CS 112 Introduction to Programming  
(Spring 2012)

Lecture #2: Java Program Structure

Zhong Shao

Department of Computer Science
Yale University
Office: 314 Watson
http://flint.cs.yale.edu/cs112

Acknowledgements: some slides used in this class are taken directly or adapted from those accompanying the two textbooks: Introduction to Programming in Java: An Interdisciplinary Approach by Robert Sedgewick and Kevin Wayne and Building Java Programs: A Back to Basics Approach by Stuart Reges and Marty Stepp.

Java Programming Language: Designers

- Bill Joy
  - BSD Unix guy from UC Berkeley
  - co-founder of Sun Microsystems (1982)
  - "the network is the computer", targeting workstation market
  - focusing on network was a little ahead of its time; missed the boat on PC revolution
  - retreated to Aspen, Colorado

- James Gosling
  - early fame as the author of "Gosling Emacs" killed by GNU emacs
  - then onto Sun's "NeWS" windows system killed by X-windows
  - lesson: keeping things proprietary is kiss of death

Java Programming Language: History

- Joy and Gosling joined force: Sun subsidiary, FirstPerson, Inc. (1992)
  - target consumer electronics: PDAs, appliances, phones, all with cheap infra-red kinds of networks
  - need a language that's safe, portable, secure, wired
    - started working on C++
    - soon gave up hope, decided to start from scratch
  - a little ahead of time (again): PDAs died with the demise of Apple Newton
  - switched to interactive TV (ITV)
    - the resulting language was called "Oak"
  - a little ahead of time (yet again): ITV died too

- Good luck (finally)
  - the net exploded in 1993
  - Oak became Java

Why Java?

- Java is an elegant, safe, object-oriented programming language
  - simpler than other object-oriented languages [e.g., C++]
  - Java is the basis of other modern programming languages [e.g., Microsoft C#]

- Java is (largely) platform independent --- write once run everywhere
  - Java supports multiple platforms (Unix, Windows, Mac), multiple types of devices (desktops, phones, embedded devices)

- Java has rich libraries and good support
  - good multimedia, graphics packages
  - good client-server and network support (applet, serverlet)
  - good, free Integrated Development Environments (IDE)

- Widely used
  - #1 in popularity: http://www.tiobe.com
Java is Still Evolving

<table>
<thead>
<tr>
<th>Version</th>
<th>Year</th>
<th>Important New Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1996</td>
<td></td>
</tr>
<tr>
<td>1.1</td>
<td>1997</td>
<td>Inner classes</td>
</tr>
<tr>
<td>1.2</td>
<td>1998</td>
<td>Swing, Collections</td>
</tr>
<tr>
<td>1.3</td>
<td>2000</td>
<td>Performance enhancements</td>
</tr>
<tr>
<td>1.4</td>
<td>2002</td>
<td>Assertions, XML</td>
</tr>
<tr>
<td>5</td>
<td>2004</td>
<td>Generic classes, enhanced for loop, auto-boxing, enumerations</td>
</tr>
<tr>
<td>6</td>
<td>2006</td>
<td>Library improvements</td>
</tr>
</tbody>
</table>

- New features added by following the Java Community Process
- Others extend Java to other settings: Google Android uses Java on mobile devices

Machine Language

- The “brain” of a computer is its Central Processing Unit (CPU)
- A CPU can understand only very basic instructions
  - e.g., store a given value at a memory location; do some arithmetic operations; compare two values; start to execute the instruction at another location
- The instruction set of a CPU forms the machine language of the CPU

- New features added by following the Java Community Process
- Others extend Java to other settings: Google Android uses Java on mobile devices

Machine Language

- A typical sequence of machine code:
  - Load the contents of memory location 40.
  - Load the value 100.
  - If the first value is greater than the second value, continue with instruction stored in memory location 240.
  - Ask the operating system to do service with code 21
    ```
    21 40
    16 100
    163 240
    12 21
    ```
- Different machines understand different machine languages (why?)
- Early programmers wrote programs in machine languages

Assembly Languages

- **Assembly language** or simply assembly is a human-readable notation for the machine language

  It’s much easier to remember:

  ```
  movl (40(4(%edx))), %eax
  movl 12(%ebp), %eax
  leal 0(%eax,%eax), %edx
  movl $nodes, %eax
  movl (%edx,%eax), %ecx
  movl 12(%ebp), %eax
  leal 0(%eax,%eax), %edx
  movl $nodes, %eax
  movl (%edx,%eax), %eax
  ```

  Example assembly code fragment
High-Level Programming Languages

- A high-level programming language enables a programmer to specify, in a high level (close to natural language), what data a computer will act upon, how these data will be stored, and what actions to take under various circumstances.

- The syntax and grammar of a high-level language is independent of CPU.

```java
// Example Higher-level Source Code fragment

// Initialize values
int celsiusTemperature = 32;

// Convert to Fahrenheit
int fahrenheitTemperature = celsiusTemperature * 9 / 5 + 32;

// Check if it's hot
if (fahrenheitTemperature > 100)
    System.out.println("Hot!");
else
    System.out.println("OK!");
```

Some Major Types of High-Level Languages

- **Procedural languages**: programs are a series of commands
  - Pascal (1970): designed for education
  - C (1972): low-level operating systems and device drivers

- **Functional programming**: functions map inputs to outputs

- **Object-oriented languages**: programs use interacting "objects"
  - Smalltalk (1980): first major object-oriented language
  - C++ (1985): "object-oriented" improvements to C
  - Java (1995): designed for embedded systems, web apps/servers
    - Runs on many platforms (Windows, Mac, Linux, cell phones...)
    - The language taught in this textbook

High-Level Language Influence

Problem

- **Language barrier**
  - Computers: understand machine platform languages---to build efficient hardware
  - Programmers: want more readable high-level languages---to be more productive
Hire a Translator: Compiler

- A program written in a high-level language must be translated into the language of a particular platform (type of CPU and operating system) before execution.

- A compiler is a program which translates source code into a specific target platform (CPU + OS).

![Diagram of the translation process from source code to machine code for intel x86 + Win7.]

Problems of Compiling to Each Specific Computer Platform

- Multiple versions of the same software.

Java Virtual Machine

- To be platform independent, Java designers introduced Java Virtual Machine (JVM), a machine different from any physical platform, but a virtual machine.
  - The language of the virtual machine is referred to as bytecode.
  - Thus Java actually has two programming languages.

- A Java compiler translates Java source code (.java files) into bytecode (in .class files).
  - Each Java software program needs to be compiled only once: from the Java source code to bytecode.

Java Execution

- To execute a Java program, another piece of software called an interpreter, translates between bytecode and the actual machine.
  - An interpreter is specific to a specific platform.
  - The interpreter understands java bytecode, and then issues instructions in the specific platform for which it is written.
  - We also say that an interpreter provides a Java virtual machine (JVM).
Java Translation and Execution

- Java source code
- Java compiler
- Java bytecode
  - bytecode interpreter for Windows
  - bytecode interpreter for Droid
  - bytecode interpreter for Mac
  - bytecode interpreter for Linux

Comparing Traditional (e.g., C/C++) and Java Software Development

**Traditional, e.g., C/C++**
- A developer writes a program in C/C++
- The C/C++ source code is generally considered proprietary, and not released
- The developer compiles the C/C++ program for each platform it intends to support, and distributes one version for each platform
  - thus each program has multiple compiled versions
  - each compiled version can run by itself
- Platform dependency handled by each software developer

**Java**
- A developer writes a program in Java
- The Java source code is generally considered proprietary, and not released
- The developer compiles the Java program to bytecode, and distributes the bytecode version
  - thus each program has only one compiled version
  - the compiled bytecode needs an interpreter for each platform
- Platform dependency handled by platform vendor

High-level Picture

- **C/C++**
  - Prog 1
    - Prog 1; Arch 1
      - Prog 1; Arch n
    - Prog 2; Arch 1
      - Prog 2; Arch n

- **Java**
  - Prog 1
    - Prog 1/bytecode
      - Interp; Arch 1
        - Interp; Arch n
    - Prog 2
      - Prog 2/bytecode

Recall: Java Programming Steps

- Programming in Java consists of three tasks
  - edit java source code (.java files)
  - compile java source code to generate bytecode (.class files)
  - execute/run/test bytecode using an interpreter
Programming in Java (Step 1): Create/Edit

- Create/Edit the program by typing it into a text editor, and save it as HelloWorld.java.

```java
/* Prints "Hello, World" */
* Everyone's first Java program.
*
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, World");
    }
}
```

HelloWorld.java

Programming in Java (Step 2): Compile

- Create the program by typing it into a text editor, and save it as HelloWorld.java.
- Compile it by typing at the command-line:
  - javac HelloWorld.java

  ```
  % javac HelloWorld.java
  (or click the Compile button in DrJava)
  ```

- This creates a Java bytecode file named: HelloWorld.class.

Programming in Java (Step 3): Execute

- Create the program by typing it into a text editor, and save it as HelloWorld.java.
- Compile it by typing at the command-line:
  - javac HelloWorld.java
- Execute it by typing at the command-line:
  - java HelloWorld

  ```
  % javac HelloWorld.java
  % java HelloWorld
  Hello, World
  ```

Dr. Java

[DrJava](http://drjava.org)
Another Java Program

```java
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.println();
        System.out.println("This program produces");
        System.out.println("four lines of output");
    }
}
```

Syntax and Semantics

- The **syntax rules** of a language define how we can put identifiers, symbols, and numbers together to make a valid program.
- The **semantics** of a program define what a program does:
  - A program that is syntactically correct is not necessarily logically (semantically) correct.
- At the very beginning, the challenge is to resolve syntax issues; but quickly, we will focus on the semantics—let a program do what we want.
Java Program Syntax: Overview

- In the Java programming language:
  - a program is made up of one or more **classes**
  - a class contains one or more **methods**
    - a Java application always contains a method called **main**
  - a method contains one or more program **statements** (instructions)

- These terms will be explored in detail throughout the semester

Structure of a Java program

```
public class <class name> {
public static void main(String[] args) {
    <statement>
    <statement>
    ...
    <statement>
}
}
```

A class:
- has a name, defined in a file with same name
- starts with {, and ends with }
- includes a group of methods

A method:
- has a name
- starts with {, and ends with }
- includes a group of statements

A statement:
- a command to be executed
- end with ;

System.out.println

- A statement that prints a line of output on the console.
  - pronounced "print-linn"

Two ways to use `System.out.println`:

- `System.out.println(<string>);`
  - Prints the given message `<string>` as output.

- `System.out.println();`
  - Prints a blank line of output.

Syntax: Strings

- **string**: A sequence of text characters.
  - Starts and ends with a " (quotation mark character).
  - The quotes do not appear in the output.

- **Examples**:
  - "hello"
  - "This is a string. It's very long!"

- **Restrictions**:
  - May not span multiple lines.
    - "This is not a legal String."
  - May not contain a " character.
    - "This is not a "legal" String either."

- This begs the question...
Escape Sequences

- **Escape sequence**: A special sequence of characters used to represent certain special characters in a string.
  - \t: tab character
  - \n: new line character
  - \": quotation mark character
  - \\: backslash character
  - Example:
    ```java
    System.out.println("hello\thow\tare \"you\"");
    
    Output:
    hello how are "you"?
    ```

Questions

- What is the output of the following `println` statements?
  ```java
  System.out.println("ta\tb\tc");
  System.out.println("\\\\");
  System.out.println("\n");
  System.out.println("C:\\in\\the\\downward\\spiral");
  ```

- Write a `println` statement to produce this output:
  ```
  / \ // /// \
  ```

Syntax: Identifier

- **Identifier**: A name given to an item (e.g., class and method) in your program.

- Syntax requirement on identifier:
  - must start with a letter or _ or $
  - subsequent characters can be any of those or a number

- Important: Java is case sensitive:
  - Hello and hello are different identifiers

Three Types of Identifiers

1. Identifiers chosen by ourselves when writing a program (such as `HelloWorld`)

2. Identifiers chosen by another programmer, so we use the identifiers that they chose (e.g., System, out, println, main)

```java
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```
Three Types of Identifiers (cont’d)

3. Special identifiers called **keywords**: A keyword has a reserved meaning in Java.

| Abstract    | Default | If          | Implement  | Private   | This        | Java revered words: they are all lowercase! |
|-------------|---------|-------------|------------|-----------|-------------|
| Break       | Do      | Else        | Instanceof | Return    | Transient   |
| Byte        | Extend  | Int         | Interface  | Short     | Try         |
| Case        | Finally | Long        | Static     | Void      |             |
| Char        | Float   | Native      | Super      | Switch    |             |
| Class       | For     | New         | Synchronized |         |             |
| Const       | Goto    | Package     |            |           |             |
| Continue    |         |             |            |           |             |

Examples

- Which of the following are legal non reserved-word identifiers?
  - Greeting1
  - g
  - class
  - 101Dalmatians
  - Hello, World
  - <greeting>

Syntax Errors

- A **syntax error** is also called a compiler error: A problem in the structure of a program that causes the compiler to fail, e.g.,
  - Missing semicolon
  - Too many or too few { } braces
  - Class and file names do not match
  - ...

Syntax Error: Example

```java
1. public class Hello {
2.   public static void main(String[] args) {
3.     System.out.println("Hello, world!");
4.   }
5. }
```
Syntax Error: Example

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

Compiler output:
Hello.java:2: <identifier> expected
  public static void main(String[] args) {
Hello.java:3: ';'. expected
  }

2 errors

- The compiler shows the line number where it found the error.
- The error messages sometimes can be tough to understand:
  - Why can’t the computer just say “You misspelled ‘public’”?

First lesson in this class

- Computers are stupid.
- Computers can’t read minds.
- Computers don’t make mistakes.
- If the computer is not doing what you want, it’s because YOU made a mistake.