Java heap objects

Always allocated with a call to "new" and referred to by a pointer
Objects and arrays are always referred to by pointers (shallow)

Primitives not in the heap

int, float, char (two byte)... not allocated in the heap

Simple "deep" semantics like C/C++

There are shadow classes like Integer and Float which can represent a single value of the corresponding primitive if you need that

Student a;

Declares the "a" variable to refer to a Student type object. Does not allocate the Student object yet -- that only happens when "new" is called.

The old "allocating the pointer does not allocate the pointee" rule.

Constructor

Special code called when "new" creates the object

Messages

obj.message();

"Receiver"

The object receiving the message.

a.getUnits() vs. b.getUnits()

The receiver getting the message is the one getting accessed or changed

Message-Method lookup

Lookup the method based on the RT class of receiver (as above)

Method executes against receiver