CS 112 Introduction to Programming

Lecture #2: Java Program Structure

Yang (Richard) Yang

Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Outline

- Admin. and recap
- Java: the programming language
- Programming levels
- Java programming steps
- Java program structure
Make sure you sign up on classes v2 server to receive email messages

pset 1 to be posted
Recap

- Programming is to apply algorithmic thinking to design computer programs to solve problems
  - Describe each step in a computer language
    - Algorithms represent imperative knowledge vs declarative knowledge
  - Figure out why the computer did not follow the instructions as you expected
Programming Language Choices
Outline

- Admin. and recap
  - *Java: the programming language*
Java Programming Language: Key Designers

- **Bill Joy**
  - BSD Unix guy from UC Berkeley
  - co-founder of Sun Microsystems (1982)
  - focus on “the network is the computer” targeting workstation market
  - failure: focusing on network was ahead of its time, but missed the PC revolution

- **James Gosling**
  - early fame as the author of “Gosling Emacs”
    - killed by open GNU emacs
  - then onto Sun’s “NeWS” windows system
    - killed by open X-windows
  - failure: keeping things proprietary led to “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

- Third time’s the charm
  - the Web exploded
  - Oak became Java
Java Features

- Java is a modern, elegant, 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) portable --- 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
  - good, free Integrated Development Environments (IDE)
Language “Beauty Contest”

<table>
<thead>
<tr>
<th>Language Rank</th>
<th>Types</th>
<th>Spectrum Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Java</td>
<td>🌐📱💻</td>
<td>100.0</td>
</tr>
<tr>
<td>2. C</td>
<td>📱📱💻</td>
<td>99.9</td>
</tr>
<tr>
<td>3. C++</td>
<td>📱📱💻</td>
<td>99.4</td>
</tr>
<tr>
<td>4. Python</td>
<td>🌐💻</td>
<td>96.5</td>
</tr>
<tr>
<td>5. C#</td>
<td>🌐📱💻</td>
<td>91.3</td>
</tr>
<tr>
<td>6. R</td>
<td>💻</td>
<td>84.8</td>
</tr>
<tr>
<td>7. PHP</td>
<td>🌐💻</td>
<td>84.5</td>
</tr>
<tr>
<td>8. JavaScript</td>
<td>🌐📱</td>
<td>83.0</td>
</tr>
<tr>
<td>9. Ruby</td>
<td>🌐💻</td>
<td>76.2</td>
</tr>
<tr>
<td>10. Matlab</td>
<td>💻</td>
<td>72.4</td>
</tr>
</tbody>
</table>

## 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>
<tr>
<td>7</td>
<td>2011</td>
<td>Lambda</td>
</tr>
<tr>
<td>8</td>
<td>2014</td>
<td>More Lambda, Javascript runtime</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
Outline

- Admin. and recap
- Java: the programming language
- Programming levels
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
- Different machines understand different machine languages
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.

```python
  celsiusTemperature = 32;
  fahrenheitTemperature =
    celsiusTemperature * 9 / 5 + 32;
  if (fahrenheitTemperature > 100)
    hot = true;
  else
    hot = false;
```

Example Higher-level Source Code fragment
Problem

- Language barrier
  - Computers: understand machine platform languages---to build efficient hardware
  - Programmers: want more readable high-level languages---to be more productive
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).
Problems of Compiling to Each Specific Computer Platform

- Multiple versions of the same software
High-level Picture

C/C++
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.

Other languages (e.g., Jruby, Jython, Scala) may also compile 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 Mac

bytecode interpreter for Android

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++

Java
Outline

- Admin. and recap
- Java: the programming language
- Programming levels
- Java programming steps
Recall: Java Programming Steps

- Programming in Java consists of 3 simple steps
  - Create and edit “Java source code” (.java files)
  - Compile into “Java bytecode” (.class files)
  - Execute bytecode with a “Java interpreter”
Programming in Java (Step 1): Create/Edit

- The basic way is to use a text editor
  - Example editors: vim, sublime, Notepad, TextEdit (Format/Make Plain Text) etc.
    - Note: MS Word is NOT a text editor
  - The key is that your .java file cannot include any markup or stylistic formatting; just text.
  - You enter your Java code following Java Language syntax (more soon).
Programming in Java (Step 2): Compile

- Compile a Java program
  $ javac HelloWorld.java

- Take a look to see that HelloWorld.class is generated
  $ ls
  HelloWorld.java HelloWorld.class
Programming in Java (Step 3): Execute

- Run Java interpreter
  $ java HelloWorld
First Java Program

/********************************************
 * Prints “Hello World”
 * Everyone’s first Java program.
 *********************************************/

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

public class Hello2 {
    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");
    }
}
Another way is to use an Integrated Development Environment (IDE)

- Example IDEs: Eclipse, DrJava, etc.
- An IDE usually presents the user with a space for text (like an editor) but layers additional features on top of the text for the user's benefit.
  - Note: The underlying file contains pure text, just like a text editor.
- These features can be very useful and save time.
  - Example features are GUI compile, GUI execution, code completion, and syntax highlighting.
- IDEs take more time to get started than a simple text editor, e.g.,
  - set up where to find the “java” and “javac” programs
  - find out where does the IDE save my file
Outline

- Admin. and recap
- Java: the programming language
- Programming levels
- Java programming steps
- Java program (syntax) structure
Java Syntax Structure: A Top-Down View

A class:
- has a name, defined in a file with same name
  Convention we follow: capitalize each English word
- starts with {, and ends with }
- includes a group of methods

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

A method:
- has a name
  Convention we follow: lowercase first word, capital following
- starts with {, and ends with }
- includes a group of statements

statement:
- a command to be executed
- end with ;
The `System.out.println` statement

- 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.
Outline

- Admin. and recap
- Java: the programming language
- Programming levels
- Java programming steps
- Java program structure
  - A top-down view
  - A bottom-up view
// Comment 1: A Java program
/* Comment 2: a long comment
   **************************************************/
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");
    }
}
Java Syntax: A Bottom-Up View

- Basic Java syntax units
  - white space and comments
  - identifiers (words)
  - symbols: { } " ( ) <> [ ]; = ...
  - strings
  - numbers

```java
// Comment 1: A Java program
/* Comment 2: a long comment
   *******************************************/
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: White Space

- White space
  - includes spaces, new line characters, tabs
  - white space is used to separate other entities
  - extra white space is ignored

- White space allows a Java program to be formatted in many ways, and should be formatted to enhance readability
  - the usage of white space forms part of programming style
Syntax: Comments

- **comment**: A note written in source code by the programmer to describe or clarify the code.
  - Comments are ignored by the compiler
  - Useful for other people (and yourself!) to understand your code

- Two types of comments in Java
  - Single-line comments use `//...`
    `// this comment runs to the end of the line`
  - Multi-lines comments use `/* ... */`
    `/* this is a very long multi-line comment */`
Syntax: Identifier

- **Identifier**: A name given to an item in a 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

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

<table>
<thead>
<tr>
<th>abstract</th>
<th>default</th>
<th>if</th>
<th>implements</th>
<th>private</th>
<th>this</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>do</td>
<td>import</td>
<td>public</td>
<td>protected</td>
<td>throw</td>
</tr>
<tr>
<td>break</td>
<td>double</td>
<td>instanceof</td>
<td>return</td>
<td></td>
<td>throws</td>
</tr>
<tr>
<td>byte</td>
<td>else</td>
<td>int</td>
<td>short</td>
<td></td>
<td>transient</td>
</tr>
<tr>
<td>case</td>
<td>extends</td>
<td>interface</td>
<td>static</td>
<td></td>
<td>try</td>
</tr>
<tr>
<td>catch</td>
<td>final</td>
<td>long</td>
<td>strictfp</td>
<td>void</td>
<td>while</td>
</tr>
<tr>
<td>char</td>
<td>finally</td>
<td>native</td>
<td>super</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>class</strong></td>
<td>float</td>
<td>new</td>
<td>switch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>const</td>
<td>for</td>
<td>package</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>continue</td>
<td>goto</td>
<td></td>
<td>synchronized</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Java reserved words: they are all **lowercase**!
Examples

Which of the following are legal non reserved-word identifiers?

- Greeting1
- g
- class
- 101dalmatians
- _101dalmatians
- Hello, World
- <greeting>
Syntax: Strings

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

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

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

Which of the following are legal strings in Java?

- "This is a string. It’s very long!"
- "This cool string spans two lines."
- "It is a great thing when children cry, "I want my mommy"!"
Escape Sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

  - \b backspace
  - \t tab character
  - \n new line character
  - " quotation mark character
  - \ backslash character

- **Example:**
  ```java
  System.out.println("\\hello\\nhow\\tare "you"?\\\\");
  ```

- **Output:**
  ```
  \hello
  how   are "you"?
  ```
Comment on syntax errors

- **A syntax/compile 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
  - …

- **Compilers can’t (DO not) read minds.**
- **Compilers don’t make mistakes.**
- **If the program is not doing what you want, do NOT blame the computer---it’s YOU who made a mistake.**
Reading and Practice Slides (Out of Class)
Questions

- What is the output of the following `println` statements?

```java
System.out.println("\ta\tb\tc");
System.out.println("\\\\");
System.out.println("\'");
System.out.println("\\\\");
System.out.println("C:\nin\the downward spiral");
```

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

```
/ \ //  \ \  ///  \\
```
Answers

- **Output of each `println` statement:**

  a  b  c

  \/
  \\,
  """

  C: in he downward spiral

- **`println` statement to produce the line of output:**

  System.out.println("/ \ \ // \ /// \ /// /// /");
Questions

What `println` statements will generate this output?

This program prints a quote from the Gettysburg Address.

"Four score and seven years ago, our 'fore fathers' brought forth on this continent a new nation."

What `println` statements will generate this output?

A "quoted" String is 'much' better if you learn the rules of "escape sequences."

Also, "" represents an empty String. Don't forget: use \" instead of " ! "' is not the same as "'}
**Answers**

- **println statements to generate the output:**

  ```java
  System.out.println("This program prints a");
  System.out.println("quote from the Gettysburg Address.");
  System.out.println();
  System.out.println("\"Four score and seven years ago,"");
  System.out.println("our 'fore fathers' brought forth on");
  System.out.println("this continent a new nation.\"");
  
  System.out.println("A \"quoted\" String is");
  System.out.println("'much' better if you learn");
  System.out.println("the rules of \"escape sequences.\"");
  System.out.println();
  System.out.println("Also, \"\" represents an empty String.");
  System.out.println("Don't forget: use \\\" instead of \"!");
  System.out.println("\" is not the same as \");
  ```
Questions

What `println` statements will generate this output?

This quote is from Irish poet Oscar Wilde:

"Music makes one feel so romantic – at least it always gets on one's nerves – which is the same thing nowadays."

What `println` statements will generate this output?

A "quoted" String is 'much' better if you learn the rules of "escape sequences."

Also, "" represents an empty String. Don't forget: use "\" instead of "! "" is not the same as ""
Answers

println statements to generate the output:

```java
System.out.println("This quote is from");
System.out.println("Irish poet Oscar Wilde:");
System.out.println();
System.out.println("\"Music makes one feel so romantic\"");
System.out.println("- at least it always gets on one's nerves -");
System.out.println("which is the same thing nowadays.\"");
```

println statements to generate the output:

```java
System.out.println("A \"quoted\" String is");
System.out.println("'much' better if you learn");
System.out.println("the rules of \"escape sequences.\"");
System.out.println();
System.out.println("Also, \"\" represents an empty String.");
System.out.println("Don't forget: use \"\" instead of \"!");
System.out.println("'' is not the same as \"");
```
Syntax and Semantics

- The **syntax rules** of a language define how we can put **characters** 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.
  - This is similar in natural language, e.g.,
    - "Yale University has no dining halls."
    - "Harvard can beat Yale."

- 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.
Some Common Compile/Syntax Errors

- **A syntax/compile 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
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```
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’*”?
  - Since the computer knows that a “;” is missing, can’t it just fix it??
Java Programming Steps and Errors

- **Compile-time errors**
  - the compiler may find problems with syntax and other basic issues
  - if compile-time errors exist, an executable version of the program is not created

- **Run-time errors**
  - a problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (crash)

- **Logical errors**
  - a program may run, but produce incorrect results
End Outside Slides
Backup Slides
Assembly language or simply assembly is a human-readable notation for the machine language.

It's much easier to remember:

```
  movl %al, 97
```

than

```
  10110000 01100001
```

Example assembly code fragment:

```
movl (%edx,%eax), %ecx
movl 12(%ebp), %eax
leal 0(%eax,4), %edx
movl $nodes, %eax
movl (%edx,%eax), %eax
fldl (%ecx)
fsubl (%eax)
movl 8(%ebp), %eax
leal 0(%eax,4), %edx
movl $nodes, %eax
movl (%edx,%eax), %ecx
movl 12(%ebp), %eax
leal 0(%eax,4), %edx
movl $nodes, %eax
```
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
    - successful in industry; used to build major OSes such as Windows
  - Java (1995): designed for embedded systems, web apps/servers
    - Runs on many platforms (Windows, Mac, Linux, cell phones...)