CSE 114 Intro to OOP

PROGRAMMING OVERVIEW

1

Announcements

- Resources on Java and Emacs installation added to Brightspace!
- TA Schedule to be posted today
- Course web:
  - Brightspace: https://mycourses.stonybrook.edu/d2l/home/691920
- Survey form: See Lecture 1
  - Please fill this out as soon as possible and upload to blackboard 'Survey' under Assignments
- Reading assignment for this week: Chapter 1 of Downey
- If you are asked to move to CSE 101, please do so ASAP

2

What is computer science?

- Very broad definition:
  Computer science (CS) is the systematic study of computing systems and computation
- Computer science is NOT just programming!
- Programming is an important part of CS
  We will be learning to program in Java but much of what you will learn are fundamental CS concepts that apply to nearly any programming language

3

What do computer scientists do?

- Some examples of what computer scientists work on:
  - Build computers and their components
  - Programming languages
  - Operating systems
  - Artificial intelligence, machine learning
  - Databases
  - Networking
  - Theory of computation
  - Computer graphics
  - Robotics
  - Biocomputing
  - Many more . . .

4

What is programming?

- Programming is the process of "giving" instructions to a computer or the computer's central processing unit (CPU)
  - Also referred to as 'Writing code' or 'Coding'
- Learning to program is similar to learning a "natural" language like Spanish. You have to learn the:
  - **Syntax:** The grammar or 'rules' of the language
  - **Semantics:** The meaning of each word and phrase in the language
- But…Computers are pretty dumb
  - You have to give super clear and precise instructions
  - A computer will happily do the same thing forever (in an infinite loop) if you tell it to do so, even if you didn't mean to!
- Unlike natural languages, programming languages are extremely picky – rules can't be violated

5

What is a program?

- **Program**: A sequence of instructions to be carried out by a computer (to perform a computational task)
- **Program execution**: The act of carrying out the instructions contained in a program
- Example: see Hello.java
  - What is the computational task that we perform in this example?

6

Hello.java

```
public class Hello {
   public static void main (String[] args) {
      System.out.println("Welcome to CS!!!");
      System.out.println("Let's have some fun.");
   }
}
```


7

Low-level computer instructions

- Machine language
  - Binary instruction (1's and 0's)
  - Most instructions just move data around or perform simple arithmetic operations
- Example: on Intel x86 processors:
  - 1011000001100001 means to copy a 97 to a particular register

Binary programming is really hard and tedious but early programmers did exactly this!

8

“Mid-level” computer instructions

- Assembly language
  - Symbolic (meaningful) names for binary instructions and memory
  - A little more readable
  - Feasible for programmers to use
- Example:

```
ADD DR, SR1, SR2        ; DR <- (SR1) + (SR2)
LD DR, LABEL            ; DR <= Mem[LABEL]
LDR DR, BaseR, Offset   ; DR <- Mem[BaseR + Offset]
STI SR, LABEL           ; Mem[Mem[LABEL]] <= SR
```


9

High-level computer instructions

- High-level language
  - Symbolic names for assembly instructions and memory
  - Symbolic names for basic operations such as looping
  - Symbolic “shorthand” for a group of instructions
  - Close to "natural" languages - close to being readable!
- Example:

```
print("Welcome to CS!");
y = a * x + b;
```


10

Why use a high-level language?

- Concise
  - High-level programming languages allow us to express common operations in a concise and readable fashion
- Maintainable
  - Modifying and maintaining code is much easier when the code is concise and easy to read (as compared to lengthy and difficult to read assembly or binary code)
- Portable
  - Different CPU's accept different binary instructions
  - Writing in a high-level language allows code to be translated or "compiled" into a platform-specific binary code
  - Allows your code to be "ported" to another platform

11

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 languages: functions map inputs to outputs
  - Lisp (1958) / Scheme (1975)
  - ML (1973)
  - Haskell (1990)
- Object-oriented languages: programs use interacting “objects”
  - Smalltalk (1980): first major object-oriented language
  - C++ (1979): “object-oriented” improvement to C
  - Java (1995): designed for embedded systems, servers
    - Runs on many platforms (Windows, Mac, Linux, cell phones, . . .)
  - Python (1989): general purpose, procedural and objects
  - Javascript (~1995): the language of the web, part of every browser

12

Why Java?

- Java is a straightforward and powerful object-oriented language
  - Simpler than C++
  - Highly productive language
- Platform independent (Mac, Windows, ...)
  - "Write once, run everywhere."
- Java is used heavily in industry
  - Many major companies have utilized Java in their internal technical infrastructure
- Java has good support
  - Many useful prewritten "packages" available as part of the Java ecosystem such as graphics, multimedia, networking, etc.
- The first programming language taught at many universities

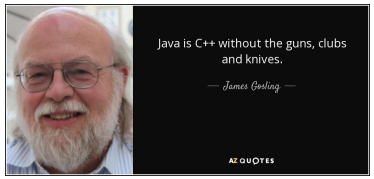
13

Java's roots

- C was developed in the early 1970's
  - C was designed to be small, fast, with little built-in safety
- C++ was developed in the late 1970's
  - C++ is a superset of C and extends C to include object-oriented concepts
- Java borrowed from C/C++ but more concerned about safety and productivity at the cost of some speed
  - Java's creator James Gosling has described Java as "C++ without the guns, clubs and knives"
  - Designed to be used in the Internet era

14

Gosling – Java's creator



https://www.azquotes.com/quote/596302

15

Java Introduction

16

Ex: Hello program (Hello.java)

```
public class Hello {
  public static void main (String[] args) {
    System.out.println("Welcome to CS!!!");
    System.out.println("Let's have some fun.");
  }
}
```

- Black text is given by Java
- Blue text was added as part of the developer's program
- Note the following (hopefully intuitive) features
  - Indentation is optional (but makes the code more readable)
  - Matching parens (and)
  - Matching braces { and }
  - Matching brackets [and]

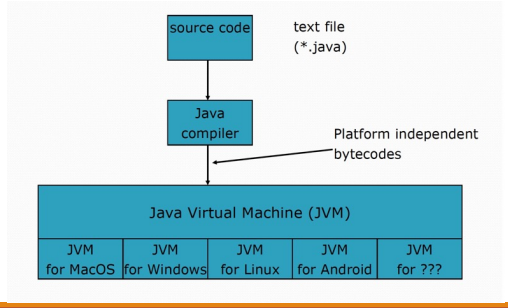
17

The coding process

1. Create/Edit source code (.java file)
   Use a text editor to write or edit your code (instructions)
   E.g., emacs, vi, atom, notepad++, but not Word
2. Compile the source file
   The Java compiler creates bytecodes – an intermediate machine independent instructions that target a Virtual Machine
   The javac command
3. Run on the Java Virtual Machine (JVM)
   The java command
4. Debug your program/application
   Repeat steps 1 – 4 as necessary

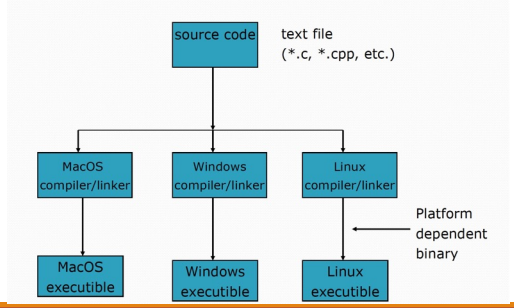
18

Java programming model



19

Traditional programming model



20

Summary – bytecodes and the JVM

- In order to "write once, run everywhere", Java uses the notation of a secure universal "Java Virtual Machine" (JVM)
- In Java you do not have to worry about cross-platform idiosyncrasies. . . your program will run "on" the JVM instead
- The Java compiler only has to worry about producing one "binary" known as bytecodes
- The JVM runs your program by interpreting or translating your program's bytecodes into platform specific machine code
- There are multiple JVM's for each supported platform (Mac, Windows, Linux, Android, etc.) but they all understand bytecodes

21

Editing code

- Programmers use a text editor to write code
  - Don't use a word processor such as MS Word, Google Docs, etc.
  - Don't use the default PC/Mac editor
- The classic editors (free): emacs, vim
  - Both are cross-platform (PC, Mac, Linux), keyboard-centric (minimal GUI)
- Standalone GUI-based editors (free):
  - PC: Notepad++, Programmer's Notepad
  - Mac: TextWrangler, Textmate2
  - Both: Sublime Text 2 (free trial), Atom (free from github)
- You only need one editor that you like!
- Text editors do not know how to compile your code!
- Text editor in IDE's (next slide)

22

Java IDE's

- IDE – Integrated Development Environment
  - Combines text editor, compiler, debugger into one GUI-based environment
- There are several reasonable IDE's for Java
  - Eclipse (PC, Mac, Linux)
  - Netbeans (PC, Mac)
  - IntelliJ IDEA (PC, Mac)
- We're going to use IntelliJ IDEA, but not yet
- Installation instructions for IntelliJ will be on the course web

23

Java program structure: class

```
// comments about the class
public class MyClass
{
}
```

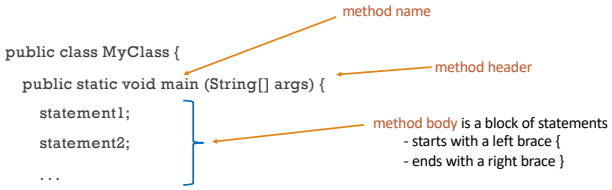
This line is the class header

class name
- must match the file name
- convention is to capitalize 1st letter of each word

class body
- starts with a left brace {
- ends with a right brace }

24

Structure: Class with a main method



Right now, do not worry about:
public, class, static, void, String[] args
- Think of them as necessary decoration for now.

25

Ex: Hello program (Hello.java)

```
public class Hello {
  public static void main (String[] args) {
    System.out.println("Welcome to CS!!!");
    System.out.println(34/10);
  }
}
```


26

Syntax errors

- syntax: the set of legal structures and commands that can be used in a particular language, e.g.,
  - Every basic Java statement ends with a semicolon (;)
  - The contents of a class or method occur between { and }
- syntax error (compiler error): a problem in the structure of a program that causes the compiler to fail, e.g.,
  - Missing semicolon
  - Too many or two few or unmatched braces { }}
  - Illegal identifier for class name, method name, variable name, etc.
  - Class and file names do not match
  - etc.
- Will use Hello.java to demonstrate some examples

27

Semantic errors

- semantics: the meaning of a language construct
  - What is the meaning of 'x = x + 1'?
- semantic error (run-time error): a problem in the meaning of a program that causes the run-time execution to fail, e.g.,
  - Divide by zero
  - Unintended infinite loop
  - Unintended piece of code, e.g., mistakenly adding when subtraction was intended
  - etc.
- Will use Hello.java to demonstrate one example

28