V22.0490.001
Special Topics: Programming Languages

B. Mishra
New York University.

Lecture # 23

—Slide 1—

Java: History

- Spring 1990–April 1991: Naughton, Gosling and Sheridan ("The Green Team") of Sun Microsystems formulate a mission statement: "*Behind the Green Door*"

  > "To develop and license an operating environment [...] that enables services and informations to be persuasively presented via the emerging digital infrastructure."

- August 1992:
  - Gosling creates an "industrial strength" object oriented programming language named Oak (`(C++)--`).
  - Naughton designs an interface called "killer app."
  - A character named Duke (imp with a red nose) is created with "killer app" to guide the user through a cartoon house.

—Slide 2—

Java: History (Contd.)

- October 1992: Sun is ecstatic. Sun tries to market Oak.
- March 1993: Sun loses to Silicon Graphics. Sun is NOT ecstatic. The project is practically killed.
- June 1993: The Web and the Mosaic Browser find wide-spread use.
- January 1995: Oak is renamed Java. "Killer app" becomes an interpreter for a Web browser and is renamed HotJava. Java becomes famous.
- Happy Ending: Netscape buys Java. Microsft acquires Java. Jim Gosling becomes a house-hold name in the entire civilized world.

—Slide 3—

Java: Motivation

- Multiple incompatible hardware architectures.
- Multiple incompatible OS's for each architecture.
- Multiple incompatible GUI's for each OS.
- Applications must work in a distributed client-server environment.
- Want to take advantage of internet, WWW, and "electronic commerce".

—Slide 4—

Java: Properties

Java: A simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic language.
— "The Java Language: A White Paper"

- *Simple*
  –Based on `C` and `C++`
  –Many problematic features of `C++` removed
- *Object-oriented*
  –*Class*: A collection of data and methods that operate on that data
  –Data and methods describe the state and behavior of an *object*
  –Classes are arranged in a hierarchy—Subclass can inherit behavior from its superclass.

—Slide 5—

Java: Properties (Contd.)

- *Distributed*
  - –Java classes can be transmitted over the net
  - –Library routines for network I/O
- *Interpreted*
  - –Java interpreter executes the compiled byte codes
  - –Byte codes provide an architecture neutral object file format
  - –Java Virtual Machine (JVM)
- *Robust*
  - –Strongly typed
  - –Many checks performed at runtime
  - –Automatic Memory Management—Garbage Collection(GC)
  - –Exception Handling
- *Secure*
  - –No way to forge pointers
  - –Java code is verified
  - –Restricted access to file systems and network

—Slide 6—

Java: Properties (Contd.)

- *Architecture Neutral*
  - –Compiles to byte codes
  - –Easy to interpret or compile on any CPU/OS
- *Portable*
  - –Size and representation of primitive data types, defined by the language
  - –Standard library, specified
- *High-Performance*
  - –Byte codes can be compiled to native code
  - –Potentially as fast as `C`
- *Multithreaded*
  - –Built-in threads and synchronization primitives
- *Dynamic*
  - –Adapts to an evolving environment
  - –Classes are loaded in as needed
  - –Run-time class definition allows classes to be dynamically linked

—Slide 7—

Portability

- Compiler compiles down to "byte codes" to run on a "virtual machine" (JVM).
  –The Java interpreter and run-time system for a particular machine take care of interpreting the byte codes.
- Easy software distribution.
  –Need only one version of software
  –Various primitive types are built into the language—Don't depend on word size of a particular platform.
- **Fully Specified**
  - All variables are assigned before they are used.
  - All variables can refer only to objects of the correct types.
    (The run-time system keeps track of type of each object.)
  - All operands to all operators are guaranteed to have the appearance of being evaluated in left-to-right order. There are no legal expressions which have undefined behavior.

—Slide 8—

Object Oriented Programming

- *Encapsulation*
  –To implement information hiding and modularity
- *Inheritance*
  –Code re-use and code organization
- *Dynamic Loading and Binding*
  –For maximum flexibility while a program is executing
  - Classes are linked in as required and can be downloaded from across networks.
  - Can look up a class definition given a string containing its name
  - Can compute a data type name and have it easily dynamically-linked into the running system.

—Slide 9—

Automatic Garbage Collection

- Allocate an Object
  –Run-time system keeps track of the object's status
  –Automatically reclaims memory when objects are no longer used.
- Memory Manager
  –Keeps track of references to an object
  –When there are no more references to it, it is a candidate for garbage collection(GC).
- GC Runs as Low Priority Thread
  –Thus (hopefully) avoids the problem of interrupting the user to garbage collect.

—Slide 10—

Java Security

- Must be able to run "untrusted" code securely.
- No pointers in traditional sense.
- Memory layout decisions made by the run-time system—not the compiler.
- Cannot infer the physical memory layout of a class by looking at its declaration
  Cannot manufacture pointers to memory.
- These arguments are "trustable" as long as restricted to using the Java compiler.

—Slide 11—

Java Types

- **Primitive Types:** Part of language
- **Classes:** Derived from Object class
- **Interfaces:** Guarantee that methods will be provided
- **Arrays:** One array type for each of the other types (Primitive, Classes and Interfaces)
- *Values:-*
  - *Primitive Types*
    –Variables which have primitive types are always passed by value
    –The only way to change the value of a primitive type variable x is by explicitly changing the value of x.
  - *Dynamically Allocated Objects*
    –All other variables (non-primitive) are passed by reference
    –If you wish to pass one of these by value, *you must explicitly copy it*

—Slide 12—

Primitive Types

- Arithmetic
  - `byte`: 8-bit two's-complement integer
  - `short`: 16-bit two's-complement integer
  - `int`: 32-bit two's-complement integer
  - `long`: 64-bit two's-complement integer
  - `float`: 32-bit IEEE 754 floating-point numbers
  - `double`: 64-bit IEEE 754 floating-point numbers
- Characters: `char`: 16-bit Unicode characters
- Boolean: `boolean`: possible values are `true` and `false`
- Java can assign "narrower types" to "wider types"

```
byte < short < int < long < float < double
        char < int
```

—Slide 13—

Primitive Types (contd)

- Any integral type may be cast to any other arithmetic type
- Java provides all the integral operators that C++ does
- Integer division by zero throws `ArithmeticException`
- Floating point types may be cast to any other arithmetic type and `char`

—Last Slide—

Classes

- Similar to C++ classes
- All classes are derived from the class `Object`
- Instance variables and methods can be `public`, `protected`, or `private` (same definitions as C++).
- Anything not declared as `public`, `protected`, or `private` is visible throughout its `package`.
  This is the only way to get "friendly" behavior
- *Single Inheritance*
  Each class (except `Object`) has exactly one superclass.
- Initialized to `null`. Takes on value only through the = operator or the `new` operator.

[End of Lecture #23]