# Java By Abstraction: Chapter 1

## Introduction to Programming

Some examples and/or figures are used with permission from slides prepared by Prof. H. Roumani

# Programming Style

- Use comments
  - Communicate a higher-level understanding of code
  - Comments are ignored by Java
  - // single-line comments
  - /*
    Multi-line comments
    */
- Follow Style Guide (Appendix C)

# Anatomy of a Java Program

- Code block
  - Defines program scope (i.e., boundaries)
  - Delimited by { and } aligned vertically
  - Indent all content (including inner blocks)
  - Example:

```
{
    {
        . . .
    }
}
```

# Anatomy of a Java Program

- Class definition
  - Defines your program's name
  - Represents outermost scope
  - Class name should always begin with a capital letter
  - public class *ClassName* // class header

    {

    . . . // class body

    }

# Anatomy of a Java Program

- Main method definition
  - Entry point of your program
  - Indented from class scope
  - public static void main(String[] args)  // header

```
{
    . . .                                  // body
}
```

# Anatomy of a Java Program

- Statements
  - Instructions to declare variables, assign values, use classes, and control execution flow
  - Example

```
System.out.println("Hello");
```

# Simple Program

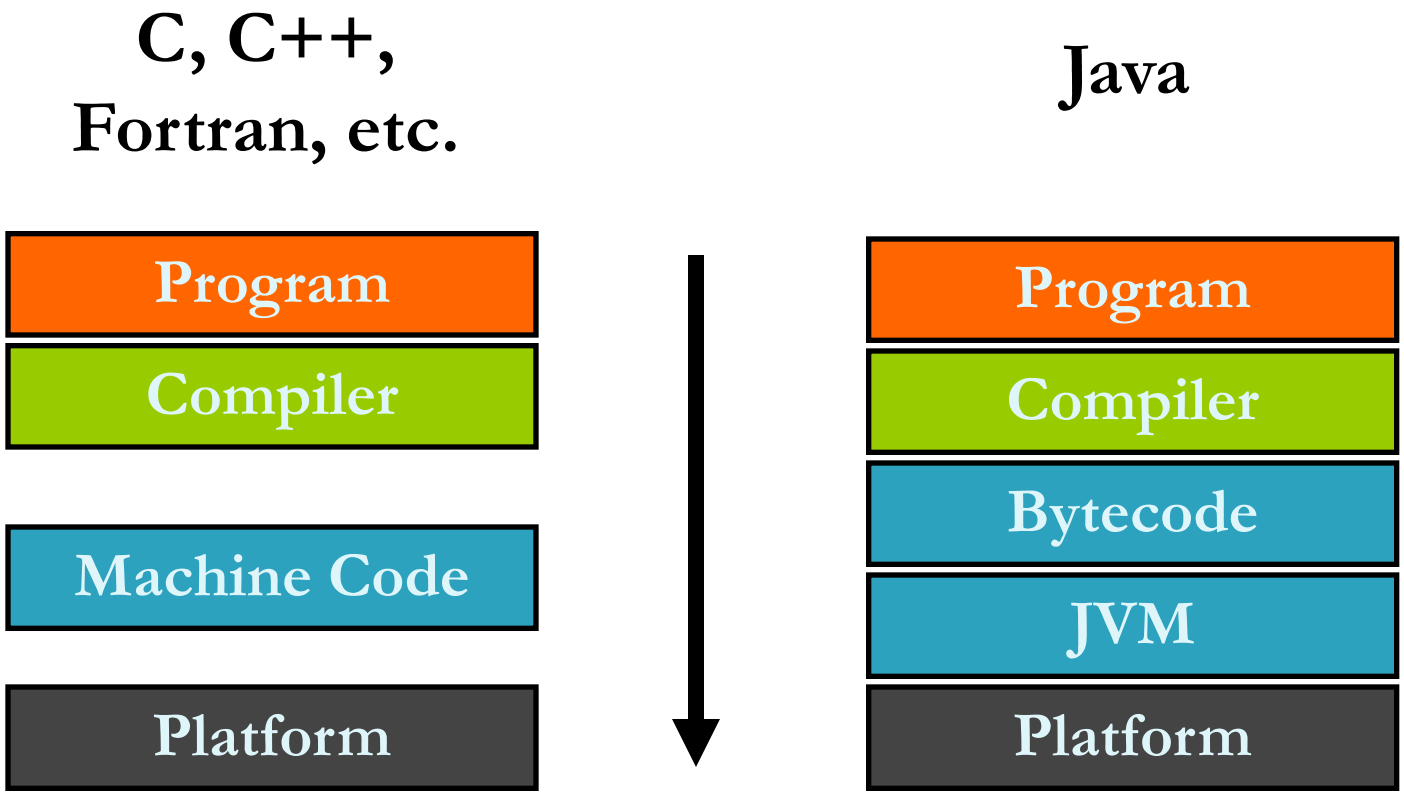
- Text file called First.java:

```
public class First
{
  public static void main(String[] args)
  {
    System.out.println("Hello");
  }
}
```

# Program Execution

- Using a text editor, create a text file with the class name and the extension .java (e.g., First.java)
- Compile the program using the javac command (e.g., javac First.java)
- Correct any syntax (runtime) errors (remember to save changes) and recompile
- A successful compilation will generate a class file
  (e.g., First.class)
- Enter the command java, followed by your program's class name (e.g., java First)

# Java Virtual Machine



# Primitive Data Types

- Integers
  - int (4 bytes): $[-2 \times 10^9 \,..\, 2 \times 10^9)$
  - long (8 bytes): $[-9 \times 10^{18} \,..\, 9 \times 10^{18})$
- Reals
  - float (4 bytes): $[-3.4 \times 10^{38} \,..\, 3.4 \times 10^{38})$, 7 sig. digits
  - double (8 bytes): $[-1.7 \times 10^{308} \,..\, 1.7 \times 10^{308})$, 15 sig. digits
- Characters
  - char (2 bytes): Unicode characters 0x0000 to 0xFFFF
- Boolean
  - boolean: (1byte): true or false

# Declaring Variables

- A variable's value can change during execution
- Declaration

  *primType identifier = value*;

  Where:

  *primType* is int, long, float, double, etc.
  *identifier* is the name you choose for the variable
  *value* is the value you want the variable to have
- Example

  int currentTemperature = 8;

# Declaring Constants

- A constant's value does not change during execution
- Declaration

  final *primType IDENTIFIER = value*;

  Where:

  *primType* is int, long, float, double, etc.

  *IDENTIFIER* is the name (all caps) you choose

  *value* is the value you want the constant to have
- Example

  final int INCHES_PER_FOOT = 12;

# Keywords

- Have special meanings in Java
- Not to be used as identifiers, class names, etc.

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

p. 8 in text

# Logical Operators

- AND &&
- OR ||
- Equal ==
- Not equal !=
- Less than <
- Less than or equal <=
- Greater than >
- Greater than or equal >=

# Arithmetic Operators

- Add +
- Subtract -
- Multiply *
- Divide /
- Remainder %

# Integer Division

- Satisfies the closure property
- When dividing an integer by another, the result in also an integer
- Ignore any remainder
- Don't confuse integer division with real division
  - Integer: 2 / 3 = 0, 5 / 3 = 1
  - Real: 2.0 / 3.0 = 2.0 / 3 = 2 / 3.0 = 0.66666…

# Operator Precedence

- Arithmetic operator precedence similar to order of operations learned in high school
  - Brackets
  - Multiplication and division (including remainder)
  - Addition and subtraction
- Left-to-right association
- Full details in Appendix B
- Also see Programming Tip 1.11 on p. 31 in text

# Example

```
5 + (4 - 3) / 5 - 2 * 3 % 4
```

# Example

```
  5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```

# Example

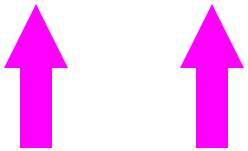
```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
```



# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

# Example

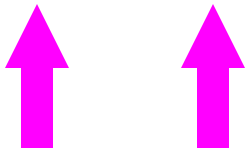
```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
```



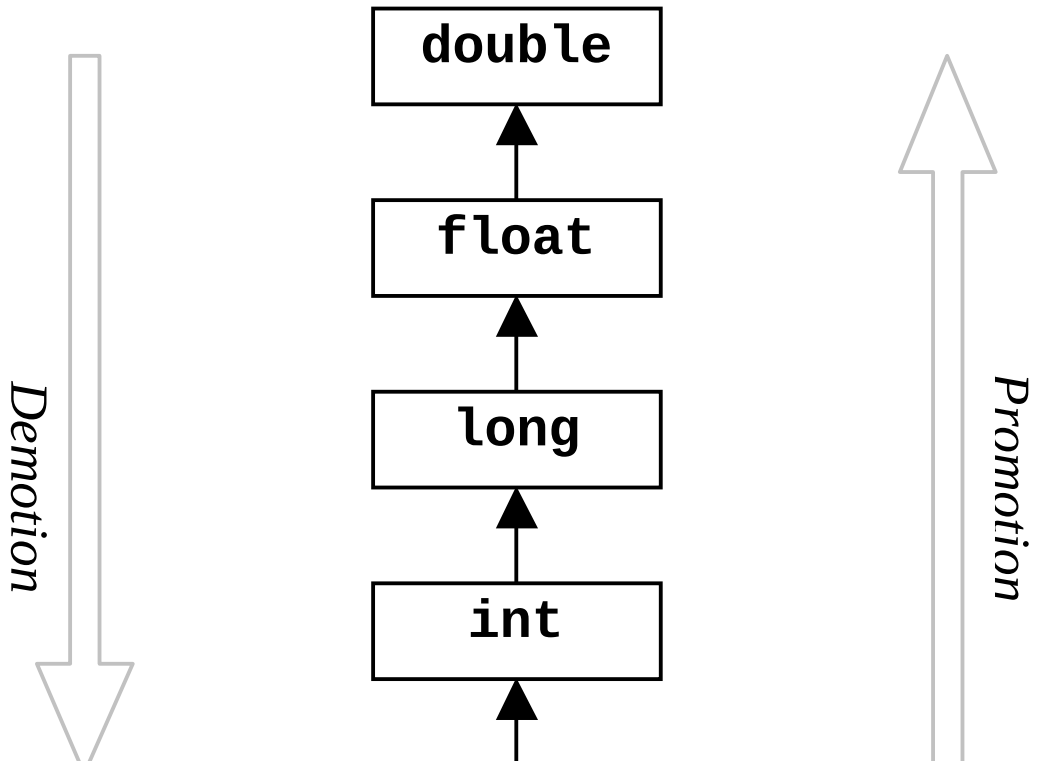
# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
```

# Example

```
   5 + (4 - 3) / 5 - 2 * 3 % 4
= 5 + 1 / 5 - 2 * 3 % 4
= 5 + 0 - 2 * 3 % 4
= 5 + 0 - 6 % 4
= 5 + 0 - 2
= 5 - 2
= 3
```

# Promotion



# Casting

- Returns compile-time exception

```
double aDbl = 5.0;
int bInt = 2;
int result = aDbl * bInt;
```

- Demotion accomplished with casting

```
double aDbl = 5.0;
int bInt = 2;
int result = (int) aDbl * bInt;
```

- Promotion via casting to force real division

  2 / 3 = 0, (double) 2 / 3 = 2 / (double) 3 = 0.66666…

# Lexical Elements