Java: Design Goals

A general-purpose concurrent object-oriented language for the Internet

Framework-Oriented Programming

- Familiar
  - C/C++ syntax
- Simple
- Portable
  - Platform-independent
  - Architecture-neutral
- Object-oriented
- Reliable
- Robust
- Interpreted
  - Efficient
- Secure
- Concurrent
  - Interactive
- Distributed
  - For Internet

Simplicity

- Programmer perspective
  - Automatic garbage collection
    - Avoids memory leaks.
    - Avoids dangling pointers
  - No C++ structure/union.
    - Only “pointer” to structure.
    - “Unconstrained” array type
- Implementer perspective
  - No multiple inheritance of code.
    - Only single inheritance of classes.
  - Restricted overloading.

Garbage Collection: Perspectives

- Garbage Collection is necessary for fully modular programming, to avoid introducing unnecessary intermodular dependencies.
  - “Uniprocessor Garbage Collection Techniques” by Paul R. Wilson et al
- Two of the most powerful tools available to the software engineer are abstraction and modularity. ... Automatic memory management gives increased abstraction to the programmer.
  - “Garbage Collection” by Jones and Lin
Portability

- Architecture-neutral
  - Size of integer, char, etc. and the meaning of floating point operations fixed by the language.
- All “behavioral” aspects of a Java program defined by the Java language specification, rather than left to the implementation.
  - Order of evaluation of operands fixed.
  - Error and Exception handling
    - Imposes a degree of uniformity on the outcome of running a program.

Non-portability: `gotcha.c`

```c
main() {
    int i = 5;
    printf("%d \n", i);
    printf("%d %d %d \n", i, i/++i, i);
}
```

- **Intuitively??**
- 5 0 6

```java
class PortJava {
    public static void main(String[] args) {
        int i = 5;
        System.out.println("i = " + i + " i/++i = " + i/++i + " i = " + i);
    }
}
```

- **Output:** i = 5 i/++i = 0 i = 6

```java
class PortableJava {
    public static void printOrder(int i, int j, int k) {
        System.out.print("i-> " + i);
        System.out.print("j-> " + j);
        System.out.print("k-> " + k);
    }
    public static void main(String[] args) {
        int i = 5;
        printOrder(i, i/++i, i);
    }
}
```

- **Output:** i-> 5 j-> 0 k-> 6
Possible Outcomes

- **SPARC**: `cc/gcc gotcha.c -ldl; a.out`
  - 6 1 6
- **Alpha, MIPS**: `cc/gcc gotcha.c; a.out`
  - 5 1 6
- **SUN-3**: `cc gotcha.c ; a.out`
  - 6 1 5

Object-oriented programming

- Programming with Data Types
  - Data Abstraction
    - Modularity
    - Encapsulation (information hiding)
- Reuse and Sharing
  - Inheritance
    - Reusing code
  - Polymorphism and Dynamic binding
    - Sharing behavior; Frameworks

Java Collections: **Interfaces**

```
Collection  
  Set  List
  SortedSet

Map
  SortedMap
```

Java Collections: **Implementations**

```
<table>
<thead>
<tr>
<th>Interfaces</th>
<th>Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hash Table</td>
<td>Resizable Array</td>
</tr>
<tr>
<td>Set</td>
<td>HashSet</td>
</tr>
<tr>
<td>List</td>
<td>ArrayList</td>
</tr>
<tr>
<td>Map</td>
<td>HashMap</td>
</tr>
</tbody>
</table>
```
Reliability and Robustness

- **Reliability**
  - Degree of Confidence in “error-free” execution of a program after a clean compile.
  - Strong typing

- **Robustness**
  - “Graceful degradation” behavior of a program when presented with an input that violates preconditions. (Cf. correctness)
  - Exception handling

Interpreted, Efficient, Secure

- A Java program is compiled into bytecode that is interpreted by the Java Virtual m/c.
- *Just-In-Time compilers* convert bytecode into native code for efficiency.
- JVM performs various security checks while executing bytecode.
  - JVM enforces severe access restrictions while running applets downloaded from a remote site.

Java : Compiler and Interpreter

```
source code
```

```
bytecode
```

```
native code
```

```
java
```

```
javac
```

Evolution of Sun’s JDK

- Java 1.0: Interpreter
- Java 1.1: Interpreter + JIT Compiler
- Java 2: Hotspot
  - Profiling and Adaptive Dynamic Compilation of “hot” code
  - Method in-lining and other aggressive optimizations, and Decompilation
  - Improved Memory Management for long-running (server) programs
  - Fast Thread Synchronization
**Improvements : Java vs C++**

“Java is better than C++, more because of what it does not have, than for what it has.”

- No global vars.
- No *gotos*.
  - Disciplined uses *abstracted* (exceptions, labeled breaks).
- No pointers.
  - Array-index out-of-bounds detected.
- No *explicit* memory management.
  - No memory leaks and no illegal access of freed storage.
- Improves readability.

**Concurrent Programming**

- Threads can share address space. In Java, threads are modeled after Hoare’s *monitors*.
- Java is a *multi-threaded* system, which is very convenient for coding *interactive, multi-media* applications.
  - Doing I/O *concurrently* with reading a document. (*downloading via browser*)
  - Periodic updating of the screen with *data* acquired in real-time. (*sports/stocks ticker*)

**Distributed Processing**

- WWW (URL+HTTP+HTML) contributed immensely to the sharing of (distributed) *static* data.
  - URL encodes description of a protocol, service, server name, location of a resource on the server, etc. succinctly.
  - [http://www.cs.wright.edu/~tkprasad](http://www.cs.wright.edu/~tkprasad)
  - [ftp://user@host:port/path](ftp://user@host:port/path)
  - [mailto:user@host](mailto:user@host)
- URL facilitated access to remote resources by integrating access to various protocols/services such as FTP, telnet, mail, http, etc in a browser.
- Java contributed immensely to the sharing of dynamic/executable content.

(cont’d)

- CGI scripts enabled dynamic customization of responses based on user input (using FORMS).
  - However, this also requires a centralized, powerful server to run scripts to process client requests.
- Java applets enabled moving code to a client machine in response to a client request (rather than the final data).
  - Computations distributed to client machines (providing an alternative to the traditional client-server model).
  - More responsive interaction, and emphasis on portability.
  - Required support for GUIs, network programming, virtual machine, etc.