

# Lecture 2 - Java Jumpstart

Bentley James Oakes

January 15, 2018

# *soup & science*



**McGill**

**Faculty of Science**

Office for Undergraduate  
Research in Science

Learn about cutting-edge research  
over lunch with cool profs

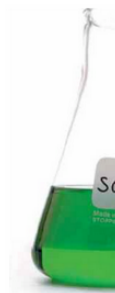
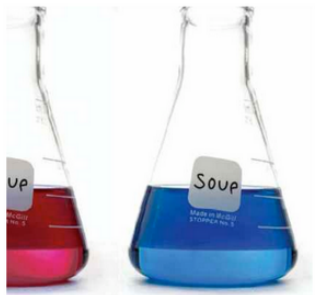
January 15-19, 2018

11:30 AM

Redpath Museum

More information:

[www.mcgill.ca/science](http://www.mcgill.ca/science)



My office hours will be:

- McConnell Building Room 233.
- Mon. and Wed. 13:30 to 15:00

# Assignment 1

Assignment is out on myCourses

- Due January 31st
- We'll cover all the material this week

# This Lecture

- 1 Binary
- 2 Java
- 3 Variables
- 4 Variable Types
- 5 Performing Calculation
- 6 Booleans
- 7 Mod Operator and Equality

# Section 1

## Binary

# 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

$$\text{Answer} = 128 + 64 + 8 + 2 + 1 = 203$$

- Output: 203

# 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



Anurag Roy found two websites to help you practice binary-to-decimal and decimal to binary conversions:

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

# Binary Addition

- Addition is a very common operation
- To add binary, just perform normal addition, but instead of carrying the one at 10, carry it at 2

Binary					Decimal
	1	0	1	0	10
+	0	0	1	1	3
<hr/>					
	1	1	0	1	13

Practice on your own with small numbers.

We won't ask for anything over four bits plus four bits (though assignment has larger numbers to practice on).

Double-check your results by converting to decimal.

An example with adding multiple numbers is on MyCourses.

Everything in programming/using a computer is based on binary  
 This is the mapping from binary numbers to characters  
 Don't memorize this

Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p

## Section 2

# Java

- Java is a high-level language from the 1990's
- Still very popular
- We write in Java to avoid writing in binary code

# Two Steps to a Java Program

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

This process is so programmers can:

- 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 3

# 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 in a room
  - Name: 'numStudents'
  - Type: Integer (int), to hold whole numbers
  - Value: 176



# Declaration

```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //A declaration of a variable
6         int numStudents;
7
8         //The initialization of this variable
9         numStudents = 176;
10
11        //prints out the value contained in the variable
12        //within the brackets
13        System.out.println(numStudents);|
14    }
15 }
```

- The statement `int numStudents;` **declares** 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

```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //A declaration of a variable
6         int numStudents;
7
8         //The initialization of this variable
9         numStudents = 176;
10
11        //prints out the value contained in the variable
12        //within the brackets
13        System.out.println(numStudents);|
14    }
15 }
```

- The next statement `numStudents = 176;` **initializes** the variable named 'numStudents' with the value 176
- This is called an **initialization**

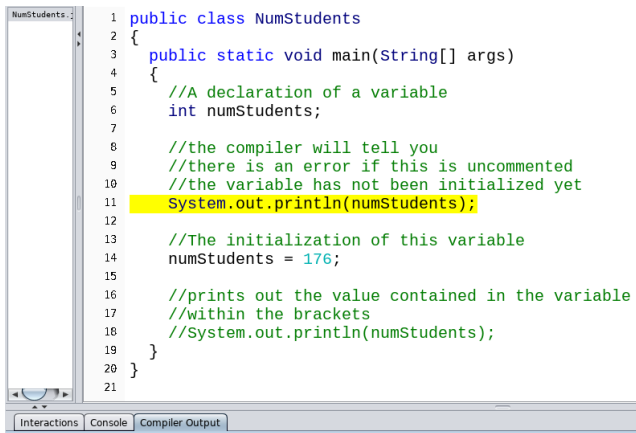
# Printing the Value of a Variable

```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //A declaration of a variable
6         int numStudents;
7
8         //The initialization of this variable
9         numStudents = 176;
10
11        //prints out the value contained in the variable
12        //within the brackets
13        System.out.println(numStudents);|
14    }
15 }
```

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

# 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 screenshot of an IDE window titled 'NumStudents.java'. The code is as follows:

```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //A declaration of a variable
6         int numStudents;
7
8         //the compiler will tell you
9         //there is an error if this is uncommented
10        //the variable has not been initialized yet
11        System.out.println(numStudents);
12
13        //The initialization of this variable
14        numStudents = 176;
15
16        //prints out the value contained in the variable
17        //within the brackets
18        //System.out.println(numStudents);
19    }
20 }
21
```

Line 11 is highlighted in yellow. At the bottom, the 'Compiler Output' tab is active, showing an error message.

**1 error found:**

**File:** /home/dcx/Dropbox/COMP 202/Lecture 1 - Binary Recap and Java/NumStudents.java [line: 10]

**Error:** variable numStudents might not have been initialized

# Declaration and Initialization

- Best way to solve this is to declare and initialize at same time
- `int numStudents = 176;`

```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //A declaration and initialization
6         //of a variable
7         int numStudents = 176;
8
9         //prints out the value contained in the variable
10        //within the brackets
11        System.out.println(numStudents);|
12    }
13 }
```

## Section 4

# Variable Types

- Java has quite a few variable types
- You'll only need to know how to use the types on this slide

Variable Type	In Declaration	Values
Integer	int	Whole numbers
Double	double	Non-whole numbers
String	String	Stores symbols
Boolean	boolean	True/False
Character	char	A symbol

Note that we use the second column when declaring a variable in Java

- `int numStudents = 2;` or `boolean isFunny = true;`

int - Integers (Whole numbers)

- Stores whole numbers (no decimal point)
- `int numStudents = 2;`
- `int numDogYears = 7;`
- `int age = 67;`



# Double Variable Type

double - Decimal numbers (like 1.4)

- Can store decimal values, like 3.5
- `double pi = 3.14159;`
- `double fractionOfPeopleWhoEatPie = 93.456;`

## String - Collection of characters

- Collections of characters
- `String s = "Hello";`
- Capital s in String
- Double quotation marks around String value
  - String values are also called 'String literals'
- Can get the length of the String, replace characters, etc.
- We'll see this later

- `System.out.println();`
- This is a **method**, which we will discuss this week
- This method takes whatever is within the brackets and prints it for the user
- Example: `System.out.println("Hello World!");`
- Here we are printing out a **String literal**
  - Again, a String literal is a piece of text with double-quotation marks

# Printing out a Variable

The method `System.out.println()`; can also print out the value of a variable

```
int numStudents = 25;
System.out.println(numStudents);

String hello = "Hello World!";
System.out.println(hello); //prints "Hello World!"
System.out.println("hello"); //prints hello
```

- Note that in the first two `System.out.println()`; statements, the value of the variable is printed
- In the last `System.out.println()`; statement, the String literal is printed
- The `System.out.println()`; statement is **evaluating** what's between the brackets first

- Evaluation means that `System.out.println()`; is determining the value of whatever is within its brackets

```
String s = "Hello";  
System.out.println(s); //prints "Hello"  
System.out.println(2 + 4); //prints 6
```

- In the first `System.out.println()`; statement, the value of the variable is printed
- In the second `System.out.println()`; statement, the calculation `2+4` is evaluated
- The result `6` is then printed

- This concatenation can be used to print out some nice text with a variable's value

```
System.out.println("Value of x: " + x);
```

- One last thing for `System.out.println()`...
- `System.out.println()` prints out, then starts a new line

```
System.out.println("Value of x: " + x);  
System.out.println("Value of y: " + y);
```

```
Value of x: 10  
Value of y: 29
```

- What if we want these on the same line?
- We can concatenate Strings together
- Or we can use `System.out.print()`;
- Note that the last word is **print**, not **println**
- `System.out.print()` prints out, then **doesn't** start a new line

```
System.out.print("Value of x: " + x);  
System.out.println(" Value of y: " + y);
```

Value of x: 10 Value of y: 29



## Section 5

# Performing Calculation

- Let's learn how to do some basic calculations
- Before we move onto different types of variables

# Assignments

- When performing calculations, we place the result in a variable
- We call this **assigning** a new value to a variable

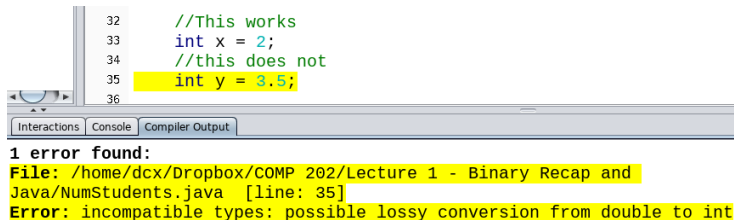
```
1 public class NumStudents
2 {
3     public static void main(String[] args)
4     {
5         //initialize the variable to have the value 54
6         int numStudents = 54;
7         System.out.println(numStudents);
8
9         //assign the variable the new value 76
10        numStudent = 76;
11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

Try this in your own program, and see what it prints

- Note that this is different from math you've done before
- `78 = numGrades;` doesn't work in programming
- **Assigning:** The variable on the left is assigned the value on the right
- `numGrades = 78;`

# Type Matching

- We need to assign correct types
- Typing `int x;` means that the variable can only store integer numbers (whole numbers)
- `int x = 2;` is okay
- `int y = 3.5;` is not.



```
32 //This works
33 int x = 2;
34 //this does not
35 int y = 3.5;
36
```

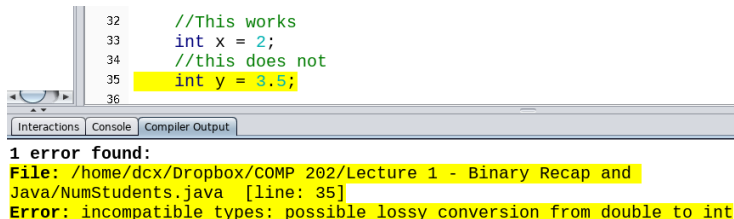
**1 error found:**  
**File:** /home/dcx/Dropbox/COMP 202/Lecture 1 - Binary Recap and Java/NumStudents.java [line: 35]  
**Error:** incompatible types: possible lossy conversion from double to int

- Every **expression** in Java has a type

Examples of literals:

- 2 is an int literal
- 3.5 is a double literal.
- “raining cats and dogs” is a String literal.
- true is a boolean literal.

# Incompatible Types



```
32    //This works
33    int x = 2;
34    //this does not
35    int y = 3.5;
36
```

1 error found:  
File: /home/dcx/Dropbox/COMP 202/Lecture 1 - Binary Recap and Java/NumStudents.java [line: 35]  
Error: incompatible types: possible lossy conversion from double to int

- The compiler will give us a compiler error **Error: incompatible types.**
- Java is **strongly typed**
- This means it checks if you are assigning or initializing the right type of value to a variable
- This can be annoying, but very helpful!

We can assign results of calculations to a variable

```
//x is set to 36  
int x = 4 * 9;  
//x is set to 17  
x = 2 + 3 * 5;
```

- Note that the order of operations matters
  - The multiplication happens before the addition
  - BEDMAS: Brackets - Exponents - Division/Multiplication - Addition/Subtraction
- You can use brackets to change the order  $x = (2 + 3) * 5;$



# 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

- Operations in Java have to agree on types
- Most of these rules are straightforward

Two examples of confusion:

- The plus operator `+`
  - Used for both addition and concatenation
- The division operator `/`
  - Used for both division and integer division

- What happens if we try to add together two Strings?

```
System.out.println("Hello " + "World!");  
//prints out Hello World!
```

- This is **concatenation**
- When “Hello” + “World!” is evaluated, the two Strings are joined together
- “Hello World!” is printed

# Concatenation vs Addition

- Java has consistent rules about what the + sign means
- But it can be confusing

```
System.out.println(3 + 5);
```

Prints 8 - Addition

```
System.out.println("3" + "5");
```

Prints 35 - Concatenation

```
System.out.println("3" + 5);
```

Prints 35 - Concatenation

```
system.out.println("3" + (1+4));
```

Prints 35

Calculation then concatenation

```
system.out.println(3 + 5 + "7");
```

Prints 87

Calculation then concatenation

Rule: If there's a *String* on either side of the + sign, then it's concatenation. The order of evaluation is left-to-right.

# Division vs Integer Division

- Java can be confusing with the / sign
- When it is between two ints, the result is an int
- Otherwise, it will produce a double value

```
double w = 99.0/25.0;
```

Result: 3.96 - Division

```
double x = 20/30.0;
```

Result: 0.666 - Division

```
int y = 99/25;
```

Result: 3 - Integer Division

```
int z = 3/4;
```

Result: 0 - Integer Division

```
int h = 4/2.0;
```

Result: Error - Right side is double

- 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**



## Section 6

# Booleans

- Often in programs we need to make comparisons and tests
- For example, if you are over a certain age, you get a senior's discount
- We could write this in English as:
  - Is your age over 65?
- And this will give either a *true* or *false* result

# Comparison Operators (For Numbers)

Recall these from math class

- $>$ 
  - Greater than
  - $4 > 3$  is *true*
- $<$ 
  - Less than
  - $5 < 2$  is *false*
- The alligator must be eating the bigger number for the **expression** to be *true*

- We also have
- $\geq$ 
  - Greater than or equal
  - $4 \geq 3$  is true
  - $4 \geq 4$  is also true
- $\leq$ 
  - Less than or equal

- So we are dealing with tests that give a *true* or *false* answer
- In programming, these values are called **Boolean values**
- And the calculation is called a **Boolean expression**
- Therefore, they are stored in **Boolean variables**

```
boolean b = 4 > 3;  
System.out.println(b);
```

- Here, a **Boolean variable** stores the result of a **Boolean expression**
- And the type of the variable is **boolean** (lower-case b)

# Testing Variables

```
1 public class TestingVars
2 {
3     public static void main(String[] args)
4     {
5         //a meal is 45 dollars
6         int mealPrice = 45;
7
8         //is mealPrice less than 10?
9         boolean mealIsCheap = mealPrice < 10;
10
11         System.out.println("The meal is: $" + mealPrice);
12         System.out.println("This meal is cheap: " + mealIsCheap);
13     }
14 }
```

```
    The meal is: $45
    This meal is cheap: false
```

Note that we can use `System.out.println()`; to print out the value of a boolean variable

## Section 7

### Mod Operator and Equality

- Let's take detour for a second
- How do we know if a number is odd or even?
- What's special about 2 and 4 where they are even?
- They are divisible by 2
- That is, when you divide 4 by 2, there is no remainder
- Wouldn't it be nice if there was a remainder operator so we could test odd/even?



- The remainder operator is called 'mod'
- Its symbol is the number sign: %

Examples:

- $2/2 = 1$ 
  - 2 divided by 2 gives 1
- $2\%2 = 0$ 
  - The remainder of 2 divided by 2 is 0

Let's look at examples mod 2

- $14 \text{ divided by } 2 = 7$
- $14 \% 2 = 0$
- $19 \text{ divided by } 2 = 9.5$
- $19 \% 2 = 1$
- If the remainder of dividing a number by 2 is 0,
- then the number is even
- otherwise, the number is odd

- Now that we have this remainder, we need to test if it is 1 or 0

== Test if equal  
!= Test if unequal

Examples:

- `3 == 3` is true
- `3 != 3` is false
- `4 == 3` is false
- `4 != 3` is true

# Testing Equality Example

```
int x = 3;  
boolean isThree = x == 3;  
System.out.println("X is three: " + isThree);  
X is three: true
```

```
int x = 3;  
boolean isThree = x == 3;  
System.out.println("X is three: " + isThree);
```

- = and == look alike, but do different things
- = is **assignment**
- == is **testing**
- You will have errors if you use them incorrectly

Remember:

- = is one word **assigned**
- == is two words **is equal**

# Finishing Up Our Odd/Even Test

- If we want to see if a value is even
- Get the remainder of the value when divided by 2
  - Calculate the value mod 2
- Test if this remainder is equal to 0
- If this test is true, the value is even

```
int remainder = 5 % 2; //gives 1
boolean isEven = remainder == 0; //gives false
System.out.println("5 is even: " + isEven);
```

# Comparing Different Variable Types

- Comparing different variable types works as you would expect
- $3.5 < 4$  is true
- $3 == 3.0$  is true
- $5 == "5"$  doesn't compile - can't compare these types
  - We'll talk about this case later