University of British Columbia
CPSC 111, Intro to Computation
Alan J. Hu

Machine vs. High-Level Languages
Interpreters and Compilers
Writing a Simple Java Program
Lecture 1

Some slides borrowed from Kurt Eiselt and Steve Wolfman.
Some learning goals from Beth Simon.

Readings
Your textbook is Big Java (3rd Ed).
This week’s reading: Chapter 1.
Next Week: Ch 2.1-2.5, Ch 4.1-4.2.

Administrative Stuff
You will need: a Campus-Wide Login (CWL), and an account with the CS department.

You can find instructions for getting all of these on WebCT. This course is on the new Vista server: http://www.vista.ubc.ca

A good starting point is the course website, which will also get you to WebCT:
http://www.ugrad.cs.ubc.ca/~cs111/

Labs and Tutorials
Labs and tutorials start next Monday.

However, do Lab #0 (on WebCT) on your own, this week. Lab #1 will be available soon, too, so you should read that to be ready for next week.

You must be enrolled in a lab section. If the lab sections you need are full, go to ICICS Room 201 and see a Computer Science undergrad advisor!

Learning Goals
By the end of class today you will be able to...

- Explain why it’s useful to have both machine/assembly language and high-level computer languages (and how they are different)
- Describe what compilers and interpreters do
- Type in, compile, and run a simple Java computer program.
  - HINT: You should actually do this on your own before the next class!
- Navigate around a simple Java program and recognize some basic building blocks, such as the class header, methods, identifiers, strings, numerical constants, and comments.
- Use white space and comments to make programs easier to read.

Last Time...

- Computer science gets physical artifacts to aid or exhibit thinking!
- Basics of how a computer is organized, how it works.
  - Binary representation: everything is just 0s and 1s
  - Memories: fixed size chunks of data that are accessed via numerical addresses
  - Processors/CPU’s: execute instructions one-at-a-time
- You can make the computer do anything you want, if you program it correctly…

But you must write in a language the computer understands!
George and Stephen go to France

- George is American. He knows only English.
- Stephen is Canadian. He is bilingual in English and French.
- How can George communicate in France?

1. If he wants to communicate quickly, then Stephen can interpret—translating French to English and English to French on-the-fly.
2. If there's a lot of stuff to translate (e.g., a speech, or a long document), then Stephen can translate the whole thing at once. Now, George can read it whenever he wants.

Translators can be combined:
In the Louvre, they see inscriptions in Egyptian hieroglyphics.

A museum sign gives a French translation.

Stephen interprets the sign for George.

George can understand the hieroglyphics.

Health Education in Remote Areas

- In remote areas of the world, there are languages spoken by small groups of people, and also a national language spoken by the mainstream, e.g.:
  - Many native languages vs. Spanish in Latin America
  - Minority languages vs. Mandarin in China
  - Regional languages vs. Hindi or English in India

- How do you provide health info to the isolated?

<table>
<thead>
<tr>
<th>Original Info (English)</th>
<th>Localized Field Manual (Spanish)</th>
<th>English-Spanish Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Interpreter</td>
<td>Local Interpreter</td>
<td>Local Interpreter</td>
</tr>
<tr>
<td>Nahuatl Speaker</td>
<td>Maya Speaker</td>
<td>Zapoteco Speaker</td>
</tr>
</tbody>
</table>

Machine Language

- This is the “native language” of a computer.
- Remember: Everything is in binary!
- Each instruction does very little.
- The computer does them very fast.
- Each kind of processor has its own machine language, e.g.:
  - x86 (Intel, AMD), Windows and new Apples
  - PowerPC (Freescale, IBM), older Apples, Sony PS3
  - SPARC (Sun), used in Sun servers
  - Many more…
### Machine Language
- Examples: An instruction to read the content of a memory location:
  - x86: `movl %eax, (%ebx)`
    - `10001011 11000000 00011011`
  - SPARC: `ld [%r8+0], %r9`
    - `1 1100101 00000 01000 1 0000000000000`

(Examples not guaranteed to be perfect. It’s very error-prone to do these by hand!)

### Assembly Language
- Assembly language is just an easier-to-read version of machine language:
  - x86: `movl %eax, (%ebx)`
    - `10001011 11000000 00011011`
  - SPARC: `ld [%r8+0], %r9`
    - `11 01001 00000 01000 1 0000000000000`

(Almost never do humans write machine language. It’s just a nit-picky translation.)

### High-Level Language
- A High-Level Language is a computer language designed to be easier for humans:
  - `a=b+c;`
- Must be translated into machine language so the computer can understand it.
- Who does the translation? The computer!

### High-Level Language
- A High-Level Language is a computer language designed to be easier for humans:
  - `a=b+c;`
- Must be translated into machine language so the computer can understand it.
  - You could translate it as you go (**interpreter**).
  - You could translate it in advance (**compiler**).
- Java is the high-level language we’ll use.
  - Modern, widely used, portable, safe.

### Compilation Example
- On an Intel-based Linux PC, the `a=b+c` example turns into:
  - `movl -8(%ebp), %eax`
  - `addl -12(%ebp), %eax`
  - `movl %eax, -16(%ebp)`

### Interpreters and Compilers
- An interpreter translates the high-level language into machine language on-the-fly, executing the instructions as it goes.
- A compiler translates the high-level language program all at once in advance.
- Both compilers and interpreters are themselves computer programs.
- Which is better? (Remember George and Stephen in France?)
Java Does Both!

Java Does Both!

A Simple Java Program

// Our first Java program.
/* Traditionally, one's first program in a new
language prints out "Hello, World!"
*/
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Controlling the Computational Behavior

A procedure or algorithm is a collection of instructions in
some meaningful order that results in useful behaviour on
behalf of the device that executes the instructions.

When the instructions are written in a symbolic language
that can be executed by a computer, the procedure is
called a computer program.

A process is what happens when a computer follows a
program - it's a procedure in execution.

Procedures and algorithms

Computer people often use the words “procedure” and “algorithm” interchangeably...we will too.

An algorithm is

• a finite procedure
• written in a fixed symbolic vocabulary
• governed by precise instructions
• moving in discrete steps, 1, 2, 3, ...
• whose execution requires no insight, cleverness,
  intuition, intelligence, or perspicuity
• and that sooner or later comes to an end

David Berlinski in The Advent of the Algorithm

Procedures and algorithms

Here's why we get frustrated when we start to learn
to write program to make computers do stuff:

An algorithm is

• a finite procedure
• written in a fixed symbolic vocabulary
• governed by precise instructions
• moving in discrete steps, 1, 2, 3, ...
• whose execution requires no insight, cleverness,
  intuition, intelligence, or perspicuity
• and that sooner or later comes to an end

We don't have a lot of practice at being precise!

Procedures and algorithms

Here's why we get frustrated when we start to learn
to write program to make computers do stuff:

An algorithm is

• a finite procedure
• written in a fixed symbolic vocabulary
• governed by precise instructions
• moving in discrete steps, 1, 2, 3, ...
• whose execution requires no insight, cleverness,
  intuition, intelligence, or perspicuity
• and that sooner or later comes to an end

We don't have a lot of practice at being stupid!
How to avoid frustration

Practice, Practice, Practice

It takes a lot of practice to learn to be precise enough to make a computer do what you want

It takes a lot of practice to keep from assuming that the computer is smarter than it really is

It takes a lot of practice to get good at this stuff

What do you see?

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!” */
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Comments

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!” */
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

White Space (e.g., Indentation)

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!” */
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
White Space

- “White space” means the spaces, line breaks, and tabs in your program.
- Java ignores white space (with very few exceptions), so you can format any way you like.

Example…

Good Formatting

- Use white space to format programs so they’re easier for people to understand.
  (Appendix A is a good start.)

Write Readable Programs!

- Use comments, formatting, and other ways to make your program easy to understand.
- TAs will be grateful.
- Other programmers will be grateful.
- You will be grateful.
- A program is both an expression of an idea, and a way to make a computer do a task.
  (Interesting legal/philosophical issues…)

White Space (e.g., Indentation)

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!”
*/
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Strings

// Our first Java program.
/* Traditionally, one’s first program in a new language prints out “Hello, World!”
*/
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Strings

- Anything in quotation marks “…” is a string.
- Beware: Java doesn’t use “smart” quotes.
- This tells Java that you really want exactly those characters (letters, digits, whatever), just like that. As if you’re quoting someone.
Strings
// Our first Java program.
/* Traditionally, one’s first program in a new
language prints out “Hello, World!” */
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Identifiers (and Reserved Words)
// Our first Java program.
/* Traditionally, one’s first program in a new
language prints out “Hello, World!” */

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

Identifiers and Reserved Words
- Word-like things in the program.
  - Special rules, which we’ll learn very soon.
  - Identifiers are usually names of things, like
    “HelloTester”
  - Reserved words are special words that have
    special meanings to Java, which we’ll learn
    later, like “class” and “public”

Curly Braces
// Our first Java program.
/* Traditionally, one’s first program in a new
language prints out “Hello, World!” */
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Curly Braces, Square Brackets, Parentheses
- Java uses three different pairs of parenthesis-like symbols to group things:
  - Curly Braces: {}  
  - Square Brackets: []  
  - Parentheses: ()
- We’ll learn the rules for what to use where later. Curly braces usually group big things over many
  lines; parentheses usually group on one line.
- Must be balanced, just like parentheses in math:
  \( v(w+x(y-z)) \)
Curly Braces

// Our first Java program.
/* Traditionally, one's first program in a new
language prints out "Hello, World!"
*/
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Numbers

- There were no numbers in our simple program, but you'll often see numbers in Java programs, and they usually mean what they look like:
  - 3
  - 3.14159

Operators

- There weren't many operators in our simple Java program, but whenever you see symbols like + or =, they are probably Java operators.
- Operators tell Java to perform some action.
- You'll learn most of the Java operators in this course.

A Simple Java Program

// Our first Java program.
/* Traditionally, one's first program in a new
language prints out "Hello, World!"
*/
class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}

Multiple Statements

class HelloTester {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
        System.out.print("I am feeling chatty");
        System.out.println(" today.");
    }
}

Another Example

class ManyOreos {
    public static void main(String[] args) {
        System.out.println("Feed me more Oreos!");
        // I'm being sloppy. You'll learn better ways to do this later.
        javax.swing.JOptionPane.showMessageDialog(null,"Give me a cookie!");
        javax.swing.JOptionPane.showMessageDialog(null,"I want cookie!");
        javax.swing.JOptionPane.showMessageDialog(null,"Cookie!!!!");
    }
}
Remember the Program Counter

- Program Counter was the special part of a processor that points to the instruction to execute.
- Fetch-Decode-Execute: one instruction at a time.
- Java works the same way.
- You can use your finger to point to “where you are” in your program.
- Even expert programmers do this sometimes!

Identifiers

More on Identifiers

- Words we use when writing programs are called identifiers
  - except those inside the quotes

Sample Java Application Program

```java
public class Oreo
{
    public static void main (String[] args)
    {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Kurt made up identifier Oreo

Sample Java Application Program

```java
public class Oreo
{
    public static void main (String[] args)
    {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Other programmers chose identifiers System, out, and println
  - They wrote printing program.
  - Part of huge library of useful programs that comes with Java
  - By using the right identifier, we find what we want.

Sample Java Application Program

```java
public class Oreo
{
    public static void main (String[] args)
    {
        System.out.println("Feed me more Oreos!");
    }
}
```

- Special identifiers in Java called reserved words
  - don’t use them in other ways
Reserved Words

- Get familiar with these (Appendix G)
- But you don’t need to memorize all 52 for exam

abstract   do    if    private    throw
boolean   double    implements    protected    throws
byte     enum    instanceof    return    true
case     extends    int    short    try
catch  false    interface    static    void
char  finally    long    strictfp    volatile
class  finally    native    super    void
const   float    new    switch
continue  for    null    synchronized
default   goto    package    this

Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
  - Digits are 0 through 9.
  - Letters are
    - the 26 lowercase letters in English alphabet
    - the 26 uppercase letters in English alphabet
    - plus the $ and _
    - also alphabetic characters from other languages

Which of the following are not valid identifiers?

userName  user_name  $cash  2ndName
firstName  user.age  _note_  note2

Identifiers

- Java is case sensitive
- Oreo  oreo  OREO  0reo
  - are all different identifiers, so be careful
  - common source of errors in programming

Which of these are all valid identifiers?
Identifiers

- Creating identifiers in your Java programs
  - Remember other people read what you create
  - Make identifiers meaningful and descriptive for both you and them
  - No limit to how many characters you can put in your identifiers
  - but don’t get carried away

```java
public class ReallyLongNamesWillDriveYouCrazyIfYouGoOverboard {
    public static void main (String[] args)
    {
        System.out.println("Enough already!");
    }
}
```

Laws versus Customs/Conventions

- In real life, some things are illegal: murder, shoplifting, running stop signs, etc.
- Other things are legal, but social norms dictate that we don’t do them, because life is better for everyone if we agree to follow the rules: cutting in line, talking on your cell phone in class/movie/concert, being rude, etc.

Java has conventions, too.

- Rules for identifiers are like laws: If you break them, the compiler won’t let your program compile:
  - E.g.,: 3am is not a legal identifier.
- Java also has conventions. These are the normal way Java programmers do things. It’s good to follow them. E.g.,:
  - CamelCase: Makes it easier to read.
  - Don’t start identifiers with $ or _
  - We’ll learn more as the course goes on…

Questions?