Introduction to Java Programming Language

- Java's History
- Java's Features compared to C++

Java's Brief History

Embedded controller market is originally targeted

- Designed for programs for small embedded computers in consumer electronic appliances (TV, Microwave, etc.)
- Should be small, distributed, robust

Rapidly accepted as a de facto network programming language as the Internet grows fast (WWW, electronic commerce)

Why?

Features as a Network Programming Languages

- Run on a wide variety of hardware platforms
- Loaded dynamically via a network
- Provides robustness features
- Provides security features

Features as a General Programming Languages

- Completely object-oriented language
- Much simpler than C++
- Can work on multiple tasks simultaneously
- Automatically recycles memory
- Exception handling features
- Interpreted
- High-Performance

Portability

Designed to support applications operating in networked environment with different CPUs, OS, and language systems

- Java compiler dose not generate “machine code”
- Rather, Java is compiled into *byte code*, a high-level machine-independent intermediate code
- Byte code is indeed an instruction set for an hypothetical stack machine;e.g., **iload, istore, pop, getfield, iadd, ..**
- The size of byte code is small (2 times smaller than the RISC code from C++ compiler)

Portability (cont')

```
public class Arr {
    public static void main(String [] args){
        int[] array = new int[1000];

        for( int i=0; i<array.length;i++ )
            array[i] = i;
    }
}
```

```
Method void main
0 sipush 1000
3 newarray int
5 astore_1
6 iconst_0
7 istore_2
8 goto 18
11 aload_1
12 iload_2
13 iload_2
14 iastore
15 iinc 2 1
18 iload_2
19 aload_1
20 arraylength
21 if_icmplt 11
24 return
```

Portability (cont')

- The byte code is executed by a software called a Java *virtual machine* by *interpretation*
- Java's data types and operator behaviors are strictly defined;
  e.g., **bytes** : 8-bit two's complement, **char** : 16-bit Unicode,…
- Java libraries define portable interface
  (e.g., Abstract Window Toolkit (AWT) for UNIX, Windows, Mac)
- Java language environment is easily portable
  - Java compiler is written in Java
  - Java run-time system is written in ANSI C
- This approach is good for single-system software distribution as well as for network-based applications

Robustness

Reliable programming

- Strict compile-time checking by Java compiler
- Run-time checking by JVM (null dereference, array bound check)
- Do not corrupt memory (no pointer arithmetic, w/ garbage collection)
- Fast prototyping (like Lisp)

Security

How Java compiler restricts “hacking” code

- No pointers (memory cells that contain the address of others)
- Memory layout is decided at run-time, not at compile-time
  (memory references using handles are resolved at run-time)
- Violation of these are detected at compile-time

How Java VM restricts “hacking” code

- Byte code verification by JVM (verifier, class loader)
- Verifies no operand stack overflow, illegal data conversions, incorrect parameter types, etc.

Object-Oriented Programming in Java

Completely object-oriented language

- Programs consist of class definitions only (no stand-alone functions)
- Class definition establish the blueprint of application-specific category
- Objects (class instances) of the category can be created using the blueprint

OO Technology in Java (cont')

Dynamic loading and *binding* of classes

- No separate, static "link" phase after compilation. Load classes dynamically only when needed during execution
- Linking in Java is loading new classes into JVM by the *class loader* and is incremental and lightweight
- Dynamic binding solves the *fragile superclass* problem of C++ : when a class definition changes, all other classes that reference the class must be recompiled ; in Java, the references are compiled into symbolic names (not numeric offsets) which are then resolved by the Java interpreter (once) at run-time
- Results in highly dynamic and dynamically-extensible system ; classes are linked as required and are downloaded across networks ; you can also provide your own class loader!

Simpler than C++

- No functions
- No multiple inheritance (instead, use interface)
- No operator overloading
- No pointers
- No Typedefs, Defines, and Preprocessing
- No structures and unions

Multithreading

Multithreading is part of the Java language

- Thread : a single sequential flow of control within a program
- Concurrent activities can be programmed by running multiple threads at the same time (e.g. HotJava browser)
- Synchronization is required for accessing shared data
- Java provides primitives for threads and locks
- More “thread-safe”compared to previous thread libraries
  (e.g., what happens then exception occurs while holding a lock)

Garbage collection

Java frees you from the headache of memory management

- A background garbage collector runs in a low-priority thread
- It frees unused memory for reuse
- Significantly improves the reliability and production cycle of code

Exception Handling Features

Separate Error-Handling Code from “Regular” code

Example : reading an entire file into memory

```
readFile {
      open the file;
      determine its size;
      allocate that much memory;
      read the file into memory
      close the file;
}
```

Exception Handling Features (cont')

- What happens if the file cannot be opened?
- What happens if the length of the file cannot be determined?
- What happens if enough memory can't be allocated?
- What happens if the read fails?
- What happens if the file cannot be closed?

Exception Handling Features (cont')

```
errorcode Type readFile {
    init error_code = 0;
    open the file;
    if ( the file is opened ) {
        determine the length of the file;
        if (got the file length) {
            read file into memory
            if(read failed) {
                error_code = -1;
            }
        } else error_code = -2;
    .........
}
```

Is the file really being closed if the function fails to allocate memory?

Exception Handling Features (cont')

```
readFile    {
      try     {
                open the file;
                determine its size;
                allocate that much memory;
                read the file into memory;
                close the file;
      } catch (fileopenfailed) {
                do something
      } catch(sizedeterminefailed) {
                do something
      } catch ( .... )
}
```