Lecture 1 - Binary Recap And Java

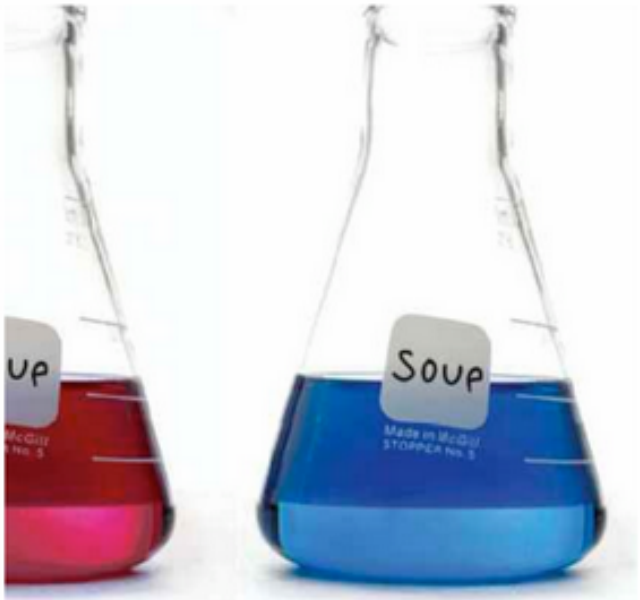
Bentley James Oakes

January 11, 2018

Soup and Science

soup & science

McGill
Faculty of Science
Office for Undergraduate Research in Science



Learn about cutting-edge research over lunch with cool profs

January 15-19, 2018
11:30 AM
Redpath Museum

More information:
www.mcgill.ca/science

This Lecture

Section 1

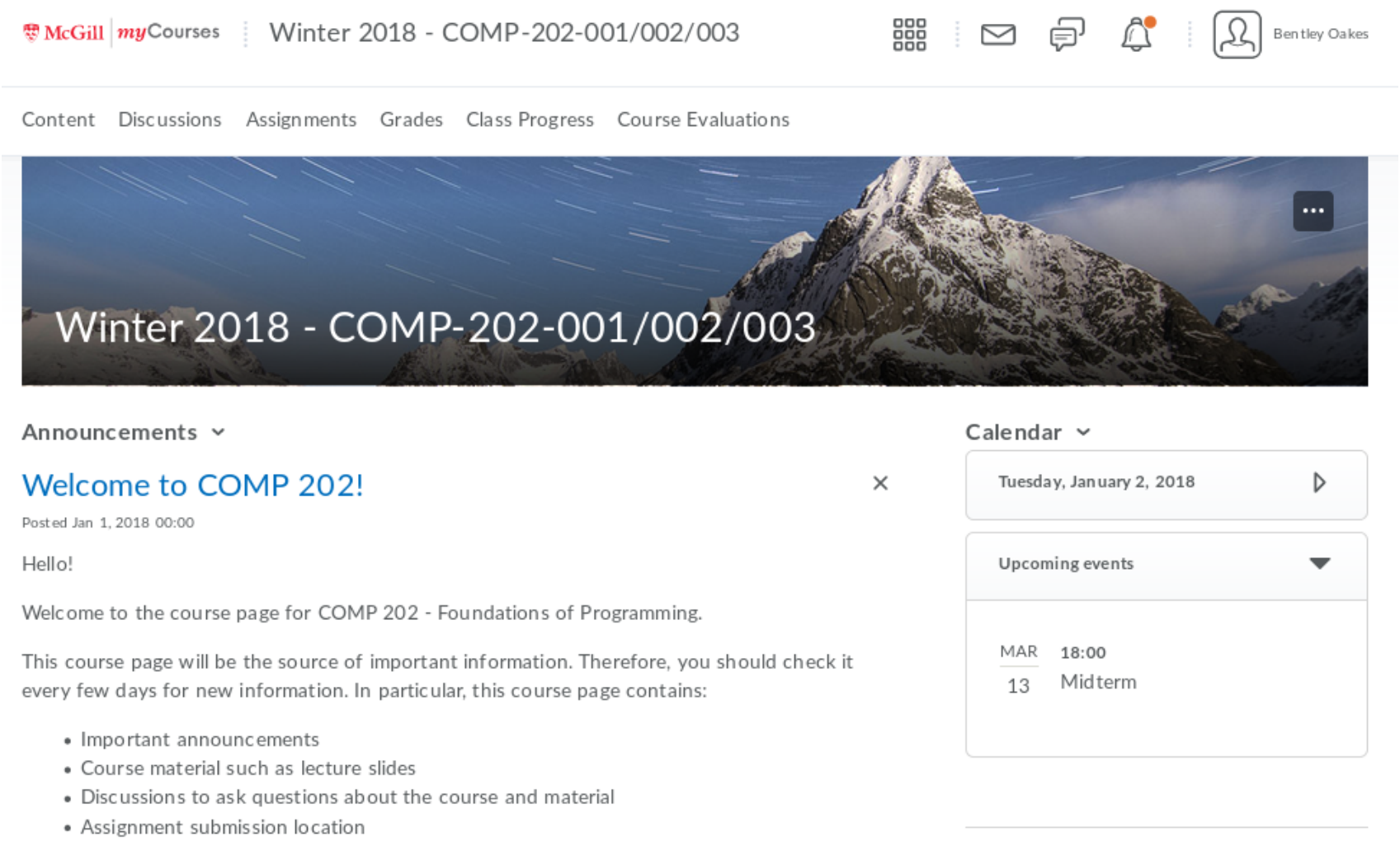
Syllabus Refresh

Syllabus

- Available on myCourses
- Freq. Questions
  - Assignments/Midterm/Final same for all sessions
  - Can join lectures if you need to, don't need to switch on Minerva
  - Session full? Keep trying
  - Slides are posted before class
  - Recordings **may** be posted
    - Don't substitute these for going to class!
  - Course assumes no programming knowledge
- Please read syllabus as soon as possible

MyCourses

- URL: `http://www.mcgill.ca/lms/`



Course Updates

- Posted slides from last class on myCourses
  - Please read if needed
- Assignment out now
  - Due January 31st
  - Has Java programming questions
  - Warm-up questions

Section 2

Explaining Programming

Algorithms

- Programming is about writing *algorithms*
- An algorithm is a 'step-by-step set of operations to be performed'
- Composed of **Input** $\rightarrow$ **Instructions** $\rightarrow$ **Output**
- Examples:
  - Baking
    - Ingredients $\rightarrow$ Recipe instructions $\rightarrow$ Pinata cake
  - Counting people in a room
    - Please watch this video later:
      `https://www.youtube.com/watch?v=6hf0vs8pY1k`
    - Input: Room $\rightarrow$ Instructions: Counting $\rightarrow$ Output: Num. of people

Section 3

Bases and Binary Numbers

Binary Intro

- We want to represent numbers in our computers
- But there's a problem...
- Computers are made of wires
- These wires can be powered on or off

- How can we represent numbers using just on and off?
- We use the 'base 2' system - called 'binary'

Bases

Normally, we deal in base ten numbers - call it 'decimal'

For example, consider 4156
This number contains thousands, hundreds, tens, and ones
Each position in the number represents a different value
Numbers to the left represent the larger values

We have broken up the number into 'powers of ten'
$1000 = 10 * 10 * 10$, $100 = 10 * 10$, $10 = 10$, $1 = 1$

Base 2 Introduction

- In binary, we use powers of two instead of tens to represent numbers
- For example, consider the binary number 101
- The right digit represents the 'ones'
- The middle digit represents the 'twos'
- The left digit represents the 'fours'

- 101bin
- $= 1 * 4 + 0 * 2 + 1 * 1$
- $= 1 * 2^2 + 0 * 2^1 + 1 * 2^0$
- $= 5$ dec

Exponents and powers

$2^3 = 2 * 2 * 2 =$ 'two raised to the power of three'
$Anything^0 =$ 'anything raised to the power of 0' $= 1$

4156 in decimal
$= 4 * 1000 + 1 * 100 + 5 * 10 + 6 * 1$
$= 4 * 10^3 + 1 * 10^2 + 5 * 10^1 + 6 * 10^0$
Each place from the ones to the thousands is a power of ten more

1011 in binary
$= 1 * 8 + 0 * 4 + 1 * 2 + 1 * 1$
$= 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0$
Each place from the ones to the eights is a power of two more

Up to 8 in Binary

Decimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000

Powers of Two

Memorizing these aren't necessary, but they can speed up these conversions on the midterm and final

Power of Two	Decimal
2^0	1
2^1	2
2^2	4
2^3	8
2^4	16
2^5	32
2^6	64
...	
2^{10}	1024

Section 4

Conversion Algorithms

Conversions

- As easy algorithms, we'll start with converting between binary and decimal
  - Shows step-by-step nature of algorithms
  - Shows how binary is used within computers
  - Shows how we remove tedious things by using programming
- Guaranteed to be on the midterm/final
  - Do practice problems on your own

Binary-to-Decimal

- Input: A binary number
- Instructions:
  - Write the powers of 2 below each digit
  - Starting with 1 to the far right
  - Add up the powers of 2 if a 1 appears in the binary number
  - This sum is the answer
- Output: The number in decimal

Binary-to-Decimal Example 1

- Input: 1110
- Write the powers of 2 underneath

1	1	1	0
8	4	2	1

- Add up the powers where a 1 appears in the binary

Answer $= 8 + 4 + 2 = 14$

- Output: 14

Binary-to-Decimal Example 2

- Input: 10101101
- Write the powers of 2 underneath

1	0	1	0	1	1	0	1
128	64	32	16	8	4	2	1

- Add up the powers where a 1 appears in the binary

Answer $= 128 + 32 + 8 + 4 + 1 = 173$

- Output: 173

Binary-to-Decimal Example 3

- Input: 11001011
- Write the powers of 2 underneath

1	1	0	0	1	0	1	1
128	64	32	16	8	4	2	1

- Add up the powers where a 1 appears in the binary

Answer $= 128 + 64 + 8 + 2 + 1 = 203$

- Output: 203

Decimal-to-Binary

- Input: A decimal number
- Instructions:
  - Go through the powers of two from higher to lower
  - If you can subtract the power, add a 1 to the binary number
  - If not, add a zero
- Output: The number in binary

Decimal-to-Binary Example

- Input: 5
- Go through the powers of two and subtract if you can
- If you subtract, add 1 to the binary number, otherwise add 0

	4	**2**	**1**	**Powers of two**
5				Start
1	1			4 fits into 5, remainder 1
1		0		2 does not fit
0			1	1 fits into 1, remainder 0

- Write down the 1s and 0s

- Output: 101

Decimal-to-Binary Example

- Input: 27
- Go through the powers of two and subtract if you can
- If you subtract, add 1 to the binary number, otherwise add 0

	16	**8**	**4**	**2**	**1**	**Powers of two**
27						Start
11	1					16 fits into 27, remainder 11
3		1				8 fits into 11, remainder 3
3			0			4 does not fit into 3
1				1		2 fits into 3, remainder 1
0					1	1 fits into 1, remainder 0

- Write down the 1s and 0s
- Output: 11011

Decimal-to-Binary Example

- Input: 61
- Go through the powers of two and subtract if you can
- If you subtract, add 1 to the binary number, otherwise add 0

	32	**16**	**8**	**4**	**2**	**1**	**Powers of two**
61							Start
29	1						32 fits into 61, remainder 29
13		1					16 fits into 29, remainder 13
5			1				8 fits into 13, remainder 5
1				1			4 fits into 5, remainder 1
1					0		2 does not fit into 3
0						1	1 fits into 1, remainder 0

- Write down the 1s and 0s
- Output: 111101

Practice

Anurag Roy found two websites to help you practice these conversions:

- `http://acc6.its.brooklyn.cuny.edu/~gurwitz/core5/binquiz.html`
- `http://www.free-test-online.com/binary/binary2decimal.htm`

Section 5

More Binary

Binary Addition

- Addition is a very common operation
- And it's very fast for computers (and you) to perform in binary
- To add binary, just perform normal addition, but instead of carrying the one at 10, carry it at 2

				Binary	**Decimal**
	0	0	1		1
+	0	1	0		2
	0	1	1		3

Binary Addition

- An example with carrying the one:

				Binary	**Decimal**
	0	1	1		3
+	0	1	0		2
	1	0	1		5

Binary Addition

- Another example:

					Binary	**Decimal**
	1	0	1	0		10
+	0	0	1	1		3
	1	1	0	1		13

Practice on your own with small numbers.
We won't ask for anything over four bits plus four bits.
Double-check your results by converting to decimal.

Overflow

- What happens if we run out of bits to carry the one?
- Let's assume we are only using four bits

					Binary	**Decimal**
	1	1	1	1		15
+	0	0	0	1		1
	0	0	0	0		$\neq 16, = 0$

Java will not tell you about this **overflow**.

This can (and will) give you errors in your programs

Characters

- Letters in Java are called *characters*
- These characters must also be represented in binary
- Java maps each character to a number, and automatically converts when needed
- Therefore **don't memorize the following table**
- Provided to show the mapping only, won't be on tests

ASCII Table

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040		Space	64	40	100	@	@	96	60	140	`	`
33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
52	34	064	4	4	84	54	124	T	T	116	74	164	t	t

Hexadecimal

```
0001DC30 41 57 41 56 41 89 FF 41 55 41 54 4C 8D 25 CE C8 00 00 55  AWAVA..AUATL.%....U
0001DC43 48 8D 2D D6 C8 00 00 53 49 89 F6 49 89 D5 4C 29 E5 48 83  H.-....SI..I..L).H.
0001DC56 EC 08 48 C1 FD 03 E8 EF 8D FE FF 48 85 ED 74 20 31 DB 0F  ..H........H..t 1..
0001DC69 1F 84 00 00 00 00 00 4C 89 EA 4C 89 F6 44 89 FF 41 FF 14  .......L..L..D..A..
0001DC7C DC 48 83 C3 01 48 39 DD 75 EA 48 83 C4 08 5B 5D 41 5C 41  .H...H9.u.H...[]A\A
0001DC8F 5D 41 5E 41 5F C3 90 66 2E 0F 1F 84 00 00 00 00 00 F3 C3  ]A^A_..f...........
0001DCA2 66 66 66 66 66 2E 0F 1F 84 00 00 00 00 00 48 83 EC 08 48  fffff.........H...H
0001DCB5 83 C4 08 C3 00 00 00 00 00 00 00 00 79 65 73 00 6E 6F 00  ............yes.no.
0001DCC8 2F 00 6E 2F 61 00 2E 2E 2F 73 72 63 2F 73 79 73 74 65 6D  /.n/a.../src/system
0001DCDB 63 74 6C 2F 73 79 73 74 65 6D 63 74 6C 2E 63 00 70 61 74  ctl/systemctl.c.pat

Signed 8 bit: -67          Signed 32 bit: 1207959997      Hexadecimal: BD
Unsigned 8 bit: 189        Unsigned 32 bit: 1207959997        Octal: 275
Signed 16 bit: 445         Signed 64 bit: 1207959997       Binary: 10111101
```

- Not testable - Just for understanding
- Here's a computer program in hexadecimal
- Instead of looking at binary, much easier to look at hexadecimal
- Hexadecimal is base 16: 0-9, A-F to represent 0-15
- The computer knows how to map the binary values to instructions and data

Section 6

Java

Java Basics

- First appeared in May 1995 (over 20 years ago)
- Based on older languages (going back to the 50's)
- Still heavily used, especially in business and teaching
- Students often confuse Java with Javascript
  - Very different languages

High-Level Language

- The computer knows how to execute instructions in binary code
- But this is really painful to program in
- We use a number of translation steps from our **high-level language** (Java) to binary code

Two Steps to a Java Program

1. Compile the program: Java source code $\xrightarrow{Compiler}$ Java bytecode
2. Run the program: Java bytecode $\xrightarrow{JavaVirtualMachine}$ Binary code

Java Source Code

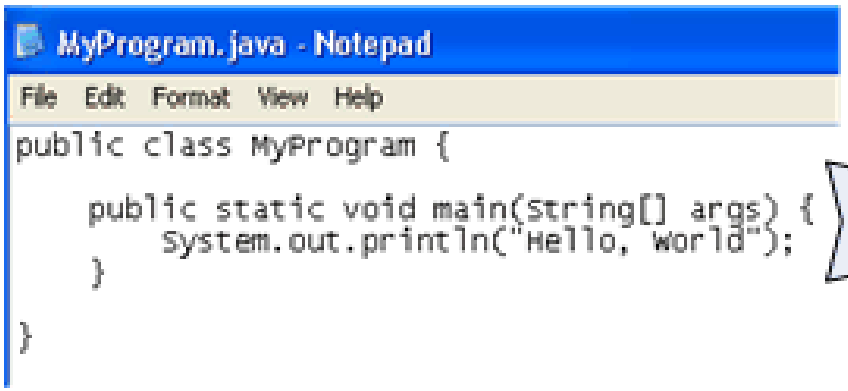
- Human-readable code in the Java language
- Stored in .java files
- This is what you will write

```java
public class HelloWorld
{
  public static void main(String[] args)
  {
    //A comment: This program
    //prints out "Hello World!"
    System.out.println("Hello World!");
  }
}
```

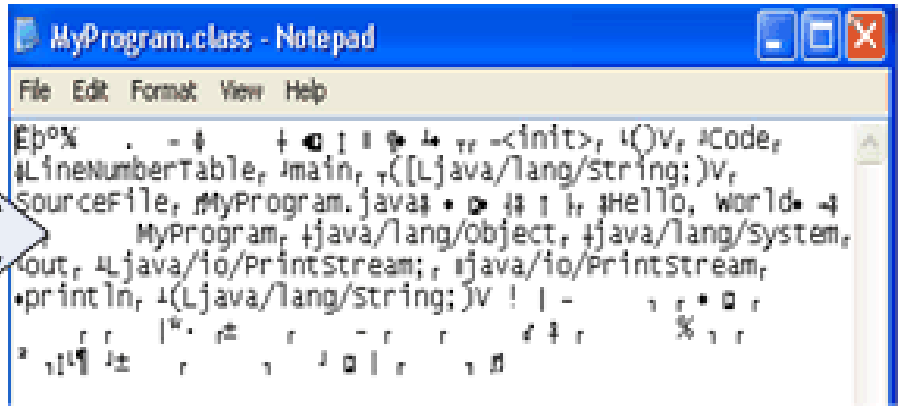
Compiler

- Transforms source code into Java bytecode
- Verifies source code is correct (as much as it can)
- Very important to understand compiler errors
  - Remember: The compiler just notices errors, it doesn't know what you want to do
  - The faster you learn what each error means, the faster you'll be able to program

Java Bytecode



Source code is first written in plain text files ending with the .java extension.



After the compilation is successful, java compiler will generate an intermediate ".class" file that contains the bytecode.

- A special language used behind-the-scenes
- Not at all human-readable
- Stored in .class files
  - Do not hand these files in for your assignment

Bytecode Example

Further reading (not needed for the class) and source of last picture:
`https://j4school.wordpress.com/java-tutorials/core-java/introduction-to-java/java-magic-bytecode-java-virtual-machine-jit-jre-jdk/`

Java Virtual Machine

- Transforms bytecode into binary code
- Then executes the binary code to actually run the program
- The compiler and Java virtual machine are in the Java Development Kit (JDK) that you need to install

Two Steps to a Java Program

1. Compile the program: Java source code $\xrightarrow{Compiler}$ Java bytecode
2. Run the program: Java bytecode $\xrightarrow{JavaVirtualMachine}$ Binary code

Why?

- Why do programmers bother with these different levels? We want to:
- Write code at a high-level
  - As close to English as possible
- Not worry about learning the binary instructions that a computer knows
- Let the compiler check our code for errors and perform optimizations

Section 7

Example Java Programs

Magic Words

- For now, lots of the words on the next few slides will be ‘magic’
- That is, you won’t understand what they mean until well into the course
- You’ll have to memorize them for now

Hello World Example

```
1 public class HelloWorld
2 {
3   public static void main(String[] args)
4   {
5     //A comment: This program
6     //prints out "Hello World!"
7     System.out.println("Hello World!");
8   }
9 }
```

Short Tutorial:

http://www.seas.upenn.edu/~pfpcse/java/DrJavaTutorial.html

Classes

```
1 public class HelloWorld
2 {
3   public static void main(String[] args)
4   {
5     //A comment: This program
6     //prints out "Hello World!"
7     System.out.println("Hello World!");
8   }
9 }
```

- Line 1 - Defining a class 'HelloWorld'
  - We'll learn what classes mean later
  - Note: The file name must match the class name
    - In this case, the filename must be 'HelloWorld.java'
- Line 2 - The opening brace to the class
- Line 9 - The closing brace to the class
  - All lines between the braces are part of the class

Main Method

```
1 public class HelloWorld
2 {
3   public static void main(String[] args)
4   {
5     //A comment: This program
6     //prints out "Hello World!"
7     System.out.println("Hello World!");
8   }
9 }
```

- Line 3 - The main method of the class
  - This is where Java will start executing instructions
  - For now, these are 'magic words' you'll have to remember
- Line 4 - The opening brace to the main method
- Line 6 - The closing brace to the main method
- All lines 3-6 are part of the main method

Instructions

```
1 public class HelloWorld
2 {
3   public static void main(String[] args)
4   {
5     //A comment: This program
6     //prints out "Hello World!"
7     System.out.println("Hello World!");
8   }
9 }
```

- Line 7 - The instruction to print out "Hello World"
- Note it ends with a semi-colon
- This is a **statement**

Comments

- Writing // makes the rest of the line a comment
- Comments are for humans, not the computer
  - Lots of comments will be required for all assignments
- Useful to temporarily disable lines of code
- There are also multi-line comments: /* */

Quote on commenting:

You know you're brilliant, but maybe you'd like to understand what you did two weeks from now. - Linux style guide

Section 8

Variables

Variables

- Variables are a key concept in programming
- Think of variables like a box to put something in
- A variable has three parts: A **name**, **type**, and **value**
- Example: A variable to store the number of students
  - Name: 'numStudents'
  - Type: Integer (int), to hold whole numbers
  - Value: 25

Declaration

- How can we create the 'numStudents' variable in Java?
- Write the statement `int numStudents;`
  - Note the semi-colon at the end
- This is called a **declaration**
- We are **declaring** to Java that we want a variable named 'numStudents'
- In our box analogy:
  - We tell Java we want a box to hold whole numbers, with a big 'numStudents' written on the side

Initialization

- Note that we haven't given 'numStudents' a value yet
- Write the next statement `numStudents = 25;`
- This is called an **initialization**
- We are **initializing** the variable named 'numStudents' with the value 25
- In our box analogy:
  - We are placing the value 25 within the box

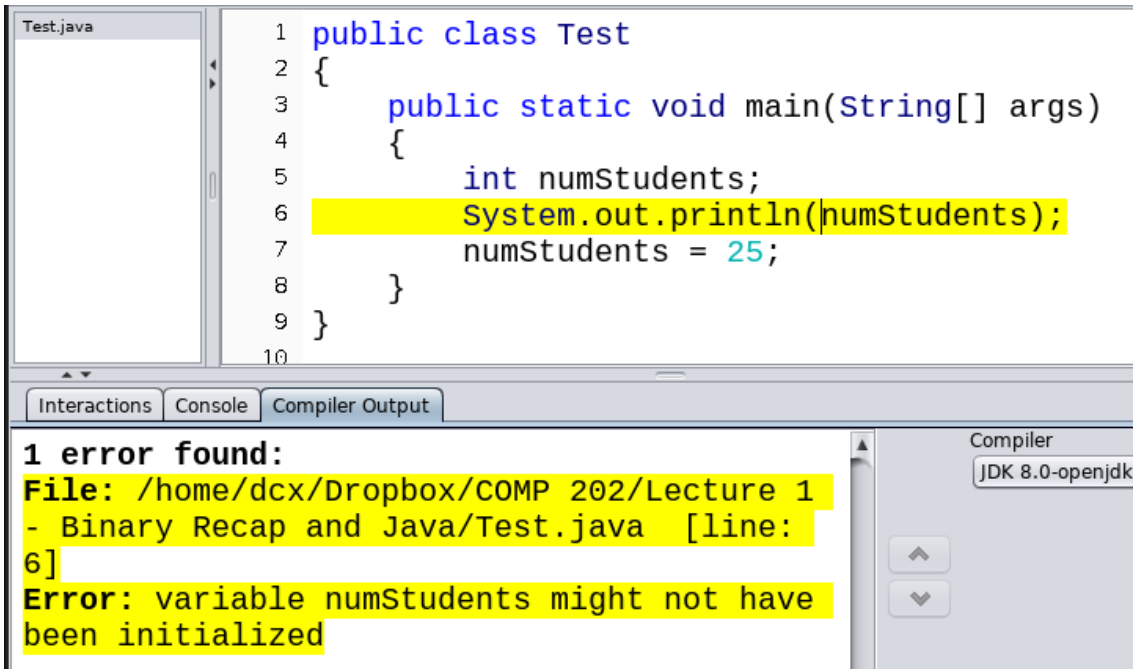
Printing the Value of a Variable

The statement `System.out.println(numStudents)` will print out the value of this variable.

```
1 public class Test
2 {
3     public static void main(String[] args)
4     {
5         int numStudents;
6         numStudents = 25;
7         System.out.println(numStudents);
8     }
9 }
```

Declaration and Initialization

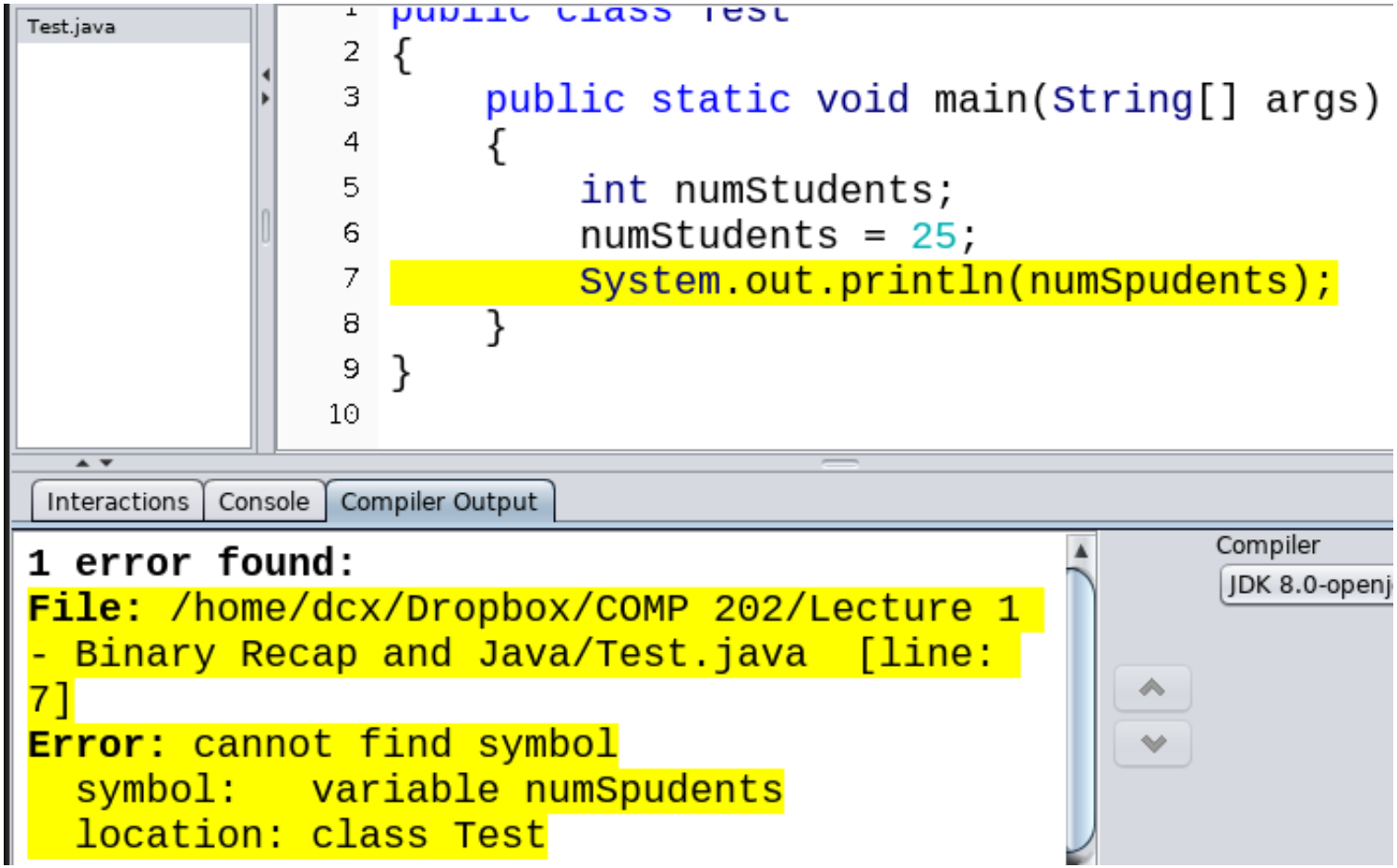
- What is the value of the variable between declaration and initialization?
  - Put the `System.out.println(numStudents)` between the declaration and initialization statements
  - This gives a compiler error ("variable might not be initialized")
  - A variable without a value is probably an error



- Best way to solve this is to declare and initialize at same time
- `int numStudents = 25;`

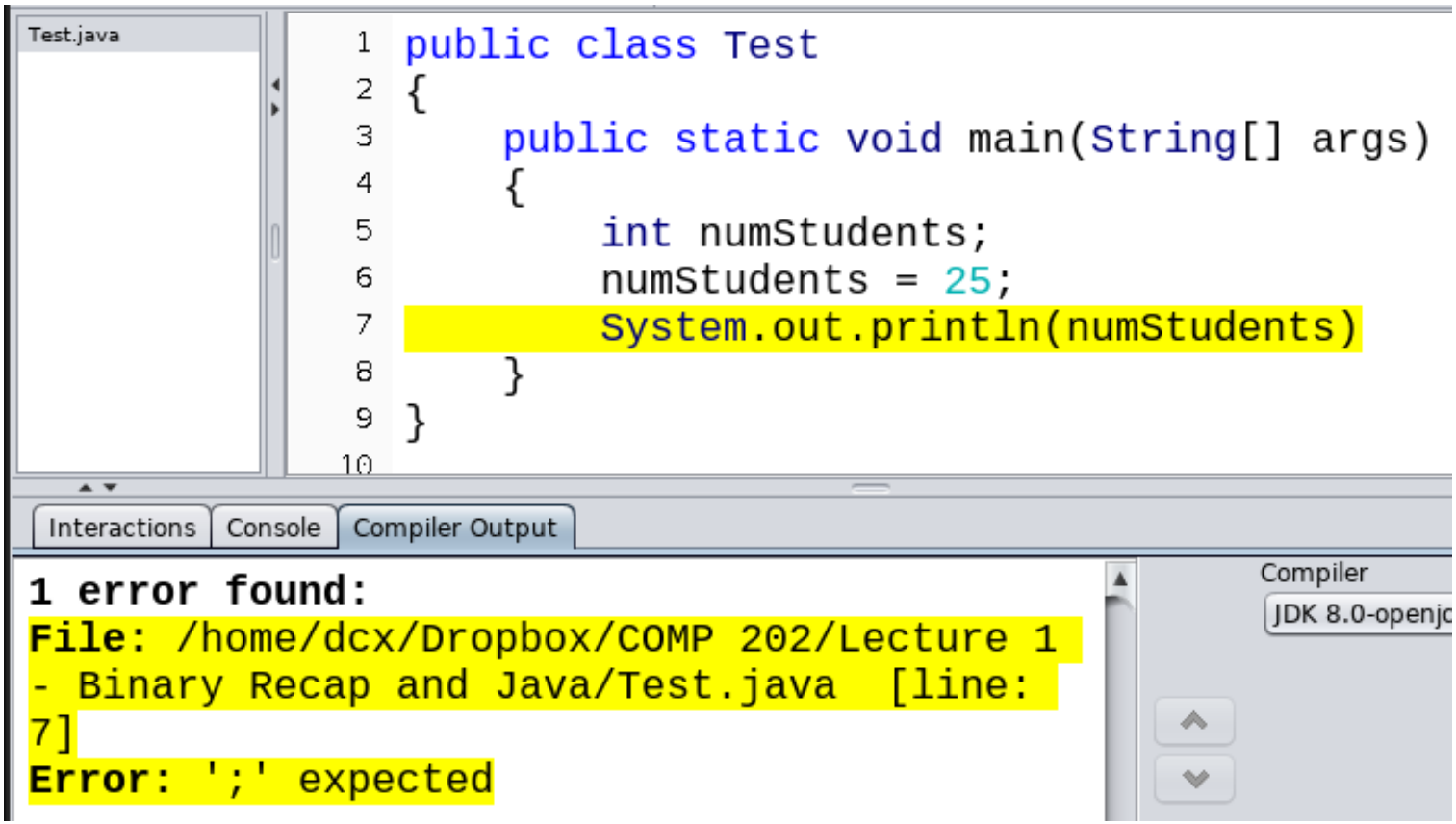
Missing Variable

- Another common error is to misspell the variable



Missing Semi-Colon

- Another common error is to forget the semi-colon ;



Section 9

Variable Types

Variable Types

- int - Integers (Whole numbers)
- float or double - Non-whole numbers (like 1.4)
- String - Collection of characters
- boolean - True/False values

Integers

- int - Integers (Whole numbers)
- Examples:
  - `int numStudents = 2;`
  - `int carsOnTheDriveway = 3;`

Float/Doubles

- float - Floating point number
- double - Double precision number
- Can store non-whole values, like 3.5
- Examples:
  - `double pi = 3.14159;`
  - `double fractionOfPeopleWhoEatPie = 93.456;`

Strings

- Collections of characters
- Examples
  - `String s = “Hello”;`
- Two important things to note:
  - Capital s in *String*
  - Double quotation marks around *String* value

Adding to a String

- We might want to add two *Strings* together
- Called **concatenation**
- Example:
  - Consider these two lines
  - `String s = “hello”;`
  - `s = s + “ world”;`
  - This will produce the *String* “hello world”
  - Note: The second line can also be written as `s += “ world”;`

System.out.println

- `System.out.println();`
- Takes the *String* between the brackets and prints it out
- This is a method, as will be discussed next week
- We can build a *String* within the brackets, or pass in a variable
- `System.out.println("hello " + "world");` or `System.out.println(s);`

Outputting a Value

- It's much nicer to print explaining text when you print a variable's value
- Example:
  - `System.out.println(“There are this many students:” + numStudents);`
- Note that this is concatenation of a *String* literal with an integer value
- The plus operator can be very tricky sometimes
  - We'll get into the details later

Section 10

More Details on Variables

Assignments

- Using the equals sign means we are **assigning** a value to a variable
- `numStudents = 2;`
- **Assigning:** The variable on the left is assigned the value on the right

- Note that this is different from math you've done before
- `2 = numStudents;` doesn't work in programming

Type Matching

- When we typed `int numStudents;`, this means that the variable can only store integer numbers (whole numbers)
- `numStudents = 2;` is okay
- `numStudents = 3.5;` is not.
- The compiler will give us a compiler error **Error: incompatible types**.
- Java is **strongly typed**, so it checks if the values match the declared variable type

Calculation

- We can assign results of calculations to a variable
- `int x = 3 * 9;`
- `x = 2 + 3 * 2;`
  - Note here that the order of operations matters
  - Can of course use brackets `x = (2 + 3) * 2;`
- Note that the right side of the equals sign is calculated first
- Then assignment is performed

Step-by-step Operation

- When we perform operations, such as printing a value or performing an assignment
- It is important to think about what value the variable has **at that time**

```
1  public class NumStudents
2  {
3      public static void main(String[] args)
4      {
5          //initialize the variable to have the value 54
6          int numStudents = 54;
7          System.out.println(numStudents);
8
9          //assign the variable the new value 76
10         numStudent = 76;
11         System.out.println(numStudents);
12     }
13 }
```

The first statement prints 54, and the second statement prints 76

Multiple Variables

- Let's start making the value of variables depend on other variables

```
int x = 3 * 9; //x = 27
int y = x + 2; //y = 29
```

- Note that part of y's calculation is looking up the value of x **at the time of assignment**

Executing Step-by-Step

```
int x = 3 * 9; //x = 27
int y = x + 2; //y = 29

x = 10;
int z = x + 2; //z = 12
```

- If we change `x`, and make the same calculation for z as for `y`,
- z has a different value than y

Don't forget:

Java executes statements line-by-line
The result depends on the variable's values **at that time**

Temperature Conversion

- We'll write a useful program, TemperatureConversion
- This will convert from Fahrenheit to Celsius

```
1  public class TemperatureConversion
2  {
3      public static void main(String[] args)
4      {
5          double f; //temp in Fahrenheit
6
7          double c;//the temp in Celsius
8
9          f = 100; //set the temp
10         c = (f-32)/1.8; //calculate the new temp
11
12         System.out.println(c); //prints out 37.7
13     }
14 }
```

- As practice, write a program to go from Celsius to Fahrenheit

TODO

- Install the JDK and Dr.Java/Eclipse/IntelliJ
- Look at the assignment
  - Especially the warm-up questions