



borrowing from slides by Kurt Eiselt, Wolfgang Heidrich,
Alan Hu

<http://www.cs.ubc.ca/~tmm/courses/111-10>

News

- labs and tutorials start this week
- my office hours: Mon 4-5, or by appointment
 - in X661
- UBC CS news

Events this week

Drop-in Resume Edition

Date: Mon. Jan 11
Time: 11 am – 2 pm
Location: Rm 255, ICICS/CS

Industry Panel

Speakers: Managers from IBM, Microsoft, SAP, TELUS, Radical ...

Date: Tues. Jan 12

Time: Panel: 5:15 – 6:15 pm
Networking: 6:15 – 7:15 pm
Location: DMP 110 for panel,
X-wing upgrad lounge for networking

Tech Career Fair

Date: Wed. Jan 13
Time: 10 am – 4 pm
Location: SUB Ballroom

Google Tech Talk

Date: Wed, Jan 13
Time: 4 – 5 pm
Location: DMP 110

IBM Info Session

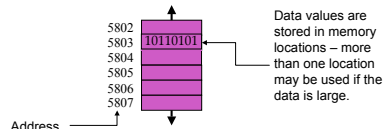
Date: Wed, Jan 13
Time: 5:30 – 7 pm
Location: Wesbrook 100

Reading This Week

- Chap 1: 1.3-1.8
- Chap 2: 2.1-2.2, 2.5
- Chap 4: 4.1-4.2

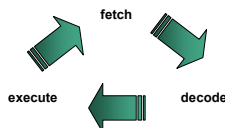
Review: Memory

- Memory consists of a series of locations, each having a unique address, that are used to store programs and data.
- When data is stored in a memory location, the data that was previously stored there is overwritten and destroyed.
- Each memory location stores one byte (or 8 bits) of data.
 - Each bit is a 0 or a 1
 - More on this soon

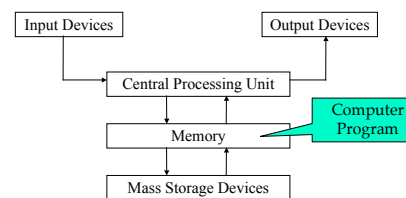


Review: Central Processing Unit

- CPU executes instructions in a continuous cycle
 - known as the "fetch-decode-execute" cycle
- CPU has dedicated memory locations known as **registers**
 - One register, the **program counter**, stores the address in memory of the next instruction to be executed



Review: Computer Programming



Review: Machine Language

- First programming languages: **machine languages**
 - Most primitive kind
 - Sample machine language instruction
 - Register: special purpose memory location inside CPU where real computation occurs
- ```

00000000000100010001001100000000000000000000
add | what's in this register | to what's in this register | and put it in this register | unimportant details for us

```
- Difficult to write programs this way
    - People created languages that were more readable

## Review: Assembly Language

- Next: **assembly languages**
  - Direct mappings of machine language instructions into helpful mnemonics, abbreviations
- Sample assembly language instruction
  - Corresponds to machine language instructions

add r1,r2,r6

```

00000000000100010001001100000000000000000000
add | what's in this register | to what's in this register | and put it in this register | unimportant details for us

```

## Review: Binary vs. Decimal Numbers

- decimal system numbers
  - have digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  - read from right to left:
    - ones (10<sup>0</sup>), tens (10<sup>1</sup>), hundreds (10<sup>2</sup>), thousands (10<sup>3</sup>), ...
    - ex: 4763 means 3\*10<sup>0</sup>+6\*10<sup>1</sup>+7\*10<sup>2</sup>+4\*10<sup>3</sup>
    - the exponents count up from 0
- binary system numbers
  - have digits 0, 1
  - still read from right to left:
    - ones (2<sup>0</sup>), twos (2<sup>1</sup>), fours (2<sup>2</sup>), eights (2<sup>3</sup>), sixteens (2<sup>4</sup>), ...
    - ex: 10010111 means: 1\*2<sup>0</sup>+1\*2<sup>1</sup>+1\*2<sup>2</sup>+0\*2<sup>3</sup>+1\*2<sup>4</sup>+0\*2<sup>5</sup>+0\*2<sup>6</sup>+1\*2<sup>7</sup>
    - = 1+2+4+16+128 = 151

## Aside – Other Bases

- The same principle works for other bases
- For example, *hexadecimal* (base 16)
  - uses digits 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
  - A-F correspond to values 10-15
- Example:

C350

- Means:

$$0*16^0 + 5*16^1 + 3*16^2 + 12*16^3 = 5*16 + 3*256 + 12*4096 = 50,000$$

## Assembly Language

- Assembly language program converted into corresponding machine language instructions by another program called an **assembler**



add r1,r2,r6

```

00000000000100010001001100000000000000000000
add | what's in this register | to what's in this register | and put it in this register | unimportant details for us

```

## Assembly Language

- Both machine and assembly languages pose big challenges for programmers
  - Difficult to read and write
  - Difficult to remember
- Each instruction does very little
  - Takes lots of instructions just to get something simple done
- Every machine or assembly language good for only one type of computer
  - Different to program IBM than Honeywell than Burroughs...

## High-Level Language

- Next step: development of high-level languages
- You may have heard of some
  - Fortran, COBOL, Lisp, BASIC, C, C++, C#, Ada, Perl, Java, Python, Ruby, Javascript
- High-level languages intended to be easier to use
  - still a long way from English.
- A single high-level instruction gets more work done than a machine or assembly language instruction.
- Most high-level languages can be used on different computers

## Java

- Java is the high-level language we'll use.
  - Modern, widely used, portable, safe.
- Developed by Sun in early 1990s
  - Originally intended for set-top boxes
  - Retargeted for the Web

## High-Level Language

- Example of a high-level instruction
  - A = B + C
- Tells computer to
  - go to main memory and find value stored in location called B
  - go to main memory and find value stored in location called C
  - add those two values together
  - store result in memory in location called A

## High-Level Language

- Must be translated into machine language so the computer can understand it.

- High-level instruction:  $A = B + C$  becomes at least four machine language instructions!

```
0001000000100000000000000000000010 load B
0001000000100000000000000000000011 load C
00000000001000100011000000100000 add them
0001010011000000000000000000000001 store in A
```

- How?
  - You could translate it as you go (**interpreter**).
  - You could translate it in advance (**compiler**).

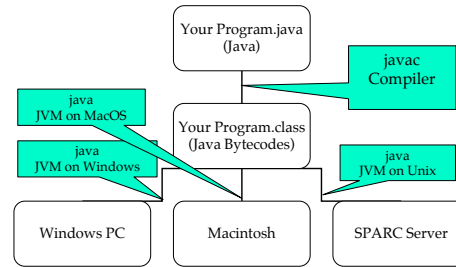
17

## Interpreters and Compilers

- An interpreter translates the high-level language into machine language on-the-fly, executing the instructions as it goes.
- A compiler translates the high-level language program all at once in advance.
- Both compilers and interpreters are themselves computer programs.
- Which is better?
  - Remember George and Stephen in France?

18

## Java Does Both!



19

## A Simple Java Program

```
// Our first Java program.
/* Traditionally, one's first program in a new
language prints out "Hello, World!"
*/
class HelloTester {
 public static void main(String[] args) {
 System.out.println("Hello, World!");
 }
}
```

20

## Sample Java Application Program

```
/*.....
// Oreo.java Author: Kurt Eiselt
//
// Demonstrating simple Java programming concepts while
// revealing one of Kurt's many weaknesses
//.....

public class Oreo
{
 /*.....
 // demand Oreos
 //.....
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

21

## Sample Java Application Program

- Comments ignored by Java compiler

```
/*.....
// Oreo.java Author: Kurt Eiselt
//
// Demonstrating simple Java programming concepts while
// revealing one of Kurt's many weaknesses
//.....

public class Oreo
{
 /*.....
 // demand Oreos
 //.....
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

22

## Sample Java Application Program

- Comments could also look like this

```
/*
Oreo.java Author: Kurt Eiselt
Demonstrating simple Java programming concepts while
revealing one of Kurt's many weaknesses
*/

public class Oreo
{
 /* demand Oreos */
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

23

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Comments are important to people
  - But not to the compiler
- Compiler only cares about

24

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Whole thing is the definition of a **class**
  - Package of instructions that specify
    - what kinds of data will be operated on
    - what kinds of operations there will be
  - Java programs will have one or more classes
    - For now, just worry about one class at a time

25

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Instructions inside class definition grouped into one or more procedures called **methods**
  - group of Java statements (instructions) that has name, performs some task
- All Java programs you create will have **main** method where program execution begins

26

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- These class and method definitions are incomplete at best
  - good enough for now
  - expand on these definitions as class continues

27

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Words we use when writing programs are called **identifiers**
  - except those inside the quotes

28

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Kurt made up identifier **Oreo**

29

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Other programmers chose identifier **System.out.println**
  - they wrote printing program
  - part of huge library of useful programs that comes with Java

30

## Sample Java Application Program

```
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

- Special identifiers in Java called **reserved words**
  - don't use them in other ways

31

## Reserved Words

- Get familiar with these
  - But you don't need to memorize all 52 for exam

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

32

## Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
  - Letters are the 26 characters in English alphabet
    - both uppercase and lowercase
    - plus the \$ and \_
    - also alphabetic characters from other languages

33

## Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
    - Letters are the 26 characters in English alphabet
      - both uppercase and lowercase
      - plus the \$ and \_
      - also alphabetic characters from other languages
  - Which of the following are not valid identifiers?

```
userName user_name $cash 2ndName
first name user.age _note_ note2
```

34

## Identifiers

- Identifier must
  - Start with a letter and be followed by
  - Zero or more letters and/or digits
    - Digits are 0 through 9.
    - Letters are the 26 characters in English alphabet
      - both uppercase and lowercase
      - plus the \$ and \_
      - also alphabetic characters from other languages
  - Which of the following are not valid identifiers?

```
userName user_name $cash 2ndName
first name user.age _note_ note2
```

35

## Identifiers

- Java is case sensitive
- Oreo oreo OREO Oreo
  - are all different identifiers, so be careful
  - common source of errors in programming

36

## Identifiers

- Java is case sensitive
- Oreo oreo OREO Oreo
  - are all different identifiers, so be careful
  - common source of errors in programming
- are these all valid identifiers?

37

## Identifiers

- Creating identifiers in your Java programs
  - Remember other people read what you create
  - Make identifiers meaningful and descriptive for both you and them
- No limit to how many characters you can put in your identifiers
  - but don't get carried away

```
public class ReallyLongNamesWillDriveYouCrazyIfYouGoOverboard
{
 public static void main (String[] args)
 {
 System.out.println ("Enough already!");
 }
}
```

38

## White Space

```

// Oreo.java Author: Kurt Eiselt
//
// Demonstrating good use of white space
//*****
public class Oreo
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

39

## White Space

```

// Oreol.java Author: Kurt Eiselt
//
// Demonstrating mediocre use of white space
//*****
public class Oreol
{
 public static void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

40

## White Space

```

// Oreo2.java Author: Kurt Eiselt
//
// Demonstrating bad use of white space
//*****
public class Oreo2 { public static void main (String[]
args) { System.out.println ("Feed me more Oreos!"); } }
```

41

## White Space

```

// Oreo3.java Author: Kurt Eiselt
//
// Demonstrating totally bizarre use of white space
//*****
public
class Oreo3
{
 public static
void main (String[] args)
 {
 System.out.println ("Feed me more Oreos!");
 }
}
```

42

```

// Oreo4.java Author: Kurt Eiselt
//
// Demonstrating deep psychological issues with whitespace
//*****
public
class
Oreo4
{
 public
static
void
main
{
 String[]
args
}
{
 System.out.println
("Feed me more Oreos!");
;
}
}
```

43

## White Space

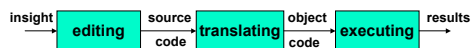
## White Space

- White space
  - Blanks between identifiers and other symbols
  - Tabs and newline characters are included
- White space does not affect how program runs
- Use white space to format programs we create so they're easier for people to understand

44

## Program Development

- Use an editor to create your Java program
  - often called **source code**
  - **code** used interchangeably with **program** or **instructions** in the computer world
- Another program, a **compiler** or an **interpreter**, translates source code into target language or **object code**, which is often machine language
- Finally, your computer can execute object code



45

## Compiling and Running

- Let's try it!
  - command line for now
  - later we'll use Eclipse
    - integrated development environment (IDE)

46

## Syntax

- Rules to dictate how statements are constructed.
  - Example: open bracket needs matching close bracket
- If program is not syntactically correct, cannot be translated by compiler
- Different than humans dealing with natural languages like English. Consider statement with incorrect syntax (grammar)
  - for weeks. rained in Vancouver it hasn't
- we still have pretty good shot at figuring out meaning

47

## Semantics

- What will happen when statement is executed
- Programming languages have well-defined semantics, no ambiguity
- Different than natural languages like English. Consider statement:
  - Mary counted on her computer.
- How could we interpret this?
- Programming languages cannot allow for such ambiguities or computer would not know which interpretation to execute

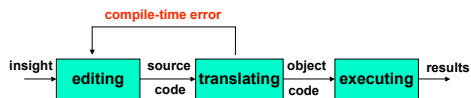
48

## Errors

- Computers follow our instructions exactly
- If program produces the wrong result it's the programmer's fault
  - unless the user inputs incorrect data
  - then cannot expect program to output correct results: "Garbage in, garbage out" (GIGO)
- Debugging**: process of finding and correcting errors
  - Unfortunately can be very time consuming!

49

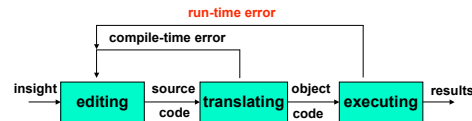
## Errors



- Error at compile time (during translation)
  - you did not follow syntax rules that say how Java elements must be combined to form valid Java statements

50

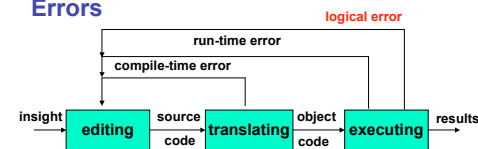
## Errors



- Error at run time (during execution)
  - Source code compiles
    - Syntactically (structurally) correct
  - But program tried something computers cannot do
    - like divide a number by zero.
  - Typically program will **crash**: halt prematurely

51

## Errors



- Logical error
  - Source code compiles
  - Object code runs
  - But program may still produce incorrect results because logic of your program is incorrect
    - Typically hardest problems to find

52

## Errors

- Let's try it!
  - usually errors happen by mistake, not on purpose...

53

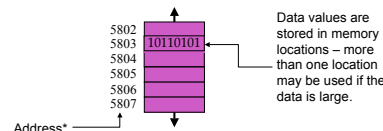
## Memory and Identifiers

- Example of a high-level instruction
  - A = B + C
- Tells computer to
  - go to main memory and find value stored in location called B
  - go to main memory and find value stored in location called C
  - add those two values together
  - store result in memory in location called A
- Great! But... in reality, locations in memory are not actually called things like a, b, and c.

54

## Memory Recap

- Memory: series of locations, each having a unique address, used to store programs and data
- When data is stored in a memory location, previously stored data is overwritten and destroyed
- Each memory location stores one byte (8 bits) of data



\*For total accuracy, these addresses should be binary numbers, but you get the idea, no? 55

56

## Memory and Identifiers

- So what's with the a, b, and c?
  - Machine language uses actual addresses for memory locations
  - High-level languages easier
    - Avoid having to remember actual addresses
    - Invent meaningful identifiers giving names to memory locations where important information is stored
- pay\_rate** and **hours\_worked** vs. **5802** and **5806**
  - Easier to remember and a whole lot less confusing!

## Memory and Identifiers: Variables

- Variable**: name for location in memory where data is stored
  - like variables in algebra class
- pay\_rate**, **hours\_worked**, **a**, **b**, and **c** are all variables
- Variable names begin with lower case letters
  - Java convention, not compiler/syntax requirement
- Variable may be name of single byte in memory or may refer to a group of contiguous bytes
  - More about that next time

57

## Programming With Variables

```
.....
// Test.java Author: Kurt
//
// Our first use of variables!
//.....

public class Test
{
 public static void main (String[] args)
 {
 a = b + c;
 System.out.println ("The answer is " + a);
 }
}
```

- Let's give it a try...

58

## Programming With Variables

```
.....
// Test.java Author: Kurt
//
// Our first use of variables!
//.....

public class Test
{
 public static void main (String[] args)
 {
 a = b + c;
 System.out.println ("The answer is " + a);
 }
}
```

- Let's give it a try...
  - b and c cannot be found!
  - need to assign values

59

## Programming With Variables: Take 2

```
.....
// Test2.java Author: Kurt
//
// Our second use of variables!
//.....

public class Test2
{
 public static void main (String[] args)
 {
 b = 3;
 c = 5;
 a = b + c;
 System.out.println ("The answer is " + a);
 }
}
```

60

## Programming With Variables: Take 2

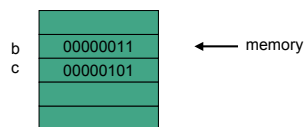
```
.....
// Test2.java Author: Kurt
//
// Our second use of variables!
//.....

public class Test2
{
 public static void main (String[] args)
 {
 b = 3;
 c = 5;
 a = b + c;
 System.out.println ("The answer is " + a);
 }
}
```

- Now what?
  - such a lazy computer, still can't find symbols...

61

## Now What?



- Java doesn't know how to interpret the contents of the memory location
  - are they integers? characters from the keyboard? shades of gray? or....

62

## Data Types

- Java requires that we tell it what kind of data it is working with
- For every variable, we have to declare a **data type**
- Java language provides eight **primitive** data types
  - i.e. simple, fundamental
- For more complicated things, can use data types
  - created by others provided to us through the Java libraries
  - that we invent
    - More soon - for now, let's stay with the primitives
- We want **a**, **b**, and **c** to be integers. Here's how we do it...

63

## Programming With Variables: Take 3

```
.....
// Test3.java Author: Kurt
//
// Our third use of variables!
//.....

public class Test3
{
 public static void main (String[] args)
 {
 int a; //these
 int b; //are
 int c; //variable declarations
 b = 3;
 c = 5;
 a = b + c;
 System.out.println ("The answer is " + a);
 }
}
```

64

## Primitive Data Types: Numbers

| Type   | Size    | Min                              | Max                             |
|--------|---------|----------------------------------|---------------------------------|
| byte   | 1 byte  | -128                             | 127                             |
| short  | 2 bytes | -32,768                          | 32,767                          |
| int    | 4 bytes | -2,147,483,648                   | 2,147,483,647                   |
| long   | 8 bytes | -9,223,372,036,854,775,808       | 9,223,372,036,854,775,807       |
| float  | 4 bytes | approx -3.4E38 (7 sig.digits)    | approx 3.4E38 (7 sig.digits)    |
| double | 8 bytes | approx -1.7E308 (15 sig. digits) | approx 1.7E308 (15 sig. digits) |

- Six primitives for numbers
  - integer vs. floating point
  - fixed size, so finite capacity

65

## Primitive Data Types: Non-numeric

- Character Type
  - named **char**
  - Java uses the Unicode character set so each char occupies 2 bytes of memory.
- Boolean Type
  - named **boolean**
  - Variables of type boolean have only two valid values
    - true and false
  - Often represents whether particular condition is true
  - More generally represents any data that has two states
    - yes/no, on/off

66

## Primitive Data Types: Numbers

| Type   | Size    | Min                              | Max                             |
|--------|---------|----------------------------------|---------------------------------|
| byte   | 1 byte  | -128                             | 127                             |
| short  | 2 bytes | -32,768                          | 32,767                          |
| int    | 4 bytes | -2,147,483,648                   | 2,147,483,647                   |
| long   | 8 bytes | -9,223,372,036,854,775,808       | 9,223,372,036,854,775,807       |
| float  | 4 bytes | approx -3.4E38 (7 sig.digits)    | approx 3.4E38 (7 sig.digits)    |
| double | 8 bytes | approx -1.7E308 (15 sig. digits) | approx 1.7E308 (15 sig. digits) |

- Primary primitives are **int** and **double**
  - Just worry about those for now

67

## Questions?

68