Introduction to Computational Thinking

Lecture #2:

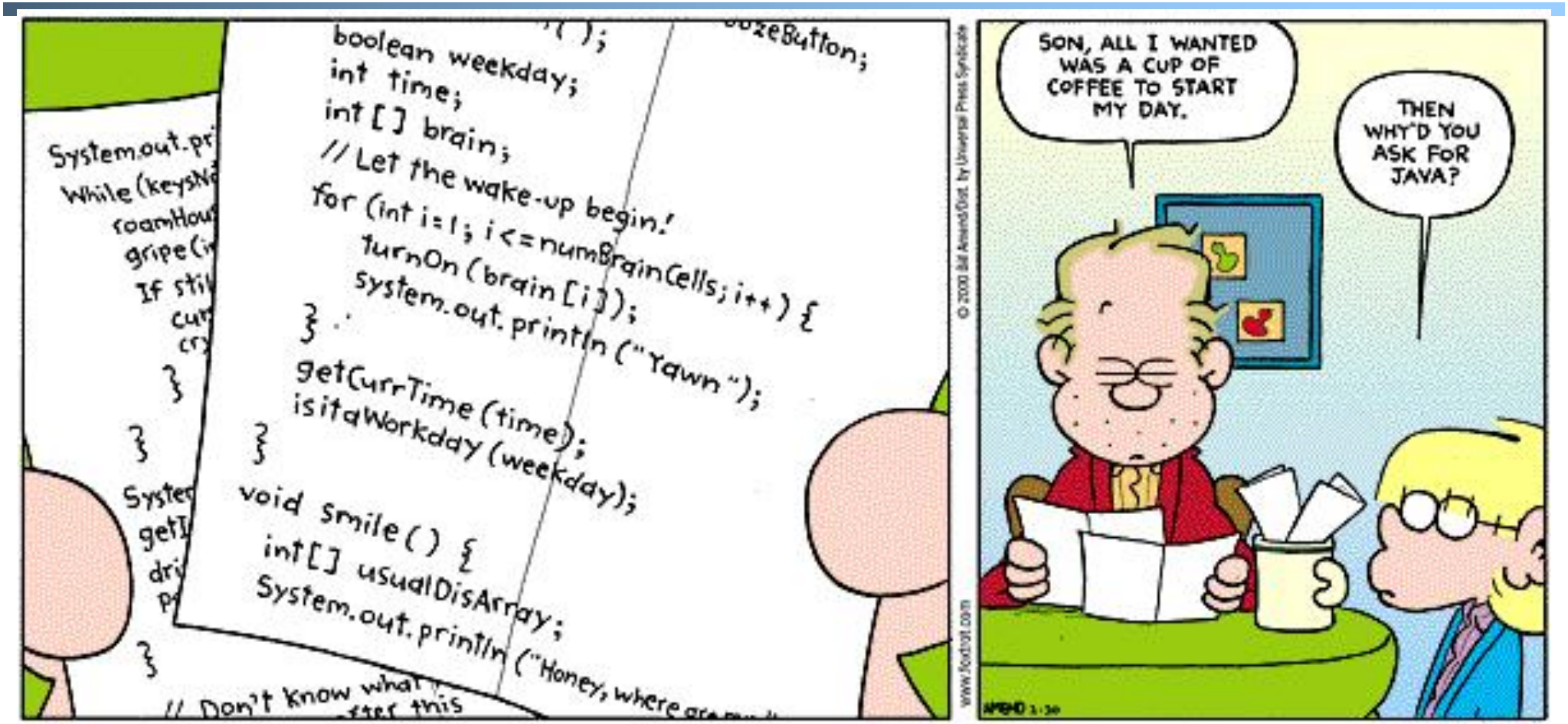
Computational Thinking Overview & *Java Program Structure*

Qiao Xiang, Qingyu Song

https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

10/11/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.


boolean weekday;
int time;
int [] brain;
// Let the wake-up begin!
for (int i=1; i<=numBrainCells; i++) {
turnOn (brain [i]);
system.out.println ("Yawn");
}
getCurrTime (time);
isitaWorkday (weekday);
}
void smile () {
int[] usualDisArray;
System.out.println ("Honey, where
SON, ALL I WANTED WAS A CUP OF COFFEE TO START MY DAY.
THEN WHY'D YOU ASK FOR JAVA?

Outline

- Admin. and recap
- Computational process and development
- Java: the programming language
- Programming levels
- Java programming steps
- Java program structure

Admin

- pset 1 to be posted

Questions for You

- Please return the class survey at the end of the class

Recap

- Computer science (CS) is the study of computational processes
  - for problem solving and creative expression
  - that are correct, smart, and practical
- CS combines
  - logic, algorithmic, systems thinking, and
  - network thinking
- Computational thinking (CT) is the way of thinking underlying the computer science discipline

Recap

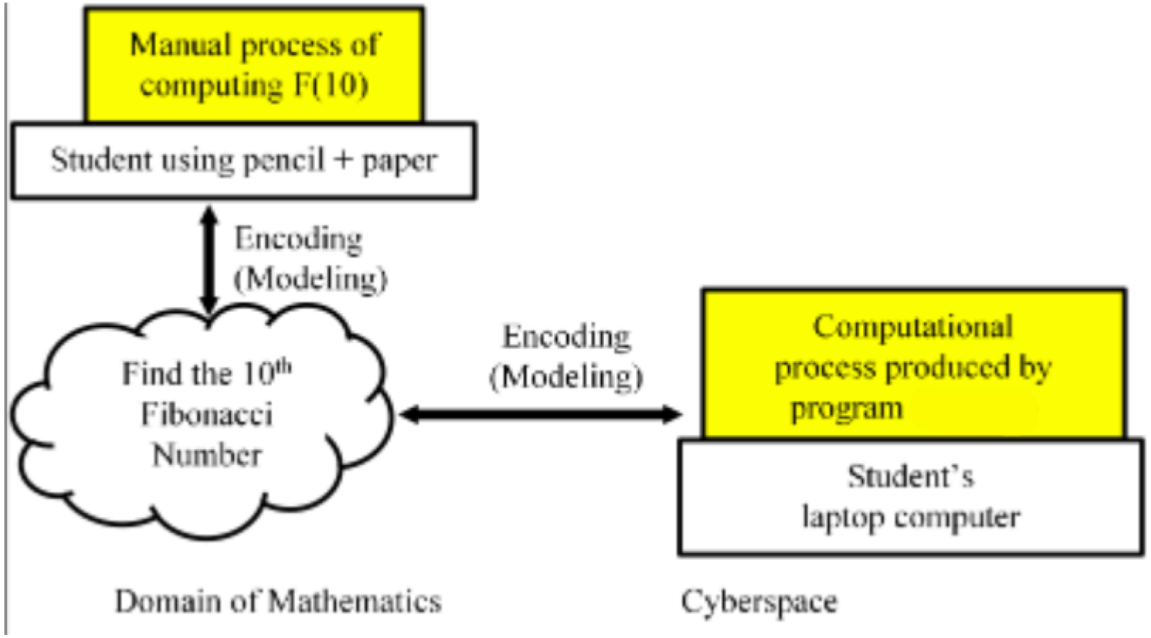
- Programming is to apply algorithmic thinking to design computer programs to solve problems
  - Describe each step in a computer language
    - Algorithms represent imperative knowledge vs declarative knowledge
  - Figure out why the computer did not follow the instructions as you expected

Outline

- Admin. and recap
- *Computational process and development*

Computational process

- A step-by-step process of information transformation
  - A sequence of symbol manipulation steps
  - Can be done manually or automatically
- Compute Fibonacci number F(10)
  - Given definition
    F(0)=0, F(1)=1,
    F(n)=F(n-1)+F(n-2)
    when n>1,
    Find F(10).
- Manual
  - Tedious
  - Impractical for large n
- Computer
  - Automatic after encoding into cyberspace
  - Practical even for n = 1 billion



2. ABC features

- Automatic execution
- Bit accuracy
- Constructive abstraction

Automatic execution

- Computational processes are automatically executed step-by-step on computers.
- Automatic execution is common when looking inside or from outside
  - Step-by-step mechanic automatic execution of digital symbol manipulation is the most fundamental characteristic of computational thinking, both without and within.
  - It underlies all the other understandings.
  - CS studies logic that is automatic executable logic, algorithms that are automatic executed algorithms, abstractions that are automatic executed abstractions.
- It partially answers the question
  - Why and how trillions of instructions can be automatically executed in a fraction of a second, sometimes across the globe, to produce correct computational results?

Bit accuracy

- Other sciences pursue their own scientific rigor
  - Experiments results are statistically significant when the p-value is less than 0.05
- Computer science uses binary values of 0s and 1s
  - One *binary digit* is called a **bit**
- Computer science pursues **bit accuracy**
  - A computational process is accurate and precise up to every bit
  - Any practical computer has finite memory
    - Cannot represent real numbers of arbitrary precision

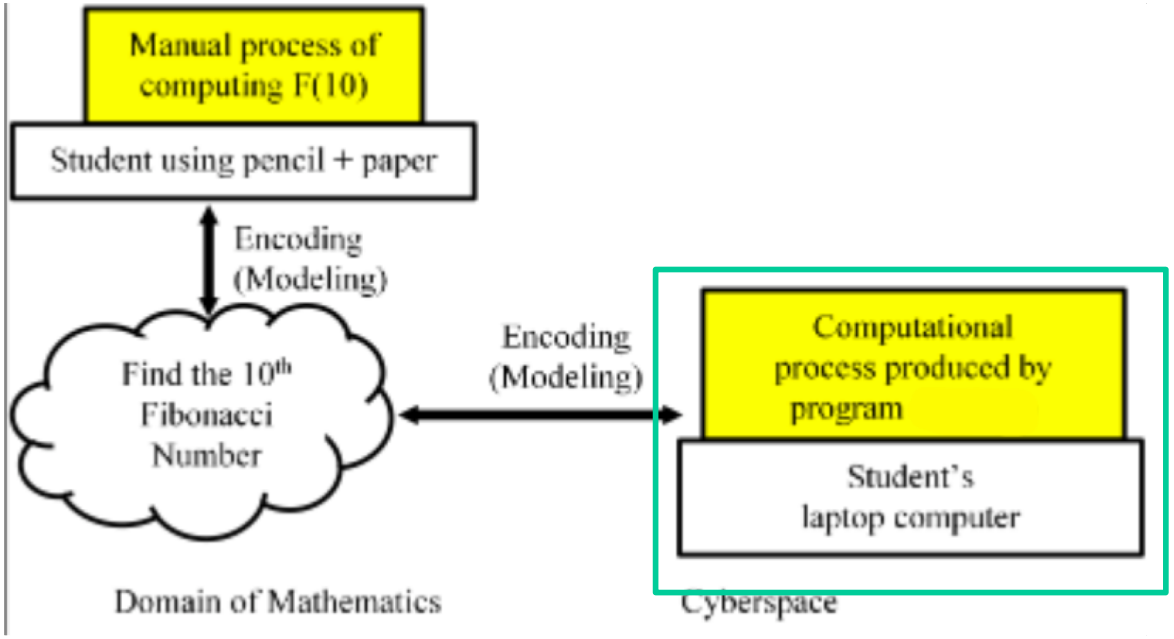
Constructive abstraction

- Computer science is about digital abstractions
  - constructive, automatically executed abstractions of information transformation

- Three layers of meaning
  - *Abstraction* from concrete instances to the general concept
  - *Constructive*: a step-by-step integration of more primitive symbols and operations
  - *Smart* construction, not ad hoc, arbitrary actions or processes
    - Although may use brute-force actions (e.g., exhaustive enumeration) and seemingly arbitrary random operations (e.g., randomly picking a number)

How to develop a computational process?

Data + Program



What are Data & Program?

- **Data** is a group of bits.
- A bit is a binary digit with a value of 0 or 1.
- A digital symbol is any notation that is representable as one or more bits, to denote any concrete or abstract entity.
- Manipulation is a sequence of operation steps on digital symbols, where the length can be one or many.
- A **program** is an algorithm written in a computer language.
- Code is a fraction (segment) of a program.

What is a digital symbol

- A notation
  - Representable as one or more bits
    - Discrete, not continuous
    - Finite, not infinite many bits
  - Denoting any concrete or abstract entity
    - Number, character, string, image, audio, video
    - Idea, program code, human genome sequence

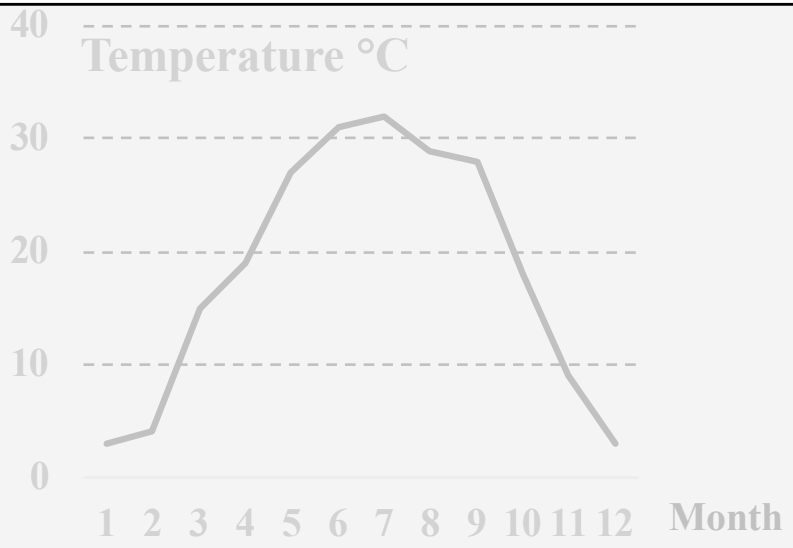
A WeChat QR code

- Manipulation is
  - Any sequence of operation on symbols
    - Find (look up), read, write, transport (send), modify

https://www.ncbi.nlm.nih.gov/nuccore/LR991581

Data are digital symbols

- Analog vs. digital values
  - Analog: Continuous
  - Digital: Discrete
- Binary-decimal conversion



Month		Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Temperature °C	Decimal	0	0	20	20	30	30	30	30	30	20	10	0
	Binary	00	00	10	10	11	11	11	11	11	10	01	00

You will learn details in another course!

- Representations
  - **ASCII:** American Standard Code for Information Interchange
  - uses 8 bits to encode English characters

Program:Digital symbol manipulation

- Computer science studies computational processes

- A computational process is a process of information transformation,
  which is a sequence of steps of digital symbol manipulation
  - Process = a sequence of steps
  - Information transformation = digital symbol manipulation

Process of information transformation

a sequence of steps

digital symbol manipulation

Take away

- Data as Symbols
- Programs to manipulate symbols

Programming Language Choices

Outline

- Admin. and recap
- Computational process and development
- *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
    the computer"
    workstation m
  - failure: focusi
    network was a
    time, but miss
    on PC revolutio

- James Gosling

  - early fame as the author of "Gosling Emacs"
    - killed by open GNU emacs
  - then onto Sun's "NeWS"
    ystem
    y open X-windows
    eping things
    y led to "kiss of

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
- Third time's the charm
  - the Web exploded
  - Oak became Java

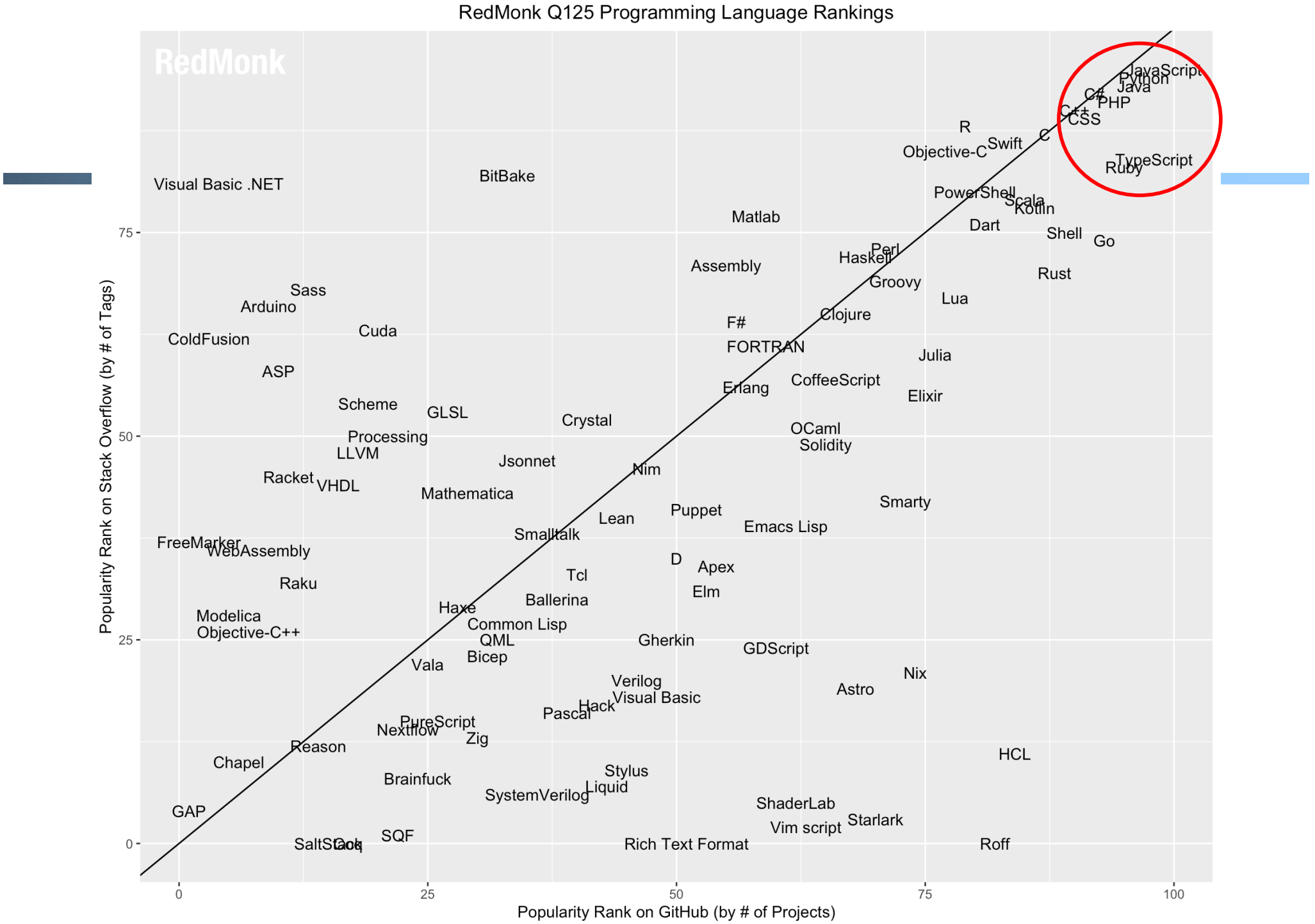
Java Features

- 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"

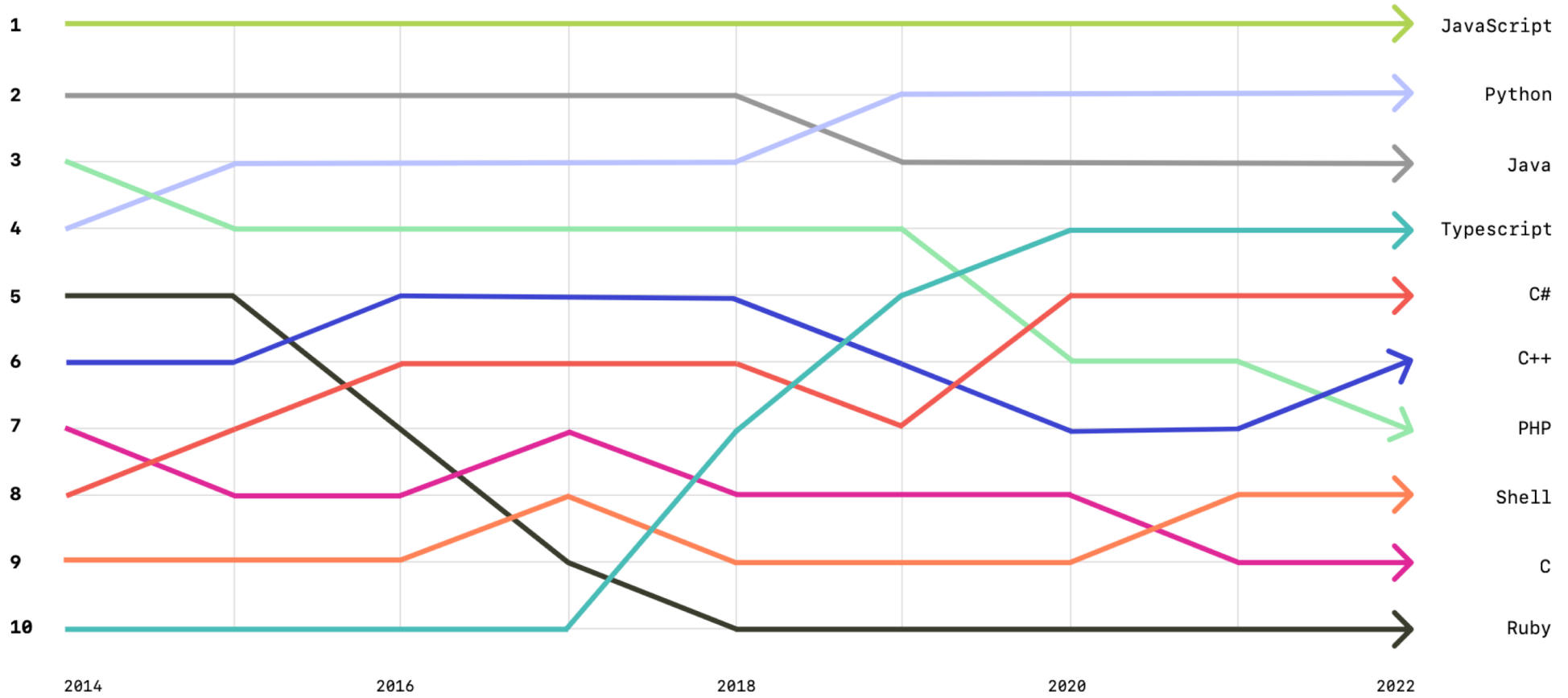
Rank	Language	Type	Score
1	Python	web, desktop, server	100.0
2	Java	web, mobile, desktop	95.4
3	C	mobile, desktop, server	94.7
4	C++	mobile, desktop, server	92.4
5	JavaScript	web	88.1
6	C#	web, mobile, desktop, server	82.4
7	R	desktop	81.7
8	Go	web, desktop	77.7
9	HTML	web	75.4
10	Swift	mobile, desktop	70.4

https://spectrum.ieee.org/top-programming-languages-**2021**



https://redmonk.com/sogrady/files/**2025**/06/lang.rank_.125.wm_.png

Language popularity



https://octoverse.github.com/**2022**/top-programming-languages

Java is Still Evolving

Version	Year	Important New Features
1.0	1996	
5	2004	Generic classes, enhanced for loop, auto-boxing, enumerations
10	2018	Local-Variable Type Inference
15	2020	Dynamic class-file constants, HTTP client (standard)
20	2023	Enhancing concurrent programming and developer productivity
25	2025	Aid in the development of AI applications

- 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
- Computational process and development
- 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

MacBookPro

Qualcomm S4
Nexus 4,
Samsung Galax

A9
iPhone 6s

High-Level Programming Languages

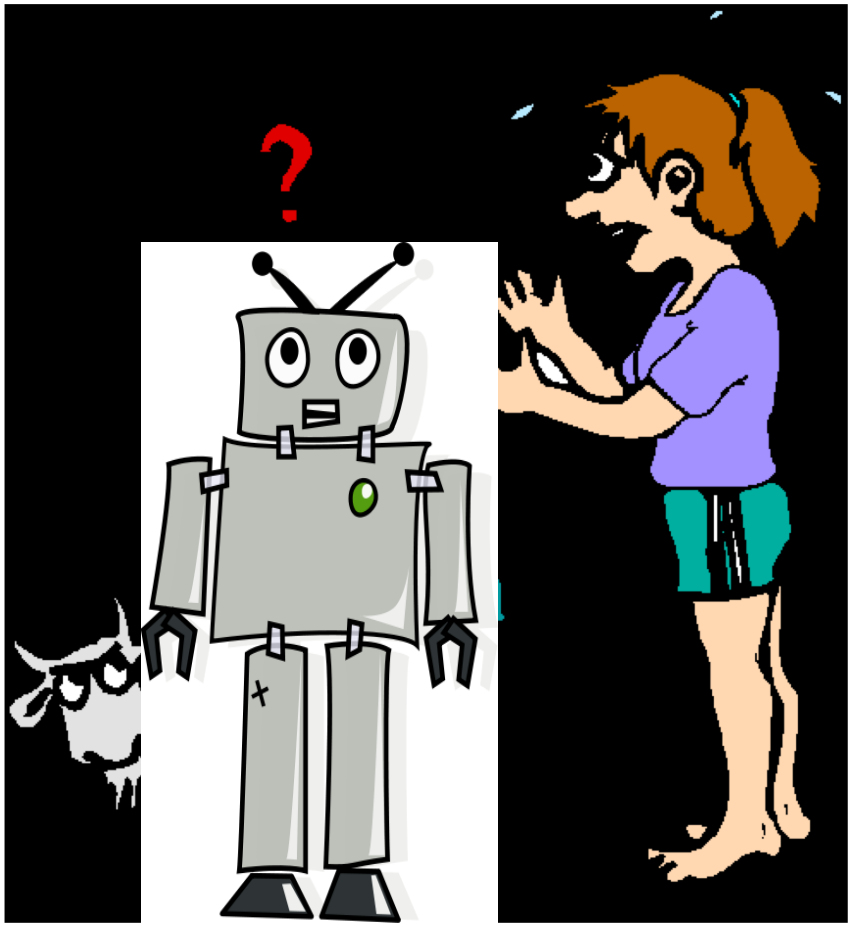
- A high-level programming language enables a programmer to specify, in a high level (close to natural language), what data a computer will act upon, how these data will be stored, and what actions to take under various circumstances
- The syntax and grammar of a high-level language is independent of CPU

```
celsiusTemperature = 32;

fahrenheitTemperature=
        celsiusTemperature * 9 / 5 + 32;

if (fahrenheitTemperature > 100)
  hot = true;
else
  hot = false;
```

Example Higher-level Source Code fragment

Problem

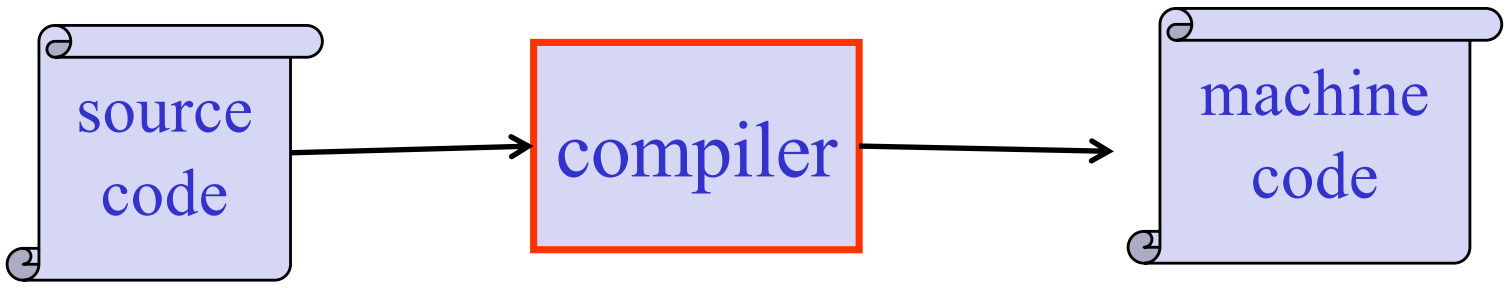
- Language barrier
  - Computers: understand machine platform languages---to build efficient hardware
  - Programmers: want more readable high-level languages---to be more productive



Hire a Translator: Compiler

- 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)

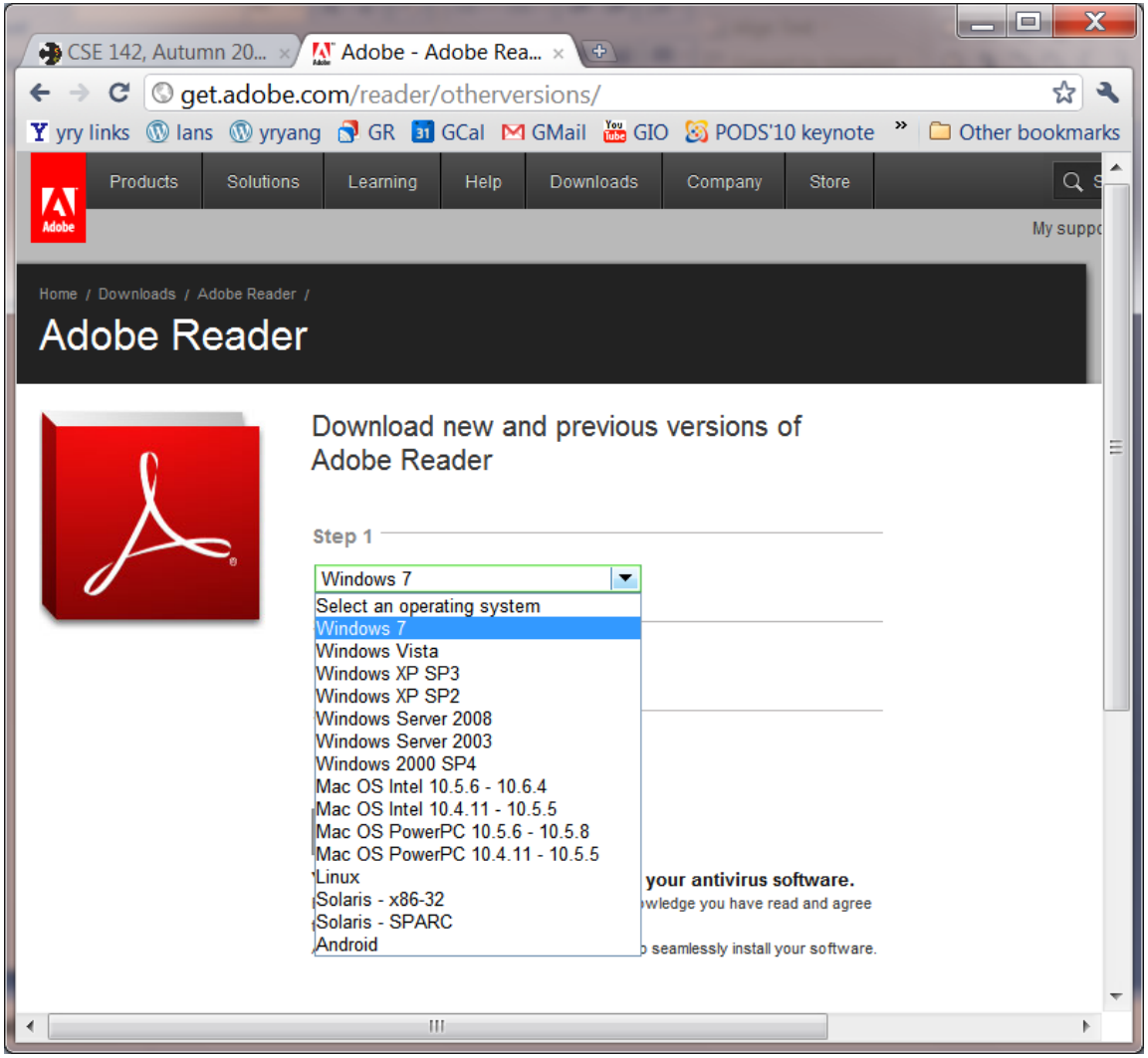


source code

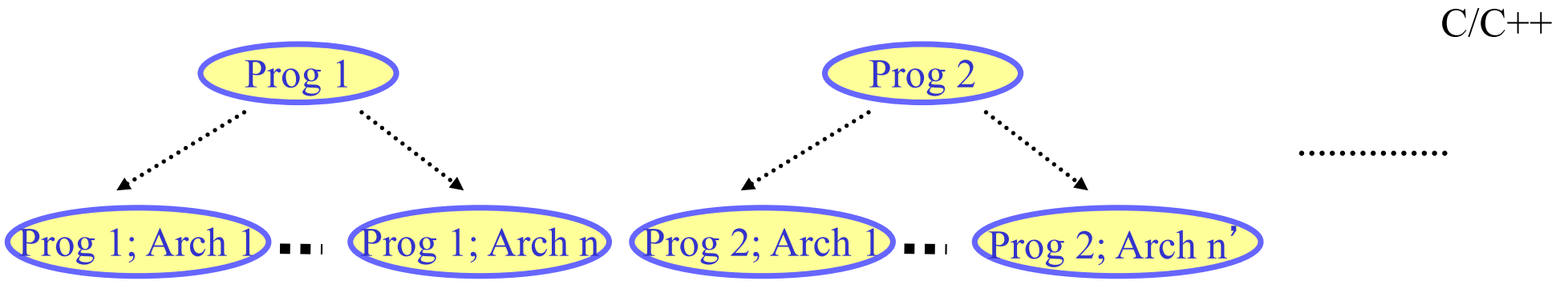
intel x86 +Win8

Problems of Compiling to Each Specific Computer Platform

- Multiple versions of the same software



High-level Picture



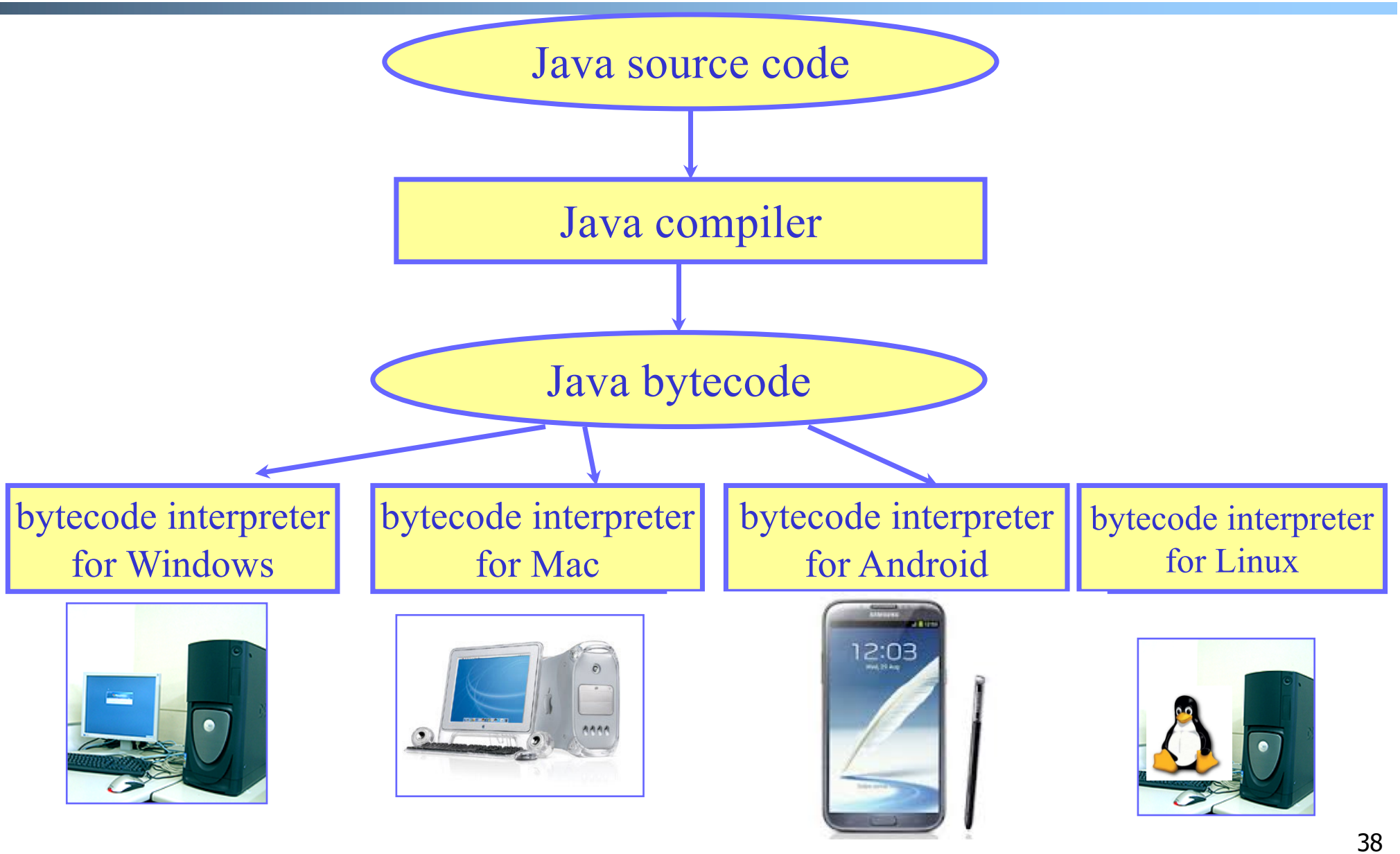
Java Virtual Machine

- 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



Comparing Traditional (e.g., C/C++) and Java Software Development

Traditional, e.g., C/C++

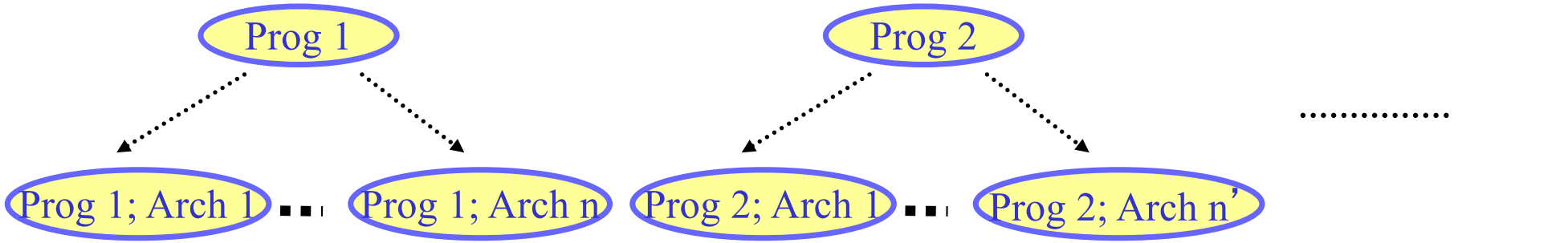
- A developer writes a program in C/C++
- The C/C++ source code is generally considered proprietary, and not released
- The developer compiles the C/C++ program for each platform it intends to support, and distributes one version for each platform
  - thus each program has multiple compiled versions
  - each compiled version can run by itself
- Platform dependency handled by each software developer

Java

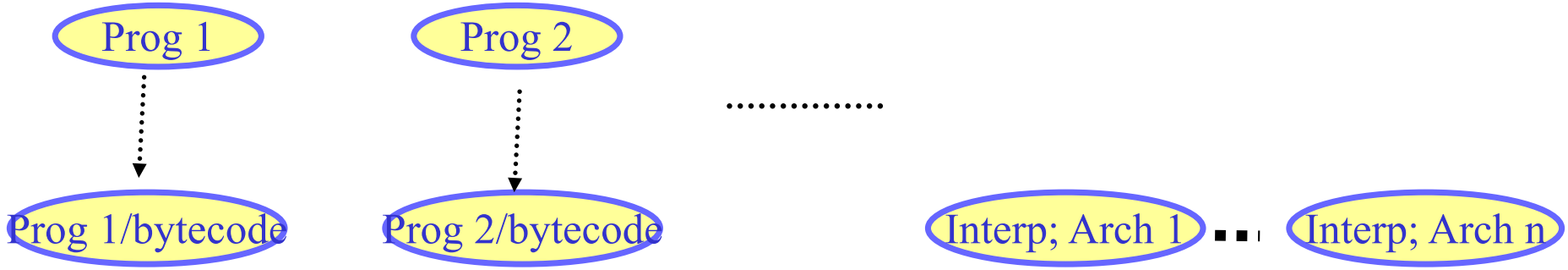
- A developer writes a program in Java
- The Java source code is generally considered proprietary, and not released
- The developer compiles the Java program to bytecode, and distributes the bytecode version
  - thus each program has only one compiled version
  - the compiled bytecode needs an interpreter for each platform
- Platform dependency handled by platform vendor

High-level Picture

C/C++



Java

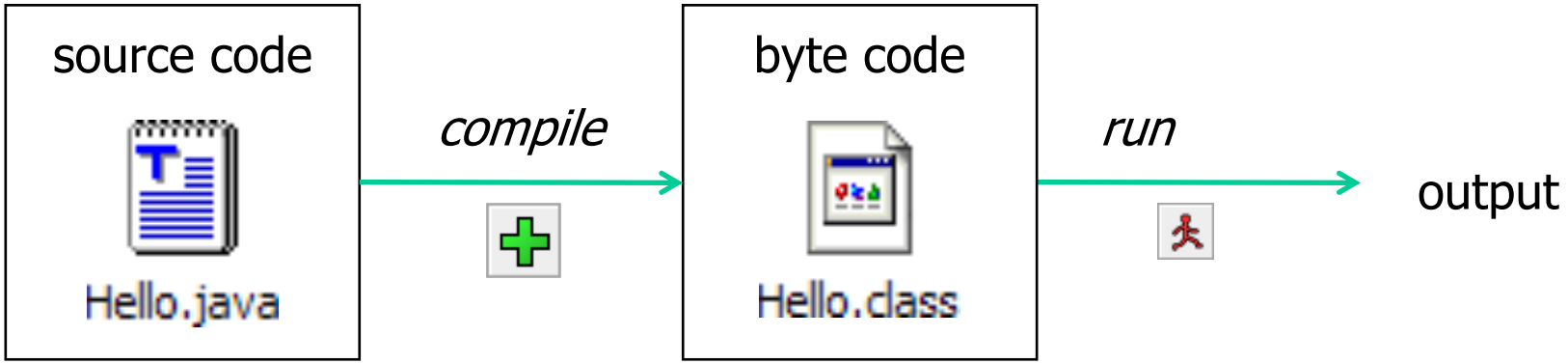


Outline

- Admin. and recap
- Computational process and development
- 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!");
    }
}
```

Another Java Program

```
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");
    }
}
```

Outline

- Admin. and recap
- Computational process and development
- 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
- Computational process and development
- Java: the programming language
- Programming levels
- Java programming steps
- Java program structure
  - A top-down view
  - A bottom-up view

Java Syntax: 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

```
// 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");
    }
}
```

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

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`)

```
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.

abstract	default	if	private	this
boolean	do	implements	protected	throw
break	double	import	**public**	throws
byte	else	instanceof	return	transient
case	extends	int	short	try
catch	final	interface	**static**	**void**
char	finally	long	strictfp	volatile
class	float	native	super	while
const	for	new	switch	
continue	goto	package	synchronized	

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:

```
System.out.println("\\hello\nhow\tare \"you\"?\\\\");
```

  - Output:

```
\hello
how     are "you"?\\
```

Comment on syntax errors

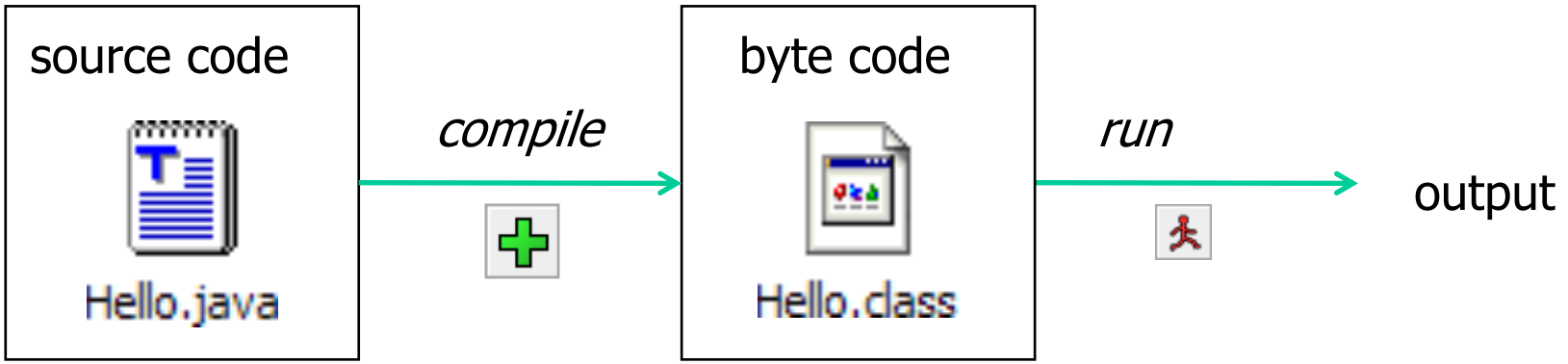
- **A syntax/compile error**: A problem in the structure of a program that causes the compiler to fail, e.g.,
  - Missing semicolon
  - Too many or too few { } braces
  - Class and file names do not match
  - …
- Compilers can't (DO not) read minds.
- Compilers don't make mistakes.
- If the program is not doing what you want, do NOT blame the computer---it's **YOU** who made a mistake.

Outline

- Admin.
- von Neumann Model: a Symbol Manipulation Platform
- Java methods

Recap: Java Programming Steps

- Programming in Java consists of three tasks
  - edit java source code (.java files)
  - compile java source code to generate bytecode (.class files)
  - execute/run/test bytecode using an interpreter



Recap: Top-Down Java Syntax Structure

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 ;

Recap: The `System.out.println` Statement

- Two ways to use the statement:
  - `System.out.println("string");`
    - You may need to use escape sequences in strings
  - `System.out.println();`

- A related statement is
  `System.out.print("string");`
  It does not print a newline

Java Syntax: A Bottom-Up Look

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

```
// This is a one-line 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 Look

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

Java depends on the identifiers and symbols to understand your program

```
public class Hello {
    [redacted]
}
```

Syntax Error: Example

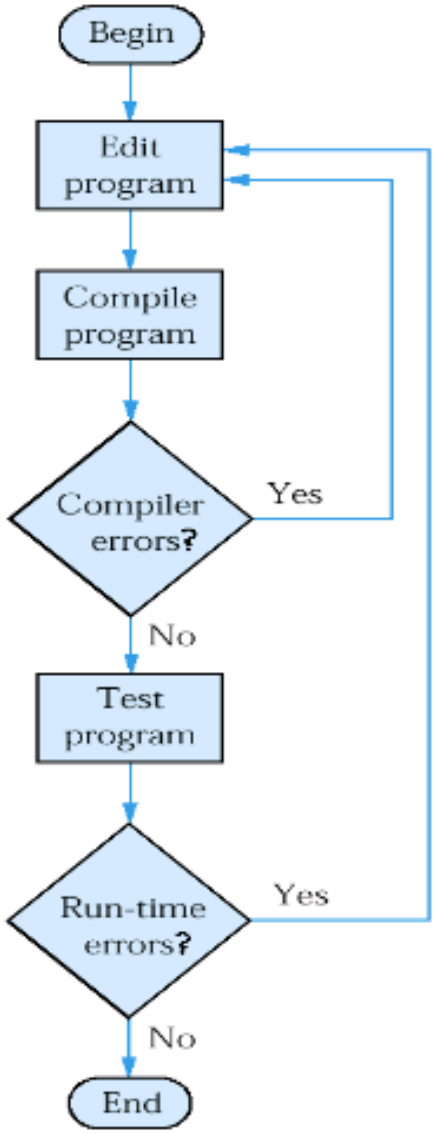
```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")_
4      }
5  }
```

- Compiler output:

```
Hello.java:2: <identifier> expected
    pooblic 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

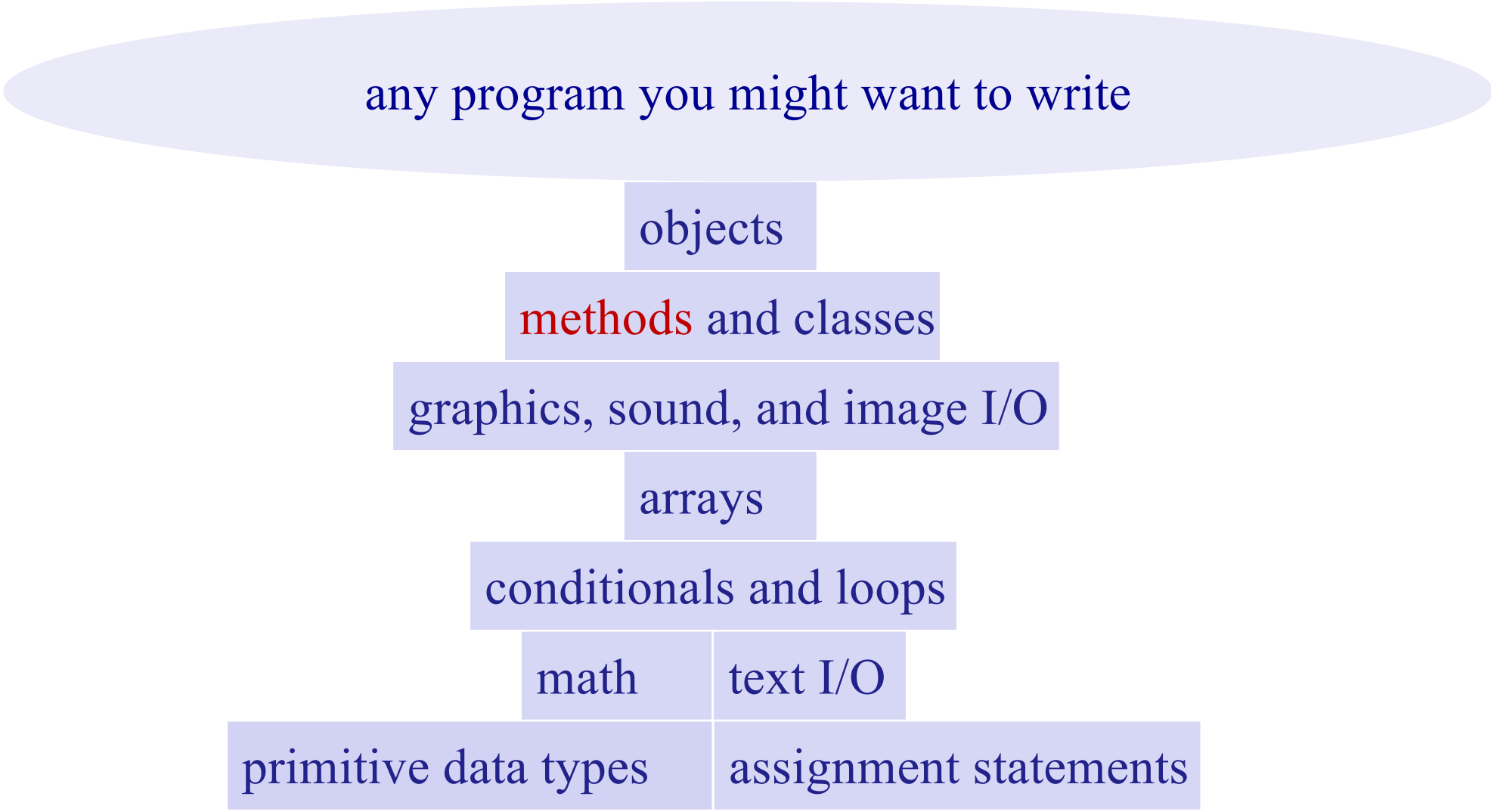


- *Compile-time errors*
  - the compiler may find problems with syntax and other basic issues
  - if compile-time errors exist, an executable version of the program is not created
- *Run-time errors*
  - a problem can occur during program execution, such as trying to divide by zero, which causes a program to terminate abnormally (crash)
- *Logical errors*
  - a program may run, but produce incorrect results

Programming in Java: IDE

- Professional programmers typically use an Integrated Development Environment (IDE)
  - Example IDEs: Eclipse, IntelliJ, 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, and we will arrange sessions to review how to use the Eclipse IDE

Roadmap


any program you might want to write
objects
methods and classes
graphics, sound, and image I/O
arrays
conditionals and loops
math
text I/O
primitive data types
assignment statements

Reading and Practice Slides (Out of Class)

Questions

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

```
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:

```
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.\"");
```

- `println` statements to generate the output:

```
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:

```
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:

```
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

```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")_
4      }
5  }
```

Syntax Error: Example

```
1  public class Hello {
2      pooblic static void main(String[] args) {
3          System.owt.println("Hello, world!")_
4      }
5  }
```

- Compiler output:

```
Hello.java:2: <identifier> expected
    pooblic 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??

End Outside Slides

Backup Slides

Assembly Languages

- 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

```
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
```

Example assembly code fragment

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
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *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...)