Lecture 4: Introduction to Processing

AP Computer Science Principles

Programming

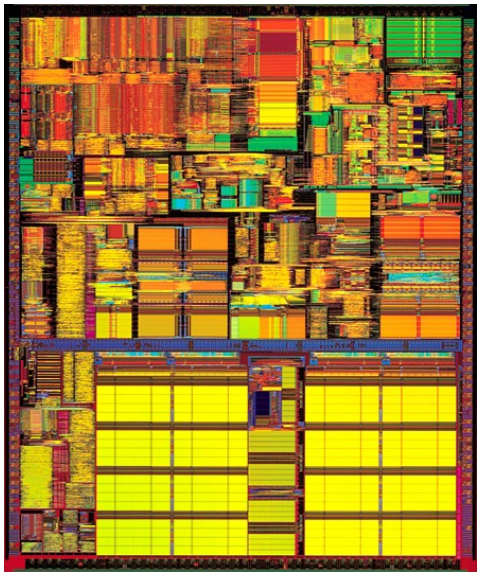
A Program

- **program**: A set of instructions to be carried out by a computer.
- **program execution**: The act of carrying out the instructions contained in a program.
- **programming language**: A systematic set of rules used to describe computations in a format that is editable by humans.

CPU

- Computers are fast
  - Pentium G4560(dual-cores) can perform approximately 3,500,000,000(3.5 Ghz) computations per second
  - made up of 3,300,000,000 transistors (a switch that is on or off)

- Computers are dumb
  - They can only carry out a very limited set of instructions
    - on the order of 100 or so depending on the computer's processor
    - machine language instructions, aka instruction set architecture (ISA)
    - Add, Branch, Jump, Get Data, Get Instruction, Store.



Neumann

- John von Neumann - co-author of paper in 1946 with Arthur W. Burks and Hermann H. Goldstine,
  - "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument"
- One of the key points
  - program commands and data stored as sequences of bits in the computer's memory
- A program:

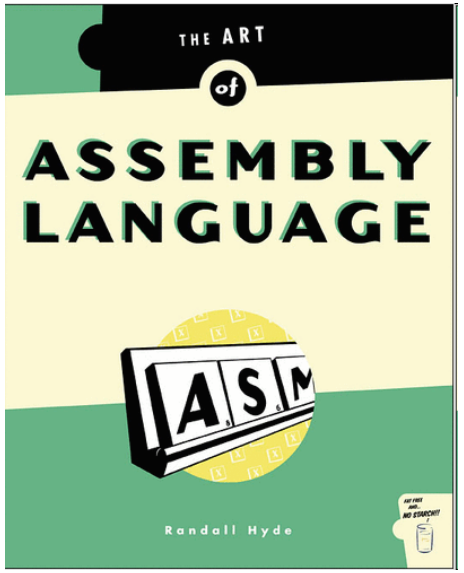
```
1110001100000000
0101011011100000
0110100001000000
0000100000001000
0001011011000100
0001001001100001
0110100001000000
```

Low-level programming

- Programming at a low-level(closer to the hardware level) with Strings of bits (1s or 0s) is not the easiest thing in the world.

- Assembly language(a little bit higher level, but not much)
  - mnemonics for machine language instructions

```
.ORIG     x3001
LD        R1, x3100
AND       R3, R3 #0
LD        R4, R1
BRn       x3008
ADD       R3, R3, R4
ADD       R1, R1, #1
LD        R4, R1
BRnzp     x3003
```

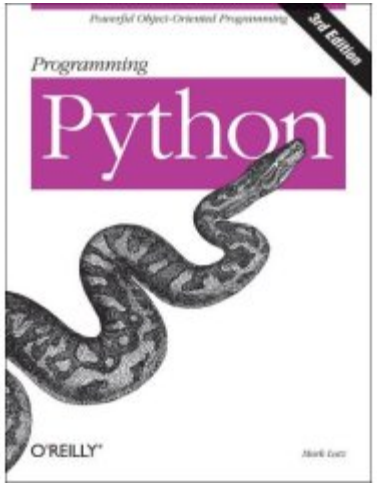


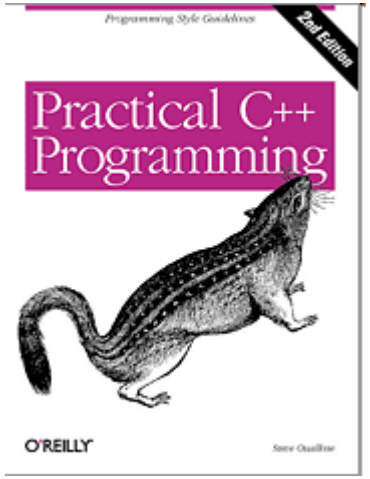
Higher Level Programming

- Assembly language has lots of commands to accomplish simple things.

- High Level Computer Languages provide the ability to accomplish a lot with fewer commands than machine or assembly language in a way that is hopefully easier to understand and write.

```
int sum = 0;
int count = 1;
while(count <= 10){
  sum = sum + count;
  count = count + 1;
}
```





Languages

- There are hundreds of high level computer languages. Java, C++, C, Basic, Fortran, Cobol, Lisp, Perl, Python, Javascript.

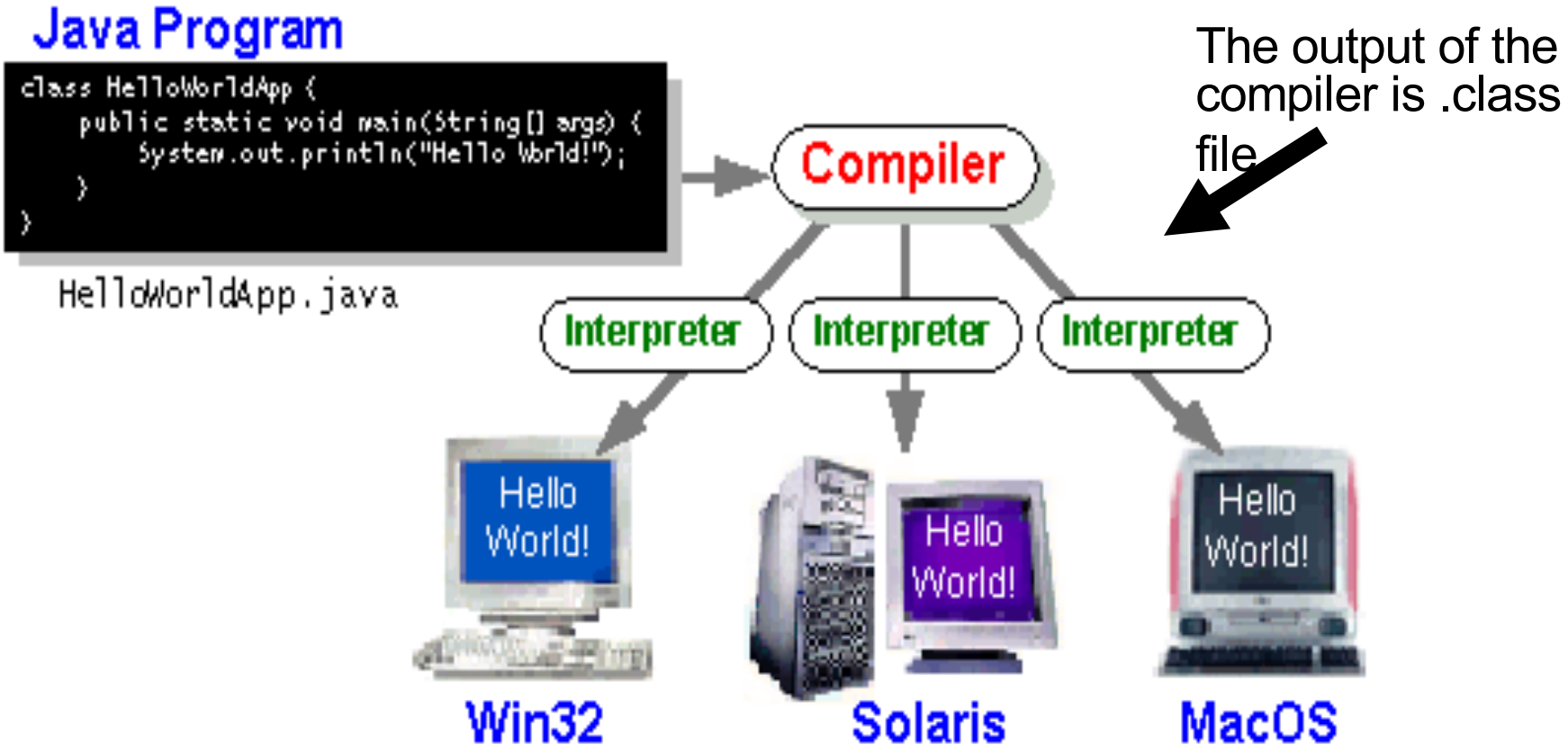
- The capabilities of the languages vary widely, but they all need a way to do
  - declarative statements
  - conditional statements
  - iterative or repetitive statements

- A **compiler** is a program that converts commands in high level languages to machine language instructions.

Some modern 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
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **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...)

A Picture is Worth…



The Interpreter's are sometimes referred to as the Java Virtual Machines

Java

•Java will be the primary language we'll use. (We'll be introduced to Python later).

•The editor(or IDE, Integrated Development Environment) is Processing.

Processing

A program, called a sketch, in Processing, at its most basic, consists of just two **methods**(a method is a sequence of code that perform a specific task): setup() and draw().

setup(): initializing variables, setting up window size, etc…

draw(): repeats 60 times a second(60hz refresh rate), responsible for the animation and flow of a program.

Our First Program

Java

- Java is case sensitive.
- Every statement of code ends with a semicolon.
- // marks the beginning of an inline comment. The compiler will ignore everything after the //.

```
// This is a comment.
// So is this.
int x = 3;  //this is an inline comment.
```

- Extra whitespace/tabs/returns are ignore.

A basic sketch

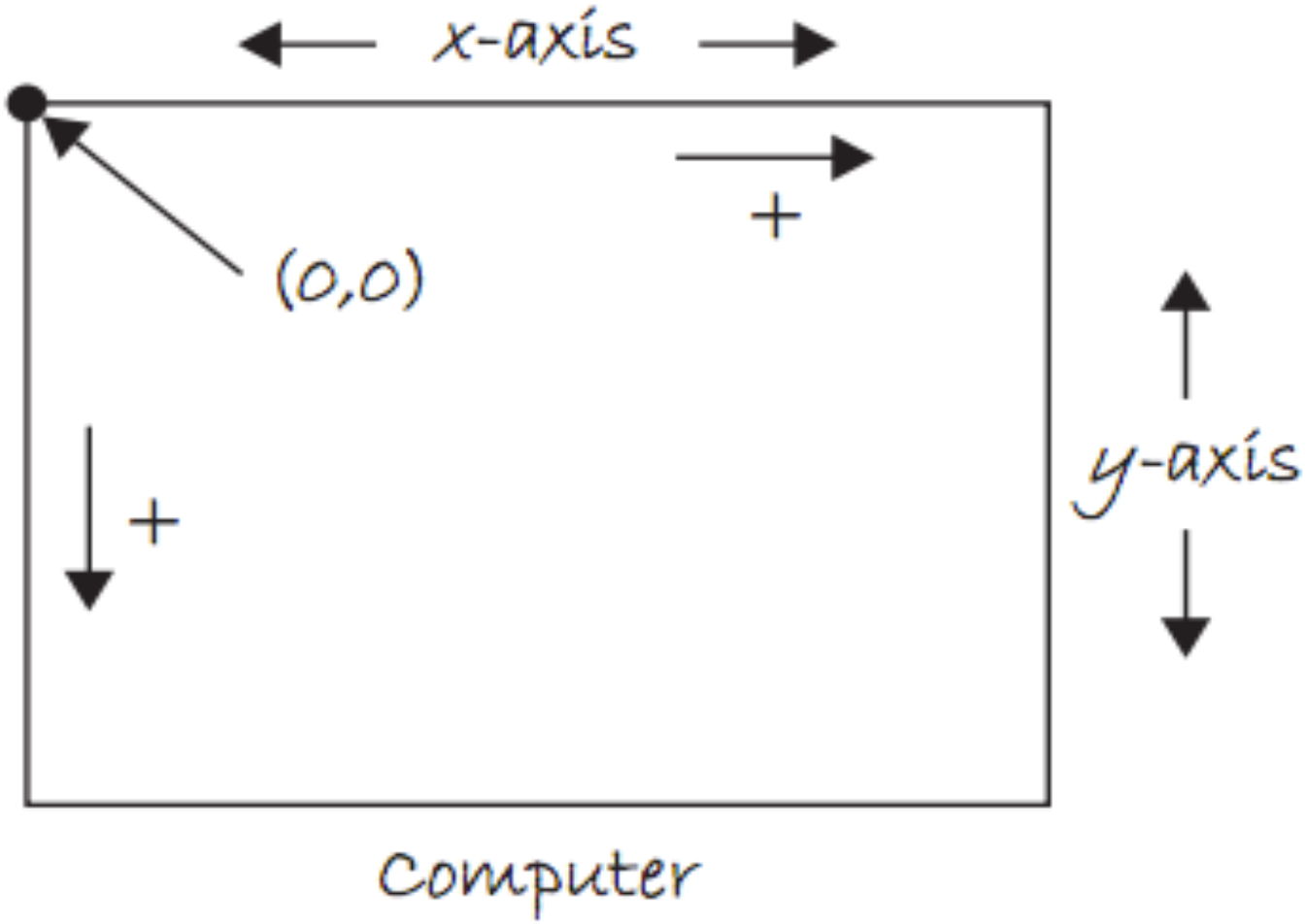
```
// global variables are declared at the top before setup()
void setup(){
        // code here to initialize global variables,
        // background color, window size, etc…
        // setup() is run only once.
}

// draw is a loop that runs
// automatically 60 frames per second
// used for animation
void draw(){
        // code for updating variables for next frame
        // creating illusion of animation
}
```

Example

```
int x; //declaring global variables; not initialized
void setup(){
      //initializing x and y
      x = 5;
}
void draw()
{
      x = x + 1; //x increases by 1
      // 60 times per second.
}
```

Coordinate System



Example 1

```
int centerX, centerY;
void setup(){
        size(800,600); // window width=800 pixels, height=600 pixels
        centerX = 400;
        centerY = 300; //(centerX,centerY)center of window
}

void draw(){
        background(255); //white background
        fill(255,0,0); //red fill (Red,Green,Blue) color (RGB)
        ellipse(centerX,centerY,80,80);
        //red circle radius 40(diameter=80)
}
```

Making it move!

```
int centerX, centerY, xspeed;
void setup(){
        size(800,600); // window width=800 pixels, height=600 pixels
        centerX = 400;
        centerY = 300; //(centerX,centerY) center of window
        xspeed = 5;
}
void draw(){
        background(255); //white background
        fill(255,0,0); //red fill (Red,Green,Blue) color (RGB)
        ellipse(centerX,centerY,80,80); //red circle radius 40
        centerX = centerX + xspeed; //updating centerX
        //every frame, centerX increases by 5,
        // ball moves horizontal 5 pixels to the right.
}
```

Back and Forth

```
int centerX, centerY, xspeed;
void setup(){
        size(800,600); // window width=800 pixels, height=600 pixels
        centerX = 400;
        centerY = 300; //(centerX,centerY) center of window
        xspeed = 5;
}
void draw(){
        background(255); //white background
        fill(255,0,0); //red fill (Red,Green,Blue) color (RGB)
        ellipse(centerX,centerY,80,80); //red circle radius 40
        centerX = centerX + xspeed; //updating centerX
        if (centerX > 800 || centerX < 0) { // || means "or"
                xspeed = -xspeed;
        }
}
```

Processing References

The Processing website is a great reference for learning about Processing. There are tutorials on different topics, videos, etc…

A quick way to learn how to do something on Processing is through Google. If I want to know how to draw a rectangle. Search for “draw rectangle processing”, the first result will take you directly to the code on Processing.

There are also many videos on Processing on youtube.

Lab 1

Download Processing at https://www.processing.org/.

Retype the previous program to see the ball bouncing back and forth. Please, for this first time, DO NOT copy and paste. Practice learning how to do this without any reference.

Modify the program so that the ball moves in a diagonal direction by adding a vertical speed, yspeed.

Add an “if” conditional to make sure the ball bounces of the top and bottom of the screen.

Lab 1(continued)

Some more things to try.

- Play around with the window size.
- Try changing the background into RGB color instead of greyscale.
- Add a vertical speed, yspeed, to have the ball moves in a diagonal direction.

Lab 1(continued)

- As the program is written here, the ball does not perfectly bounces off the edge of the window. Fix it by introducing a radius variable!

Lab 2

Watch the first two videos(Drawing Basic Shapes and Basic Animation) of the **Introduction to Processing Series** on my Youtube channel.

Write a sketch that:

- Draw a circle, a line, a rectangle, a square of different sizes and color. Add some text to the screen.
- Draw a shape that follows your mouse.
- Use beginShape() to draw any polygon with 6 sides.

Homework

1)Read and reread these lecture notes.

2)Download processing. https://www.processing.org/

3)Do the labs. Learn and understand the code.

References

Part of this lecture is taken from the following book.

Stuart Reges and Marty Stepp. Building Java Programs: A Back to Basics Approach. Pearson Education. 2008.