Introduction to Programming

Chapter 1 – Lecture Slides

Programming

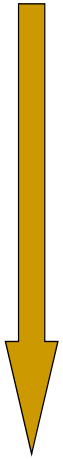
- Learning to program is learning how to problem solve.
- The goal of this course is to help you develop critical thinking skills and problem solving.
- Problem solving is important for all disciplines, and for the real-world.
- Programming requires:
  - Formulate problems
  - Think creatively about solutions
  - Express a solution clearly and accurately

Useful

- These skills are important for all fields; for example:

* Mathematics
  - use of formal languages
* Engineering
  - designing, assembling components, evaluating between solutions
* Natural Science
  - observing behavior of humans/animals/plants/weather
  - forming hypotheses and testing
* Art
  - digital image/video/sound/movie manipulation
* History
  - recording, storing, indexing, searching, analyzing and retrieving historical documents and information
* All fields
  - process of discovering something new
  - exploration of data and analysis

History of Computing



1791-1871 : Charles Babbage, "father of computing" with invention of the Analytical Engine. He died before his ideas were fully realized.

1949/1950 – first computer developed

1969 – ARPANET Advanced Research Projects Agency of U.S. Dept of Defense networked a group of computers together (UCLA, Stanford Research Institute, UC Santa Barbara and University of Utah). Internet was born!

1971 – E-mail invented

1979 – Newgroups born, first MUD (Multiuser Dungean game)

1990 – "World Wide Web" name coined for the project based on hypertext media

1991 – Gopher program released – menu interface to the internet.

1994 – Yahoo! born

1995 – Java programming language released

Definitions

- Programming language
  - Grammar to designate information and instructions that a computer understands
  - **Syntax** – grammar of a programming language
    - Different programming languages follow different syntax
- Instructions or operations
  - Steps for the computer to follow
  - A program is a set of instructions for the computer

Stupid Computer

- Computers are still relatively stupid.
  - Restrictive language (doesn't understand English)
  - Small mistake in a programming language can kill the program
- Example:
  make me draw a dot on board with red marker
  - I don't know how to "grab the marker" – where is it?
  - I don't know how to "go over to the board" – explain to me how to walk!

What Can Computers Do?

- Process information much faster than humans
  - Lots of data
  - Mathematical computation
  - Automation
- Artificial Intelligence
  Philip K. Dick and Hansen's projects
  http://hansonrobotics.com
  + elderly to have someone to talk to
  + children practice new languages
  - who are you flirting with? Man or machine?
- Don't understand English
- Do understand programming languages
- *Computers never know what you intend, only what you tell it to do*

What the machine understands

- Binary : 0 or 1
  - Similar to On/off switches (lights)
- Bit
  - 1 bit can contain either a 0 or 1
  - 2 bits: different combinations of 0 and 1

0	0		0	1		1	0		1	1

- Counting in binary
- Adding in binary
  - *Decimal hits max at 9, go back to 0 carry the 1*
  - *Binary maxs out at 1, go back to 0 carry the 1*

Counting in binary:

000	= 0
001	= 1
010	= 2
011	= 3
100	= 4
101	= 5
110	= 6
111	= 7

Decimal	Binary
9	101
+ 1	+001
10	110

How Machine Code can Differ

- Different platforms (Windows Intel vs. Macintosh vs. Sparc station) represent data in machine code differently
- For example, if a number is represented in 4 bytes (each byte = 8 bits)
  - x86: 01 23 45 67

0000 0001	0000 10111	0001 01101	0010 0011

  - Mac: 67 45 23 01

0010 0011	0001 01101	0000 10111	0000 0001

Programming

- Instructions for computer to execute to solve a problem
- **Algorithm** = steps to solve a problem
  - could be a program but not necessarily a program
    - e.g., recipe
- Languages
  - Machine 01010011 11101000 10010101
  - Assembly Add $2, 9
  - High X + 9

Source code compiled

- High-level language:
  a = b + 9;
- After compilation, may become
  lw $2, b
  lw $3, 9
  add $2, $2, $3
  sw $2, a
- Which is then translated into bits something like
  00010110 01000010 10000010
  00010110 01000011 00001001
  00010101 01000010 01000010 01000011
  00011000 01000010 10000001

From source code to execution

From source code to execution

Compiler, Interpreter

- High-level code gets mapped to low-level code
- **Compiler**: program that translates an entire program into a target language (machine or otherwise)
- **Interpreter**: translates one line of a program into the machine code equivalents

Advantages of Interpreters

- Different machines can have interpreters specific to that machine
  - Machine (platform) independence
  - No need for specific compiler for each machine
  - Code compiled on any machine can be run on any other machine with an interpreter

Java features, due to use of interpreter:

Disadvantages of Interpreters

- Code is slower to execute (10-100 times)
  (first needs to translate to machine code before executing)
- Still require an interpreter for each machine to run the code
- Limited to abilities that all machines can produce (e.g., lose the extra graphics abilities of SGIs)

Why Java?

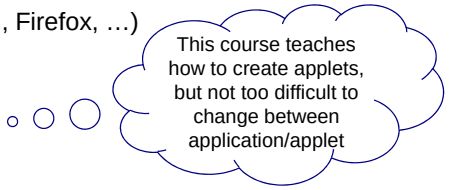
- Java provides a programming tool for web pages that
  - Has graphics capabilities
  - Is machine independent
  - Easy to implement (compared to C, assembly)
  - Create applets for the Internet

Why NOT Java?

- Undergoing constant change
- Language growing in size
- It is slower than some other languages

Java application vs. applet

- Application
  - Stand alone (e.g. MS Word, Firefox, …)
- Applet
  - Embedded in a web page



Java Files

- **ClassName.java** = Java source code (what you write)
  - HelloWorld.java
- **ClassName.class** = Java byte code (compiled from the source code)
  - HelloWorld.class
- **WebPageName.html** = web page
  - Hello.html
- Web pages reference the .class file

```
<HTML>
<BODY>
<APPLET CODE=HelloWorld.class
        WIDTH=500
        HEIGHT=300  >
</APPLET>
</BODY></HTML>
```

- Hand in the **.java** file for homeworks!

Object-Oriented Programming

- Program consists of objects from classes
- A class is a set of objects (instances) with a common structure (and purpose)
- Objects consist of:
  - Data values (instance variables)
  - Methods (class procedures/functions)

Object-Oriented Programming

- Objects consist of:
  - data values (instance variables)
  - methods (class procedures/functions)
- Example: Best vehicle on the planet: Jeep Wrangler!
- What data do we keep track of on a Jeep?
  -
- What behaviors or methods can we perform on a Jeep?
  -
  -
    -

Syntax

- **Syntax** = grammar rules
  - Like English: sentences end with punctuation, subject then verb...
  - programs:
    - Statements end with a semi-colon
    - Class header then squiggleys { }
    - Methods go inside the class squiggleys
  - Java is case sensitive: Hello/HELLO/hello all different
  - reserved words
    - Mean something special in Java – so you can't use them for other purposes
    - e.g.: public, class, void

Java Packages

- Java provides a lot of classes that perform useful actions
  - (Don't reinvent the wheel!)
  - Code reuse
- Our job is to produce specialized classes to perform particular tasks

- java.awt – abstract window toolkit (GUI)
- java.io – input/output classes
- java.lang – language features
- java.net – classes for working with networks
- java.util – other useful utility classes
- javax.swing – new applet and graphics

Hello World


```
/* First Program
 *  @author  E.S.Boese
 */
import javax.swing.*;   // add classes from swing pkg
import java.awt.*;      // add classes from awt pkg

public class HelloWorld extends JApplet
{
    public void paint( Graphics g )
    {
       g.drawString( “Gooooooood Morning!”, 30, 30 );
    }
}
```


Applet Example

```
/** @Author: E. S. Boese          e-id: sugar
  * Date: Spring 2005 Course: CS150  */

import javax.swing.*;       // swing package
import java.awt.*;          // graphics package

public class HelloWorld extends JApplet
{
   public void paint( Graphics g )
   {
      g.drawString( "Hello World!", 0, 12 );
   }
}
```

class name **HelloWorld** must match filename listed in Package Explorer

“0,12” are the coordinates where to draw the string – what happens if “0,0”?

Applet Example

```
/** @Author: E. S. Boese          e-id: sugar
  * Date: Spring 2005             Course: CS150  */

import javax.swing.*;       // swing package
import java.awt.*;          // graphics package

public class HelloWorld extends JApplet
{
   public void paint( Graphics g )
   {
      g.drawString( "Hello World!", 0, 12 );
   }
}
```

comments are for readability – they have no effect on the program

the paint method is automatically called by the browser

Comments

- Created as documentation for readability
- Compiler ignores all comments
- Examples
  - The rest of the line is a comment:

    ```
    import javax.swing.*;     // swing package
    import java.awt.*;        // graphics package
    ```
  - Multiple lines are commented:

    ```
    /** @Author: E. S. Boese          e-id: sugar
     * Date: Spring 2005
     * Course: CS150  */
    ```

Applet Example

```
/** @Author: E. S. Boese          e-id: sugar
  * Date: Spring 2005 Course: CS150  */

import javax.swing.*;        // swing package
import java.awt.*;           // graphics package

public class HelloWorld extends JApplet
{
   public void paint( Graphics g )
   {
      g.drawString( "Hello World!", 0, 12 );
   }
}
```

To create applets, we inherit from the JApplet class so we don't rewrite the code to make an applet work!

"drawString" is a method from the Graphics object **named g**

extends JApplet

- Applets need to extend (inherit) the JApplet class:

  `public class xyz` **`extends JApplet`**

- This way, we can reuse code already written instead of writing it ourselves!
- This class inherits many capabilities, including what to do to make applets work

Applet Example, using Eclipse

```
/** @Author: E. S. Boese          e-id: sugar
  * Date: Spring 2005 Course: CS150  */

import javax.swing.*;        // swing package
import java.awt.*;           // graphics package

public class HelloWorld extends JApplet
{
   public void paint( Graphics g )
   {
      g.drawString( "Hello World!", 0, 12 );
   }
}
```

import statements to tell compiler which packages to look and find classes, such as the Applet class and the Graphics class

import

import packagename.*; // imports all classes in package

OR

import packagename.classname; // imports a specific class in a package

Examples:

```
import java.awt.*;
import javax.swing.*;
import javax.swing.JApplet;
```

The * designates to include ALL classes within the package

- Java provides classes with graphics abilities
- To use these classes we need to import the packages (java.awt package)
- This way, we can reuse code already written instead of writing it ourselves!
- Must be at beginning of file

Applet Example

- Change some things and see if it works, or if you get errors
- Change the coordinates from “0,12” to “0,0” – where does it go?
- Add a second line of text – how do you align it under “Hello World”?
- Try to center the text

```
/** @Author: E. S. Boese                e-id: sugar
  * Date: Spring 2005      Course: CS150  */

import javax.swing.*;          // swing package
import java.awt.*;             // graphics package

public class HelloWorld extends JApplet
{
   public void paint( Graphics g )
   {
      g.drawString( "Hello World!", 0, 12 );
   }
}
```


Summary

- Introduction to programming
- Types of languages
- Compilers and Interpreters and Java bytecode
- Java applets
  - Import statements
  - Comments
  - Class header declaration
  - Paint method