Java

David Lu

About Me

- David Lu
- Software Engineering @ University of Waterloo, class of 2020
- Full Stack Developer @ Mattermost Inc.
- Fun facts:
  - My dream car is a F-150 pickup truck
  - Dogs are adorable
  - I drink way too much coffee

Evaluations

- Twice-Weekly Homework Assignments
  - Submitted online
- Monthly Tests
  - Submitted in class
- All feedback will be provided online

Notes/Homework/Discussion

- olympiads-david.slack.com
- Slack is a team-based messaging service
- Notes will be posted in **#java** prior to each class
- Homework must be submitted to **@homeworkbot** as a Direct Message
- Discussion can occur in **#general** or **#java**
- Sign up for Slack by filling out this form: **http://tinyurl.com/zjsallu**

Class Schedule

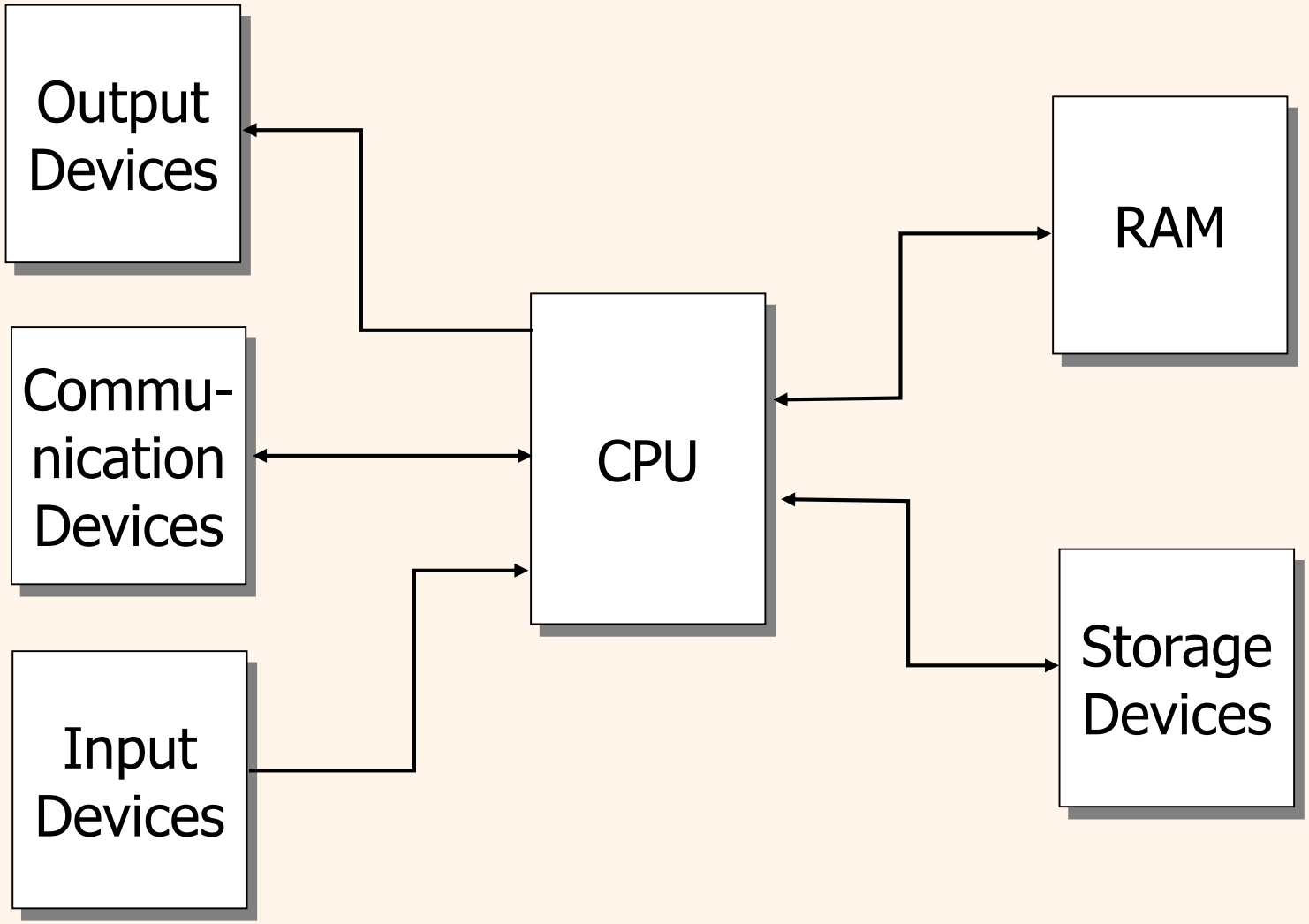
- Homework questions from previous class (not today)
- Java Concepts
- Practice Questions

Introduction to Computers
and
Programming Languages

A History of Computers

- Charles Babbage is credited as the father of computer. Although never actually built, he proposed the computing machines called Difference Engine and Analytical Engine that possessed the core characteristics of today's computers.
- Ada Lovelace, who wrote demonstration programs for Analytical Engine, is credited as the first programmer.
- The first modern computer was built by Atanasoff of Iowa State University in the late 1930s.
- An electromechanical computer MARK I was built by Howard Aiken of Harvard.
- The first completely electronic computer ENIAC I was built by Mauchly and Eckert of the University of Pennsylvania.

Computer Architecture



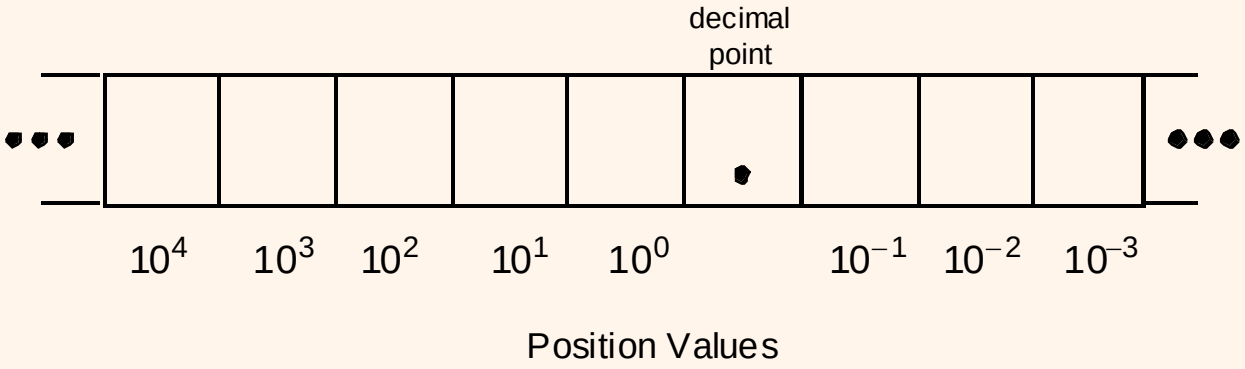
Progress of CPU Speed

	CPU	Date Introduced	Clock Speed (MHz)
1970s	4004	11/15/71	0.108
	8008	4/1/72	0.200
	8080	4/1/74	2
	8088	6/1/79	8
1980s	80286	2/1/82	12
	80386SX	6/16/88	16
	80486DX	4/10/89	25
1990s	Pentium	3/22/93	66
	Pentium Pro	11/1/95	200
	Pentium II	5/7/97	300
	Pentium II Xeon	6/29/98	400
	Pentium III	10/25/99	733
2000s	Xeon	9/25/01	2000
	Pentium 4	4/27/01	2000
	Itanium 2	7/8/02	1000
	Pentium 4 Extreme Edition	2/2/04	3400
	Core 2 Extreme	7/27/06	3200

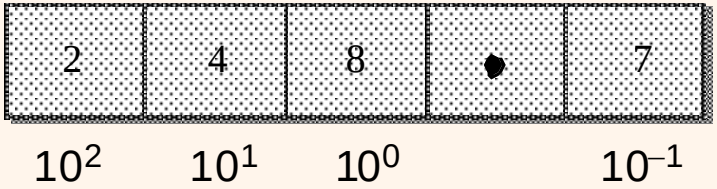
For more information on Intel CPUs, click Intel Museum

Decimal Number Representation

How the decimal number is represented.



Example:



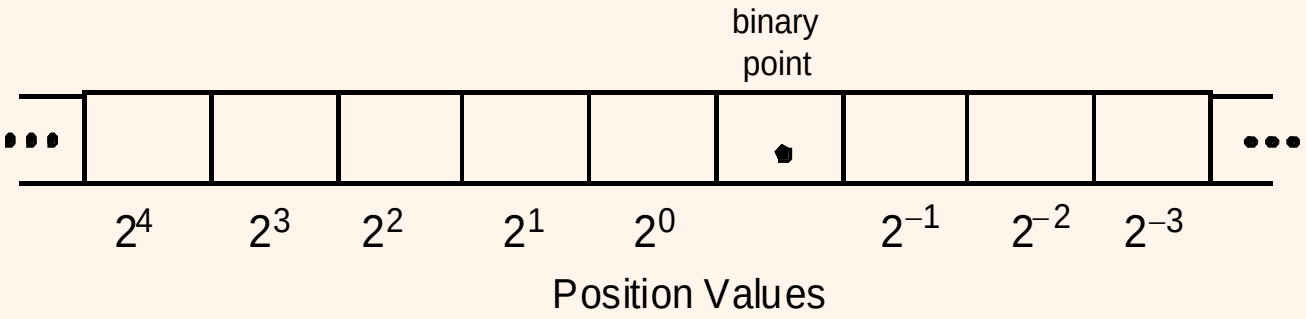
$= 2 \times 10^2 + 4 \times 10^1 + 8 \times 10^0 + 7 \times 10^{-1}$

$= 2 \times 100 + 4 \times 10 + 8 \times 1 + 7 \times 1/10$

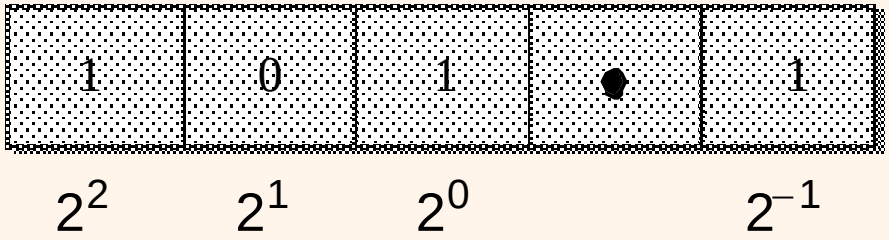
$= 200 + 40 + 8 + 7/10 = 248.7$

Binary Number Representation

How the binary number is represented.



Example:



$$= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1}$$

$$= 1 \times 4 + 0 \times 2 + 1 \times 1 + 1 \times 1/2$$

$$= 4 + 0 + 1 + 1/2 = 5.5$$

Programming Languages

- Three levels of programming languages:
  - Machine Languages
    - Machine language instructions are binary coded and very low level.
  - Assembly Languages
    - Assembly language allows symbolic programming. Requires an assembler to translate assembly programs into machine programs.
  - High-level Languages
    - High-level language provides a very high conceptual model of computing. Requires a compiler to translate high-level programs into assembly programs.

Java

- Java is a high-level object-oriented language developed by Sun Microsystems.
- So who developed Java ?

  a) Indonesian b) American c) Canadian d) English
- Java's clean design and wide availability make it an ideal language for teaching the fundamentals of computer programming.

Java Programs
Unit 1

Introduction to Java Programming

What is programming?

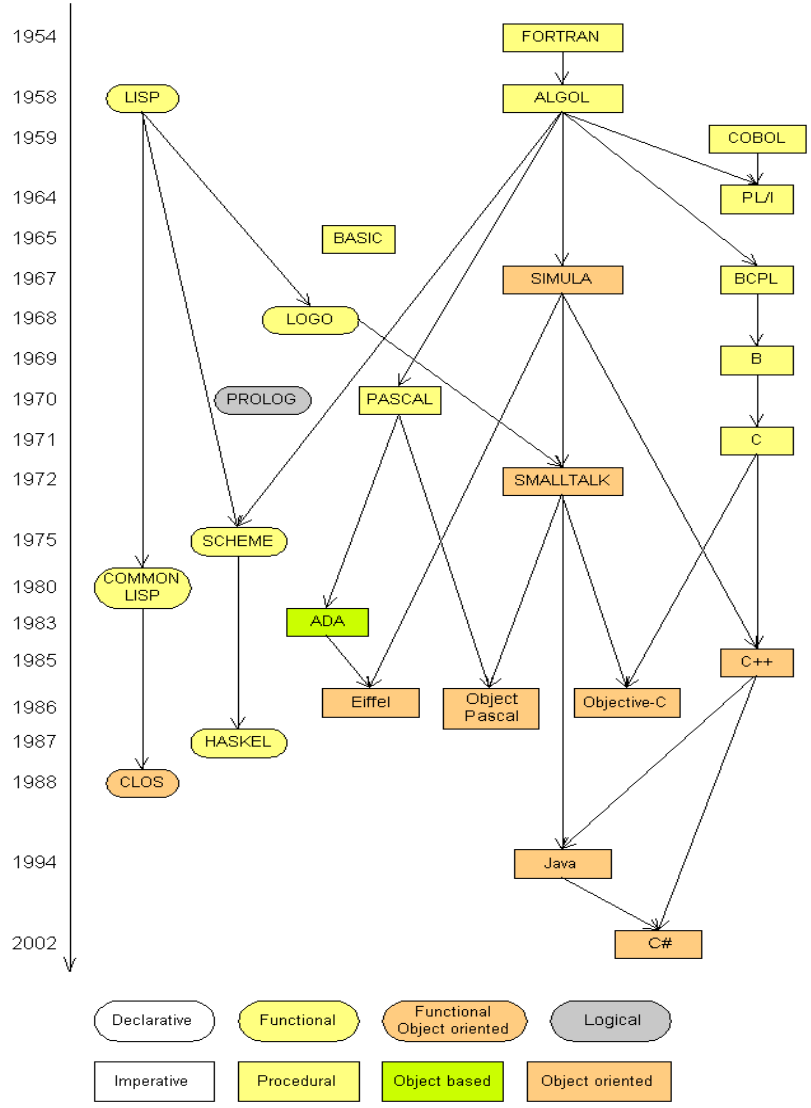
- **program**: A set of instructions to be carried out by a computer.

- **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.
  - This textbook teaches programming in a language named Java.

Programming languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



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"
  - **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...)
    - The language taught in this textbook

Some modern languages

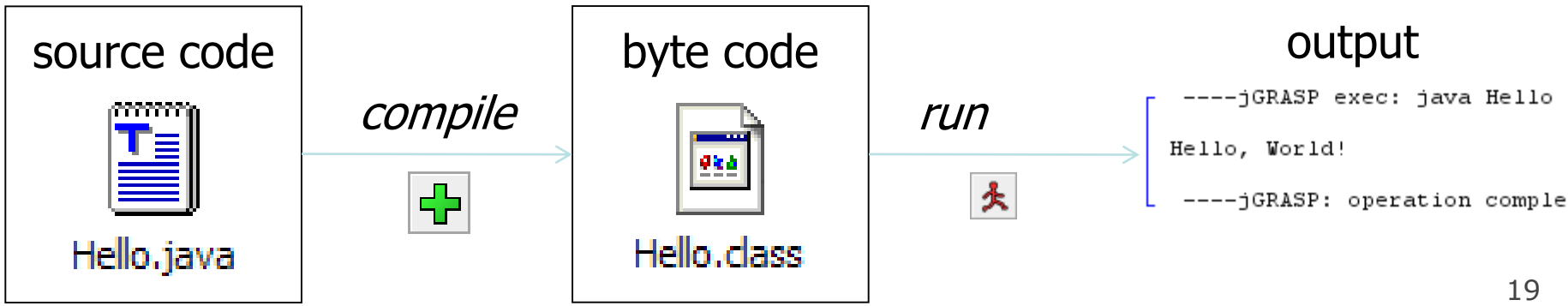
- *procedural languages*: programs are a series of commands
  - **Pascal** (1970): designed for education
  - **C** (1972): low-level operating systems and device drivers
- *functional programming*: functions map inputs to outputs
  - **Lisp** (1958) / **Scheme** (1975), **ML** (1973), **Haskell** (1990)
- *object-oriented languages*: programs use interacting "objects"
  - **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...)
    - The language taught in this textbook

Compile/run a program

1. *Write* it.
   - **code** or **source code**: The set of instructions in a program.

2. *Compile* it.
   - **compile**: Translate a program from one language to another.
   - **byte code**: The Java compiler converts your code into a format named *byte code* that runs on many computer types.

3. *Run* (execute) it.
   - **output**: The messages printed to the user by a program.



Basic Java programs with `println` statements

Displaying String Data

A Java program

```
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");
    }
}
```

- Its output:

```
Hello, world!

This program produces
four lines of output
```

- **console**: Text box into which the program's output is printed.

Structure of a Java program

```
public class name {
    public static void main(String[] args) {
        statement;
        statement;
        ...
        statement;
    }
}
```

class: a program

method: a named group of statements

statement: a command to be executed

- Every executable Java program consists of a **class**,
  - that contains a **method** named `main`,
    - that contains the **statements** (commands) to be executed.

System.out.println

- A statement that prints a line of output on the console.
  - pronounced "print-linn"
  - sometimes called a "println statement" for short

- Two ways to use `System.out.println`:
  - `System.out.println("`**`text`**`");`

    Prints the given message as output.

  - `System.out.println();`

    Prints a blank line of output.

Names and identifiers

- You must give your program a name.

```
public class GangstaRap {
```

  - Naming convention: capitalize each word (e.g. `MyClassName`)
  - Your program's file must match exactly (`GangstaRap.java`)
    - includes capitalization (Java is "case-sensitive")

- **identifier**: A name given to an item in your program.
  - must start with a letter or `_` or `$`
  - subsequent characters can be any of those or a number
    - legal: `_myName` `TheCure` `ANSWER_IS_42` `$bling$`
    - illegal: `me+u` `49ers` `side-swipe` `Ph.D's`

Keywords

- **keyword**: An identifier that you cannot use because it already has a reserved 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
```

Syntax

- **syntax**: The set of legal structures and commands that can be used in a particular language.
  - 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.
  - Missing semicolon
  - Too many or too few `{ }` braces
  - Illegal identifier for class name
  - 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  }
```

- 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 can be tough to understand!

Strings

- **string**: A sequence of characters to be printed.
  - Starts and ends with a " quote " character.
    - The quotes do not appear in the output.
  - Examples:

```
"hello"
"This is a string.  It's very long!"
```

- Restrictions:
  - May not span multiple lines.

```
"This is not
a legal String."
```

  - May not contain a " character.

```
"This is not a "legal" String either."
```

Escape sequences

- **escape sequence**: A special sequence of characters used to represent certain special characters in a string.

`\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"?\\
```

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 \"");
```

Comments

- **comment**: A note written in source code by the programmer to describe or clarify the code.
  - Comments are not executed when your program runs.

- Syntax:

```
//  comment text, on one line
```

or,

```
/*  comment text; may span multiple lines */
```

- Examples:

```
// This is a one-line comment.

/* This is a very long
   multi-line comment. */
```

Using comments

- Where to place comments:
  - at the top of each file (a "comment header")
  - at the start of every method (seen later)
  - to explain complex pieces of code

- Comments are useful for:
  - Understanding larger, more complex programs.
  - Multiple programmers working together, who must understand each other's code.

Comments example

```
/* Suzy Student, CS 101, Fall 2019
   This program prints lyrics about ... something. */

public class BaWitDaBa {
    public static void main(String[] args) {
        // first verse
        System.out.println("Bawitdaba");
        System.out.println("da bang a dang diggy diggy");
        System.out.println();

        // second verse
        System.out.println("diggy said the boogy");
        System.out.println("said up jump the boogy");
    }
}
```