



## Reading This Week

- Chap 1: 1.3-1.8
- Chap 2: 2.1-2.2, 2.5
- Chap 4: 4.1-4.2
- reminder: weekly reading questions due next time (Fri) at start of lecture

2

## Review: High-Level Language

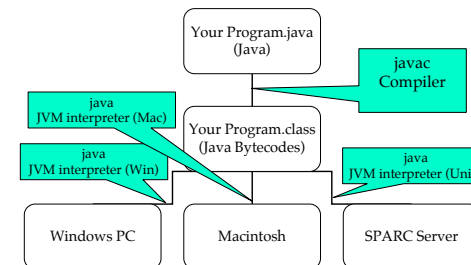
- Must be translated into machine language so the computer can understand it.
- High-level instruction:  $A = B + C$  becomes at least four machine language instructions!

```
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**).

3

## Review: Java Does Both!



4

## Review: Comments

- Comments: help humans understand
  - ignored by compiler
  - comment out rest of line: `//`
  - comment start/end: `/* */`

5

## Review: Identifiers

```
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
  - Kurt made up identifier **Oreo**
  - Other programmers chose identifier **System.out.println**

6

## Review: 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	

7

## Review: 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
```

8

## 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?

9

## 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!");
    }
}
```

10

## 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!");
    }
}
```

11

## 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!");
    }
}
```

12

## 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!"); } }
```

13

## 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!")
    ;
    }
}
```

14

## White Space

```
.....
// 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!")
    ;
    }
}
```

15

## 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

16

## 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



17

## Compiling and Running

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

18

## Compiling and Running Java

- what I did at the command line
  - create file HelloWorld.java in text editor
    - containing class HelloWorld
  - compile it: "javac HelloWorld.java"
    - compiler makes file HelloWorld.class
  - run it in the interpreter: "java HelloWorld"
- don't panic if this is mysterious!
  - hands-on practice in labs this week
  - see detailed instructions on WebCT for how to download and configure your home desktop/laptop
    - if you get stuck, bring laptop to lab or DLC for help

- a few weeks from now: Eclipse IDE

19

## 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

20

## 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

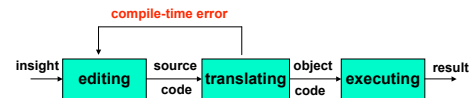
21

## Errors

- Computers follows 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!

22

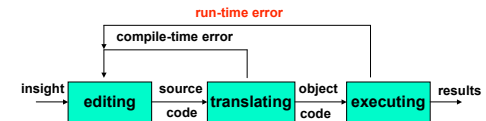
## Errors



- 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

23

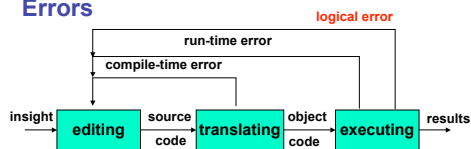
## Errors



- 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

24

## 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

25

## Errors

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

26

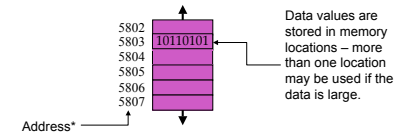
## Memory and Identifiers

- 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.

27

## 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? 28

## 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!

29

## 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

30

## 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...

31

## 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

32

## 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);
    }
}

```

33

## Programming With Variables: Take 2

```

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

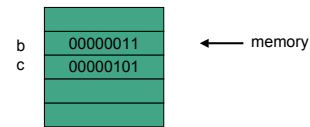
public class Test2
{
    public static void main (String[] args)
    {
        b = 3;
        c = 5;
        a = b + c;
        System.out.println ("The answer is " + a);
    }
}

```

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

34

## 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....

35

## 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 more 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...

36

## 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);
    }
}

```

37

## Data Types: Int and Double

- int
  - integer
- double
  - real number
  - (double-precision floating point)

38

## Floating Point Numbers

- significant digits
  - 42
  - 4.2
  - 42000000
  - .000042

39

## Floating Point Numbers

- significant digits
  - 42 =  $4.2 * 10 = 4.2 * 10^1$
  - 4.2 =  $4.2 * 1 = 4.2 * 10^0$
  - 42000000 =  $4.2 * 10000000 = 4.2 * 10^7$
  - .000042 =  $4.2 * .00001 = 4.2 * 10^{-5}$

40

## Floating Point Numbers

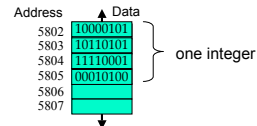
- significant digits
  - 42 =  $4.2 * 10 = 4.2 * 10^1$
  - 4.2 =  $4.2 * 1 = 4.2 * 10^0$
  - 42000000 =  $4.2 * 10000000 = 4.2 * 10^7$
  - .000042 =  $4.2 * .00001 = 4.2 * 10^{-5}$
- only need to remember
  - nonzero digits
  - where to put the decimal point
    - floats around when multiply/divide by 10

41

## Data Type Sizes

Type	Size	Min	Max
int	4 bytes	-2,147,483,648	2,147,483,647
double	8 bytes	approx -1.7E308 (15 sig. digits)	approx 1.7E308 (15 sig. digits)

- fixed size, so finite capacity



42

## Variable Declaration Examples

- person's age in years
- height of mountain to nearest meter
- length of bacterium in centimeters
- number of pets at home

43

## Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - `typeName variableName;`

44

## Assignment

```

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

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

```

45

## Assignment Statements

- Assignment statement** assigns value to variable
    - sometimes say **binds** value to variable
  - Assignment statement is
    - identifier
    - followed by assignment operator (=)
    - followed by expression
    - followed by semicolon (;)
- ```

b = 3;
c = 8;
a = b + c;
weekly_pay = pay_rate * hours_worked;

```
- Note that = is no longer a test for equality!

46

## Assignment Statements

- Java first computes value on right side
  - Then assigns value to variable given on left side
- ```

x = 4 + 7; // what's in x?

```
- Old value will be overwritten if variable was assigned before
- ```

x = 2 + 1; // what's in x now?

```

47

## Assignment Statements

- Here's an occasional point of confusion:
 

```

a = 7; // what's in a?
b = a; // what's in b?
// what's in a now???
```

48

## Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;          // what's in a?
b = a;          // what's in b?
                // what's in a now??
System.out.println("a is " + a + " b is " + b);
```

- Find out! Experiments are easy to do in CS

49

## Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;          // what's in a?
b = a;          // what's in b?
                // what's in a now??
System.out.println("a is " + a + " b is " + b);
```

- Variable values on left of = are clobbered
- Variable values on right of = are unchanged
  - copy of value assigned to a also assigned to b
  - but that doesn't change value assigned to a

50

## Assignment Statements

- Here's an occasional point of confusion:

```
a = 7;          // what's in a?
b = a;          // what's in b?
                // what's in a now??
System.out.println("a is " + a + " b is " + b);
a = 8;
System.out.println("a is " + a + " b is " + b);
```

- Memory locations a and b are distinct
  - copy of value assigned to a also assigned to b
  - changing a later does not affect previous copy
    - more later

51

## Variable Declaration and Assignment

- variable declaration is instruction to compiler
  - reserve block of main memory large enough to store data type specified in declaration
- variable name is specified by identifier
- syntax:
  - *typeName* *variableName*;
  - *typeName* *variableName* = *value*;
    - can declare and assign in one step

52