Basic Computation – Part 1

Forest Agostinelli
University of South Carolina

Outline

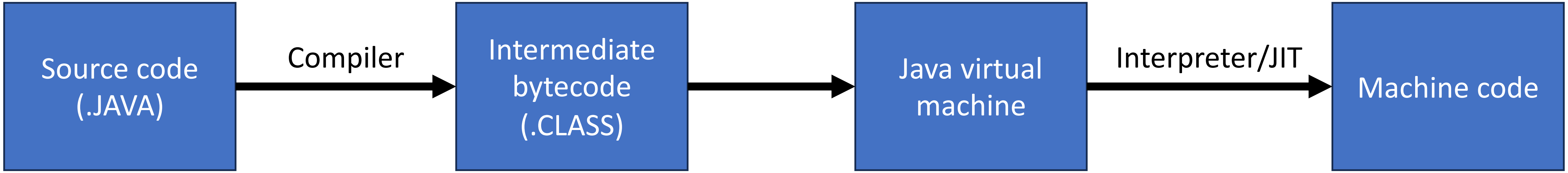
- Review
- Variables
- Primitive types
- Class types (only brief notes)
- Constants, Math operators, I/O

Terminology: Compiled vs Interpreted Languages

- **Compiled languages**
  - Convert code to assembly or machine code through a process called compilation
  - Examples: C++, Java
- **Interpreted Languages**
  - Call precompiled code based on the code written in the high-level language
  - Examples: Python, Perl

Java Virtual Machine

- Java source code is compiled into an intermediate java bytecode
  - The java bytecode will result in the same execution in the Java virtual machine (JVM) across physical machines
- A machine-specific combination of an interpreter and a just-in-time (JIT) compiler are used to convert the bytecode to machine code



Outline

- Review
- Variables
- Primitive types
- Class types (only brief notes)
- Constants, Math operators, I/O

Variables

- Variables store data
- The data that they store can then be used for computation
- The Java syntax for declaring variables: <<type>> <<identifier>>
    - Note: “<<here>>” means put something in place of this
    - For example: int numCats;
    - Only specify the type when declaring the variable

Identifiers

- An **identifier** is a name, such as the name of a variable.
- Identifiers should be meaningful
- Identifiers may contain ONLY
  - Letters
  - Digits (0 through 9)
  - The underscore character (_)
  - And the dollar sign symbol ($) which has a special meaning

- Identifiers CANNOT contain
  - Spaces of any kind
  - Digit as the First Character
  - Dots "."
  - Asterisks "*"
  - Other types of special characters
- Identifiers are Case Sensitive
  - "Stuff", "stuff", "STUFF", and "sTuFf" would all be considered different identifiers
- Identifiers CANNOT be a **reserved word**
  - Example Reserved Words: int, public, class

Identifiers

Naming Conventions

- Class Types start with an Uppercase character
  - Example: String
- Primitive Types start with a Lowercase character
  - Example: int
- Variables identifiers of both start with a Lowercase Character
- Multiword identifiers are "punctuated" using uppercase characters

Good Examples

```
int test01;
double largeValues;
boolean inClass;
```

Bad Examples

```
int 1Test;//Started with a digit
double big vals;//Used a space
boolean class;//Class is a reserved word
```

Terminology: Static vs Dynamic Typing

- **Static Typing**: The type of a variable is known at compile time and cannot change during runtime
  - Type errors are caught at compile time
  - C, C++, Java
- **Dynamic Typing:** The type of a variable may or may not be known at compile time and can change during runtime
  - Type errors can occur during runtime
  - More flexibility
  - Python, Perl, MATLAB
  - Python now has type hinting, which does not have an effect on runtime, but can be used with IDEs to prevent catch errors before runtime

Java Types

- **Primitive Types**
  - Atomic/irreducible
  - No methods
  - Identifiers contain the assigned value
- **Class Types**
  - Are composed of other types
  - Can have class methods
  - Identifiers are references to the class object

Outline

- Review
- Variables
- Primitive types
- Class types (only brief notes)
- Constants, Math operators, I/O

Primitive Types

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	**4 bytes**	**Stores whole numbers from -2,147,483,648 to 2,147,483,647**
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	**8 bytes**	**Stores fractional numbers. Sufficient for storing 15 decimal digits**
boolean	**1 bit**	**Stores true or false values**
char	**2 bytes**	**Stores a single character/letter or ASCII values**

- The name “floating-point” comes from the fact that the decimal point can be made to “float” to different places in a number in scientific notation
- What explains the range of the primitive number types?
  - Why the difference of 1 in the range of positive and negative numbers?
    - See two’s complement
- In what situation would using a byte be preferable to an int?
- What happens if one has a byte that is 127 and then adds 1 to it?

Wraparound

```java
byte val = 127;
System.out.println(val);
val++;
System.out.println(val);
```

Output

```
127
-128
```

- A byte is represented by 1 byte
- 127 is 01111111
- Adding one makes it 10000000, which is -128
- Not taking this into account can cause crucial errors in the logic of your code!

Variable Declaration

- One can declare variables without yet assigning them a value. They are then assigned a default value.

Example

```
int i;
double j;
char o;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	0.0	32
o	‘\u0000’	40
???	???	42
…	…	…

Variable Assignment

- The equals symbol "=" is the assignment operator
- Stores values found on the right hand side (RHS) of the operator into the identifier found on the left hand side (LHS)
- Assignments are valid if the type matches are is at least compatible
  - Primitive types can be stored in other primitive types as long the type's byte amount is less than or equal to value being stored
  - Otherwise "type casting" is required
  - Type casting does not round it cuts off everything past the decimal point "."
- Spoken:
  - "Store this value in this container"

Syntax

```
<<identifier>> = <<value>>;
```

Examples

```
i = 0;
j = 22.3;
o = ‘h’;
i = (int)j;//Type cast from double to int
//Value stored in “i” is 22
```

Variable Declaration and Assignment

- One can combine variable declaration and assignment into one statement
- This creates more compact code, and it is good programming practice to do so whenever possible

```
int i = 0;
double j = 22.3;
char o = 'h';
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	0	28
j	22.3	32
o	‘h’	40
…	…	…

Variable Declaration and Assignment

```
int i = 0;
double j = 22.3;
char o = 'h';
i = (int)j;
```

Memory

Identifier	Contents	Byte Address
…	…	…
i	22	28
j	22.3	32
o	‘h’	40
…	…	…

Outline

- Review
- Variables
- Primitive types
- Class types (only brief notes)
- Constants, Math operators, I/O

Class Types

- The variable to which a class is assigned does **not** hold the value of the class, but rather, a reference, which is the location in memory in which the object is stored
- This leads to significantly different behavior when working with class types as opposed to primitive types

Variable Assignment: Primitive Types

```
int a = 1;
int b = a;
```

Identifier	Value	Byte Address
a	1	4
b	1	8

```
b = b + 1;
```

Identifier	Value	Byte Address
a	1	4
b	2	8

Variable Assignment: Class Types

```java
public class IntegerMutable {
    int val;

    public IntegerMutable(int val) {
        this.val = val;
    }

    public void add(int val_add) {
        this.val = this.val + val_add;
    }
}
```

Variable Assignment: Class Types

```
IntegerMutable aClass = new IntegerMutable(1);
IntegerMutable bClass = aClass;
```

Identifier	Value	Byte Address
a	1024	4
b	1024	12
…	…	…
	1	1024

```
bClass.add(1);
```

Identifier	Value	Byte Address
a	1024	4
b	1024	12
…	…	…
	2	1024

Variable Assignment: Class Types

```
aClass = new IntegerMutable(3);
```

Identifier	Value	Byte Address
a	2048	4
b	1024	12
…	…	…
	2	1024
…	…	…
	3	2048

```
bClass = aClass;
```

Identifier	Value	Byte Address
a	2048	4
b	2048	12
…	…	…
	2	1024
…	…	…
	3	2048

- Now nothing is referring to memory location 1024
  - What happens to it?

Outline

- Review
- Variables
- Primitive types
- Class types (only brief notes)
- Constants, Math operators, I/O

Constants

- Establishes a value that cannot change
- MUST assign a value initially
- Great for avoiding “magic numbers”
- Good programming practice
  - Make the scope public
  - Make it static
  - Capitalize all characters in the identifier

Syntax

```
public static final <<type>> <<identifier>> = <<value>>;
```

Examples

```
public static final double PI = 3.14159;
public static final int BOARD_SIZE = 10;
```

Math Operators

- Performs computation and then assigns the results
- Order of Operations
- Basic Math Operations
  - Addition “+”
  - Subtraction “-”
  - Multiplication “*”
  - Division “/”
- Mod Operator “%”
  - Returns the remainder after division
  - Ex: 15 % 2 = 1

Syntax

```
<<identifier>> = <<value>> <<operator>> <<value>>;
```

The right-hand side is evaluated first and then assigned to the left-hand side.

Examples

```
//Variables
int value = 64 % i + 32;
//Constants
public static final double PI = 3.14159;
public static final double PI_SQ = PI*PI;
```

Compute and Assign Operators

- Compute and Assign (C&A) Operators
  - Shorthand for applying some operator and value to a variable
  - Same as:
    - <<identifier>> = <<identifier>> <<operator>> <<value>>;
    - Ex: i = i+1; i+=1; i++; //Same statements
- Common Versions
  - "+=" – add and assign
  - "-=" – subtract and assign
  - "*=" – multiply and assign
  - "/=" – divide and assign
  - "%=" – mod and assign
- Special versions
  - "++" – Increase by 1
    - Same as "+= 1"
  - "--" – Decrease by 1
    - Same as "-=1"

Syntax

```
<<identifier>> <<C&A operator>> <<value>>;
```

Examples

```
i += 128; //If i = 32 now it is 160
j %= 2; //If j = 28.0 now it is 0.0
```

Math Notes

- eNotation
  - Allows number to be written in scientific notation
  - Example: 865000000.0 can be written as 8.65e8
- Imprecision with Floating-Point Numbers
  - Floating point numbers are approximations as they are finite
  - Example: 1.0/3.0 is slightly less than 1/3 ergo 1.0/3.0 + 1.0/3.0 + 1.0/3.0 < 1.0
  - Logic Errors

- Integers are ALWAYS Integers
  - Anything past the decimal point is cut off
  - Also can be considered “rounding down” or “taking the floor”
  - Example: 1/3 = 0
  - Logic Error

Basic Input and Output (I/O)

- For now, input and output is done in the Console
- Command Line Interface
- Console Outputs (Writes)
  - Left to Right
  - Up to Down
- Console Inputs (Reads)
  - Left to Right
  - Up to Down

Syntax

```
System.out.println(<<value>>);
```

Examples

```
int i = 22;
System.out.println(i);
```

Basic Input and Output (I/O)

- System.out.println(<<argument>>);
  - Statement used to output the argument and adds a new line after
- System.out.print(<<argument>>);
  - Statement used to output the argument but stays on the same line
- "Prints" to the standard system output (the console)

Syntax

```
System.out.println(<<argument>>);
System.out.print(<<argument>>);
```

Examples

```
int i = 22;
System.out.println(i);
```

Basic Input and Output (I/O)

- Use Scanner to read from Console
- Must import type Scanner from “java.util” package
  - import java.util.Scanner;
- Create an instance of type Scanner that “scans” the standard system input
  - Scanner keyboard = new Scanner(System.in);
- Useful methods
  - next()
  - nextLine()
  - nextInt()
  - nextDouble()
- Also can be used to “scan” Strings, files, network traffic, etc.

Examples

```
Scanner keyboard = new Scanner(System.in);
String name = keyboard.nextLine();
int i = keyboard.nextInt();
keyboard.nextLine();//Useful “fix-up”
double j = keyboard.nextDouble();
keyboard.nextLine();//Useful “fix-up”
System.out.println(name+ “ “ + i + “ “ + j);
```

Console

```
JJ
64
3.14
JJ 64 3.14
```

Coding Example

InchesToFeet.java

```
/*
 * Written by JJ Shepherd
 */
import java.util.Scanner;
public class InchesToFeet {

    public static final double INCHES2FEET = 12.0;
    //Entry point
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Greetings! Give a height in feet, and I will give the number of
inches, and feet + inches");
        double feet;
        feet = keyboard.nextDouble();
        keyboard.nextLine();

        double inches = feet*INCHES2FEET;
        int iFeet = (int)(inches/INCHES2FEET);
        int rmInches = (int)(inches%INCHES2FEET);

        System.out.println("In "+feet+"ft there are "+inches+"in. or "+iFeet+"ft. and
"+rmInches+"in.");
    }

}
```