CS 66-460
Introduction to Graphical Human Machine Interfaces

Java (Part 1)

MONDAY,MARCH 31, 1997

Instructor: G. Bowden Wise

Rensselaer Polytechnic Institute
Spring 1997

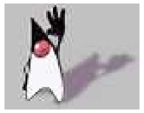
Outline

Java Beginnings ...

Java began (in 1990) as ...

- a vision to get away from "multi-API" programming;
- a venture to develop software for the "consumer electronics" market;

  but also for a variety of networked deviecs and other embedded systems
- a novel idea of a small research group of top developers at Sun, code named the "Green" team;
- a new language, called "Oak", because none of the existing OO languages — C++, Cedar/Mesa, Eiffel, Smalltalk, and Objective-C — were sufficient
  - hand-held remote control: the `*7`
  - engaging interface: "Duke" was born



Java Beginnings ... (continued)

- a foundation for a new product:
  - the "Green" team became a company: **First Person**
  - set-top boxes for video-on-demand ... due to corprorate politcs
  - in `1993` the web had become very popular;
  - the WebRunner browser is created in Oak;

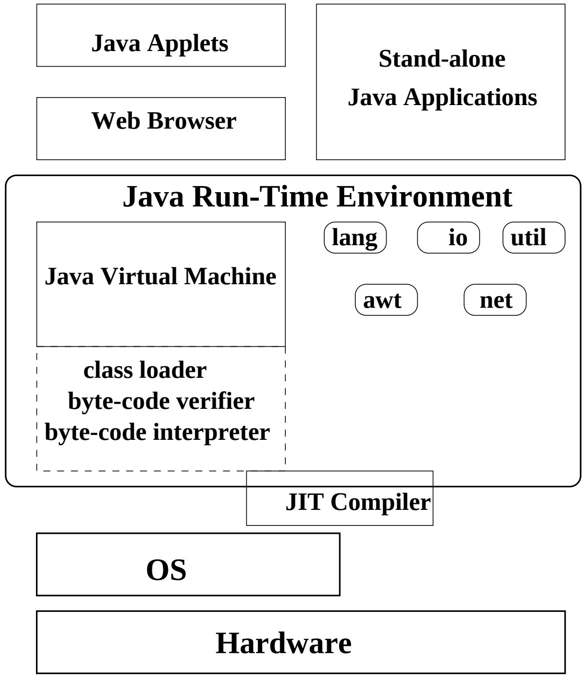
- Sun releases a new computing paradigm (`1994`):
  - renames Oak to **Java**
  - renames WebRunner to **HotJava**

Java

- Not just for creating animated/engaging Web pages;
- But a new environment for developing applications that are:
  - portable
  - robust
  - high performance
  - adaptive
  - secure
- more importantly
  - **write once, run anywhere** model
- ideal for developing
  - secure, distributed, network based end-user applications

  in many different environments
  - networked-embedded deviecs
  - World-Wide Web
  - the desktop

The Java Platform/Run-Time



Java Design Goals

- **Simple**
  so that it is easily programmed
- **object-oriented**
  to take advantage of modern software development methodologies;
  fit into distributed client-server applications
- **familiar**
  so that developers can learn quickly and easily
- **Robust and secure**
  for high reliability
- **Architecture neutral, portable, and dynamically adapable**
  to survive in a multiplatform jungle
- **High performance**
  necessary for highly interactive applications
- **Multithreaded**
  to support multiple threads in a single application;
  higher performance;
- **Interpreted**
  for maximum portability and dynamic capabilities

Java: Simple and Familiar

- **Simple**
  easy to use by developers
- **Familiar**
  so that developers can learn quickly and easily

How is this achived?

- Java looks like C++
- Removal of many "features" from C++ (and its predecessors):
  - to keep the language small
  - to make programming easiser
  - to make applications more secure and robust

Hello World!

As a Java program:

```
class HelloWorld {
   static public void main (String args[]) {
      System.out.println ("Hello world!");
   }
}
```

As a Java applet:

```
import java.applet.Applet;
import java.awt.Graphics;

public class HelloWorld extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello world!", 50, 25);
    }
}
```


Java: Primitive Data Types

- Integer types:
  `byte` (8-bit)
  `short` (16-bit)
  `int` (32-bit)
  `long` (64-bit)
- no `unsigned` integer types;
- Real types (as defined by the IEEE 754 spec):
  `float` (32-bit)
  `double` (64-bit)
- Character:
  `char` (unsigned, 16-bit Unicode)
- Boolean: `boolean` for `true` and `false`
- Arrays are first-class objects
- Strings are first class objects
  `String` (read-only)
  `StringBuffer` (read/write)

Arrays

Arrays:

- must be declared and allocated;
- do not have to be free'ed;
- have a `length()` accessor

Array example:

```
Point myPoints[];
myPoints = new Point[10];

for (int i=0; i < 10; i++) {
   myPoints[i] = new Point();
}

int howMany = myPoints.length();
```


Strings

- can be read-only

  `String hello = "Hello world";`
- or modifiable

  `StringBuffer dest = new StringBuffer(len);`

have

- `length()` accessor
- `+` operator for concatenation:

```
"Input has " + count + " chars."
```

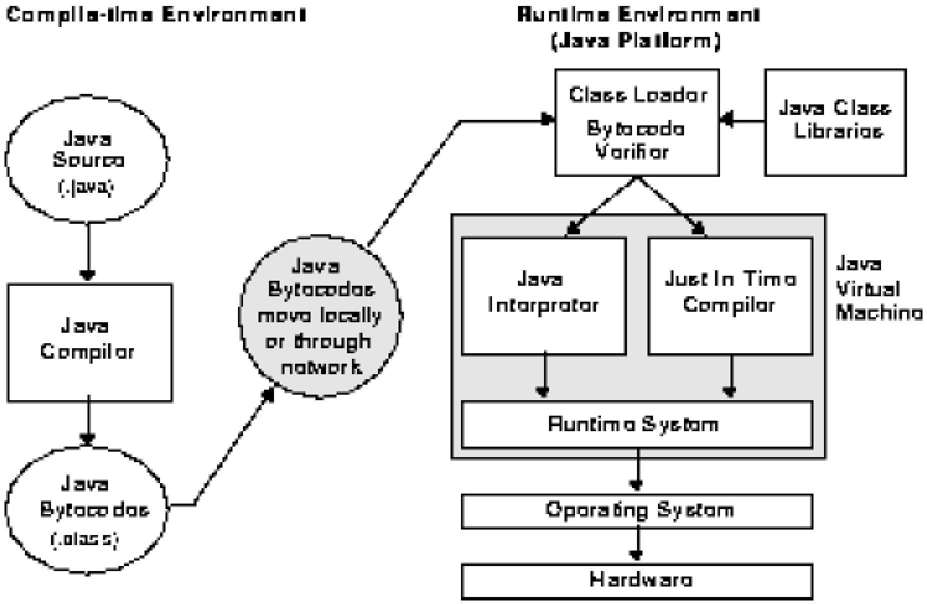
Memory Management and Garbage Collection

- Java removes memory management worries for the programmer.
- Java performs automatic garbage collection
- You use `new` to allocate memory; never call `free` (!)
- Memory model is based on objects and references to objects;
  - you get a "handle" to the object
  - java keeps track of references to objects
  - when an object has no more references, it can be garbage collected
- This model makes programming easier and eliminates many bugs

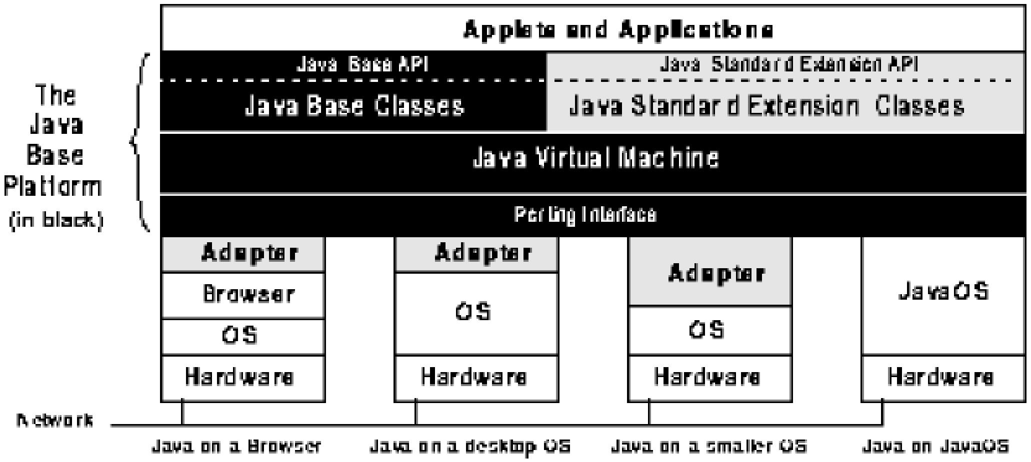
Features Removed from C/C++

- No more `typedef`s, `#defines`, or preprocessor
- No more `struct`s or `union`s
- No more (global) functions
- No more multiple inheritance
- No more `goto` statements
- No more operator overloading
- No more automatic coercions
- No more pointers

Java Compile and Runtime Environments



The Java Platform



The Bytecode Verifier

Prevent hostile applets and/or hostile Java compilers

- doesn't forge pointers to illegally manipulate objects outside the virtual machine
- doesn't violate access restrictions
- does access objects according to type
- does use correct types for all instruction parameters
- does use legal object field accesses according to their private, public, or protected definition

The Classloader

- loads built-in classes as well as those from other machines
- added security
  - each loaded class executes in its own namespace
  - single namespace for classes loaded from local file system
- when a class references another class
  - first looks for it in the local namespace
  - then in the namespace of the referencing class
  - no way for Java code to "spoof" a built-in class or a class loaded from another site
- security has been improved for JDK 1.1