Kaitlin Hoffmann

Office Hours: SH 243

M 11:30 AM – 1:00 PM

T 2:00 – 3:00 PM

R 11:00 AM – 12:30 PM

Email: hoffmank4@newpaltz.edu

VARIABLES AND JAVA TYPES

COMPUTER SCIENCE I

OBJECTIVES

- Variables and Java Primitive Types

VARIABLES AND IMPORTANT DEFINITIONS

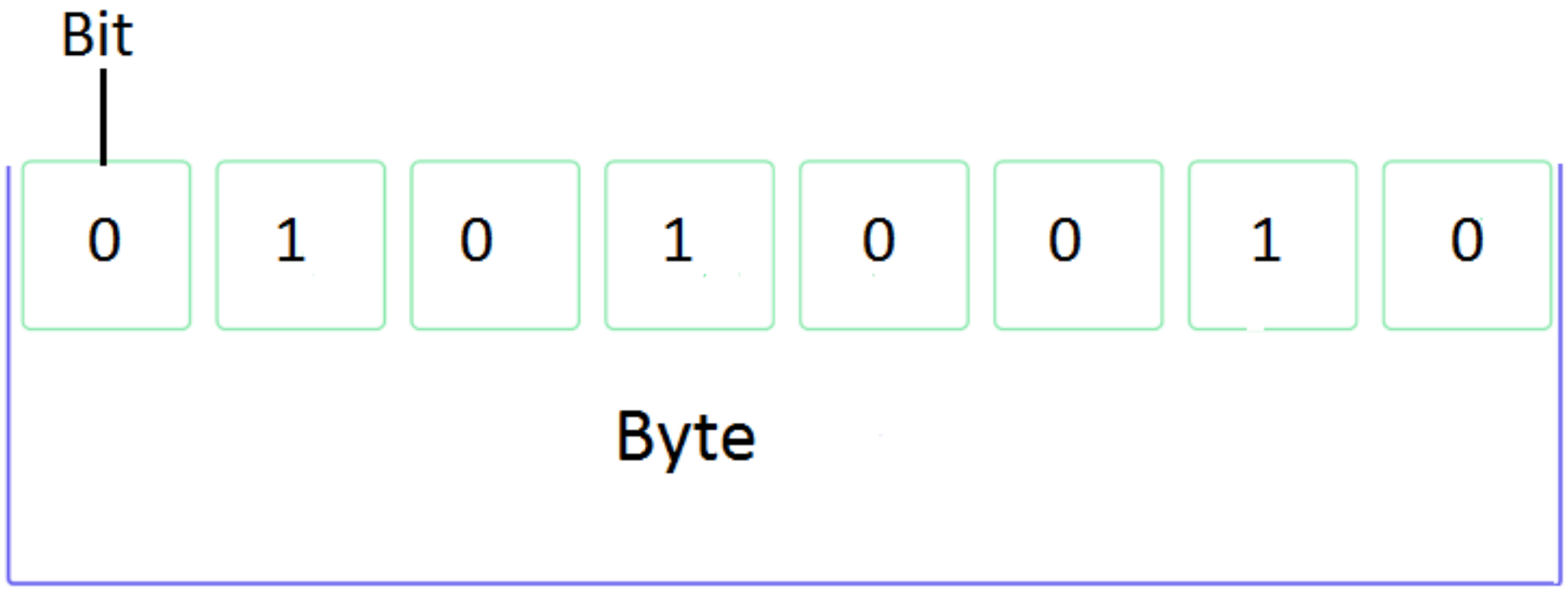
- **Variable**: a way to **label** a spot in memory to hold a value of the **type** of the variable (integers, Strings, characters, etc.). Basically, they hold some kind of data for us!
- **Initialize**: the first time we **assign** a value to a variable is called initializing the variable.
- **Declaration**: when we **create** the variable with a type and name, this is called declaring the variable.
- **Assignment**: when we **give a value** to a variable, it is called an assignment.

PRIMITIVE JAVA TYPES

* What you'll be using mostly in this course

type	range	Examples
byte	-128 to 127	
short	-32768 to 32767	
int *	≈ -2 billion to 2 billion	Integers …-2,-1,0,1,2…
long	$\approx -10^{19}\ to\ 10^{19}$	Can hold integers with more digits
float	$\approx -10^{38}\ to\ 10^{38}$	
double *	$\approx -10^{308}\ to\ 10^{308}$	Decimal numbers 3.14, -0.012, 2.0
char *	Can hold any single character	'A' to 'Z' 'a' to 'z' '?','!','@'
boolean *	Holds either true or false	

BYTE



8 bits = 1 Byte

- **Bit**: a binary value, 1 or 0.
- **Byte**: a set of 8 bits.
- You'll go in much greater detail with bytes if you take Assembly Language

SHORT, INT, LONG

- A **short**, **int** and **long** are all used to represent **integers** in Java (whole numbers – no decimals!).
- Recall that integers are a set of numbers: $-\infty$, ..., −5, −4, −3, −2, −1, 0, 1, 2, 3, 4, 5, ..., $+\infty$
- The difference between short, int and long is the range of integers that they can store:
  - **short**: [−32768, 32767]
  - **int**: [−2 billion, 2 billion]
  - **long**: [-10^{19}, 10^{19}]
- When we assign a value that is out of range of the declared data type of the variable, we call it **overflow**.

FLOAT AND DOUBLE

- A float and double are used to represent **decimals** in Java.
- Here we mean numbers with a decimal point in them.
  - ex. 3.7, −5.67, 1.0, etc.
- float and double are represented differently inside the computer and can store different ranges of values.
- A double is **more precise** and is better for storing **large numbers** than compared to a float (which is why we will use doubles in this course over float).

CHAR

- A char is used to represent a **single character** or **symbol** in Java.
- **Example**: 'A' to 'Z', 'a' to 'z', '0' to '9', '@', '$', '#', etc.
- We use **single quotes** (' ') to represent char in Java.

MORE ON CHAR

- **char** can be treated like **int** in Java.
- **ASCII** (American Standard Code for Information Interchange) is a character encoding system where characters have **integer equivalents**.
- https://www.ascii-code.com/

Code

```
System.out.println('c');

// looks like 'A' is equal to 65
System.out.println('A' + 3);

// Even numbers have an integer equivalent!
System.out.println('1' - 3);

// lower case and upper case are different numbers!
// 'A' is 65 and 'a' is 97
System.out.println('A' + 'a');
```

Results

```
c
68
46
162
```

ASCII AND CHAR

Dec	Char	Dec	Char	Dec	Char
32	SPACE	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_	127	DEL

BOOLEAN

- A boolean is a binary type that can be either **true** or **false** in Java.

Symbol	Meaning
==	Equals
!	Not
!=	Not equals
<, <=	Less than, less than or equal
>, >=	Greater than, greater than or equal
&&	And
||	Or

BOOLEAN EXAMPLES

Code

```
// checking if equal
System.out.println("1 == 1: " + (1 == 1));
System.out.println("1 == 4: " + (1 == 4));

// checking if not equal
System.out.println("1 != 1: " + (1 != 1));
System.out.println("1 != 4: " + (1 != 4));

// checking if less than, less than or equal
System.out.println("1 < 4: " + (1 < 4));
System.out.println("10 <= 4: " + (10 <= 4));

// checking if greater than or equal
System.out.println("1 >= 1: " + (1 >= 1));
```

Result

```
1 == 1: true
1 == 4: false
1 != 1: false
1 != 4: true
1 < 4: true
10 <= 4: false
1 >= 1: true
```

AND (&&)

- **AND**: is a logical operator that evaluates to true if **both** statements are true and false otherwise.
  - We use the symbol **&&** in Java

p	q	p && q
T	T	T
T	F	F
F	T	F
F	F	F

OR (||)

- **OR**: is a logical operator that evaluates to true if **one statement** is true and false if both statements are false.
  - We use the symbol **||** in Java

p	q	p || q
T	T	T
T	F	T
F	T	T
F	F	F

AND/OR EXAMPLES

Code

```
// checking if both are equal using AND
System.out.println((1 == 1) && (4 == 4));
System.out.println((1 == 1) && (4 == 2));

//checking if one statement is equal using OR
System.out.println((1 == 1) || (4 == 4));
System.out.println((1 == 1) || (4 == 2));
System.out.println((1 == 8) || (4 == 3));
```

Result

```
true
false
true
true
false
```

*Not only used with ==. You can use on =!, <, <=, >, >= as well.

BOOLEAN, AND, OR

- How would you check if 2+2 is **equal** to 4 **AND** 3 + 4 is **equal** to 7 using a print statement?

```
System.out.println((2 + 2 == 4) && (3 + 4 == 7));
```

- How would you check if 5 + 6 is **not equal** to 12 **OR** 3 + 4 is **less than** 5 using a print statement?

```
System.out.println((5 + 6 != 12) || (3 + 4 < 5));
```

STRINGS — THEY ARE NOT A PRIMITIVE JAVA TYPE!!

- Notice how Strings are **NOT** on the table. This is because they are not primitive. Strings are actually a Java **object**. We will get into Strings and objects later.

BACK TO VARIABLES!

- **Why do we need variables?** Variables give us a way to store data or information so we can use it many times.
- We declare a variable by **type** followed by a **name**.
- Naming conventions for variables:
  - should **start** with a lowercase letter
  - contain **no** spaces
  - for multi-word names use **camel case**: thisIsCamelCase
  - for multi-word names you can also use **underscores**: this_is_an_example
  - **cannot** use Java reserved words i.e. class, public, etc.
  - should be **descriptive**, if possible

DECLARATION OF VARIABLES

- **Declaration**: when we **create** the variable with a type and name, this is called declaring the variable.

Declaration	Visual
int x;	
long y;	
double z;	
char c;	
boolean b;	

*Notice there is no data yet

INITIALIZING A VARIABLE

- **Initialize**: the first time we **assign** a value to a variable is called initializing the variable.

```
int x;        //declares an int variable named x
```

x 

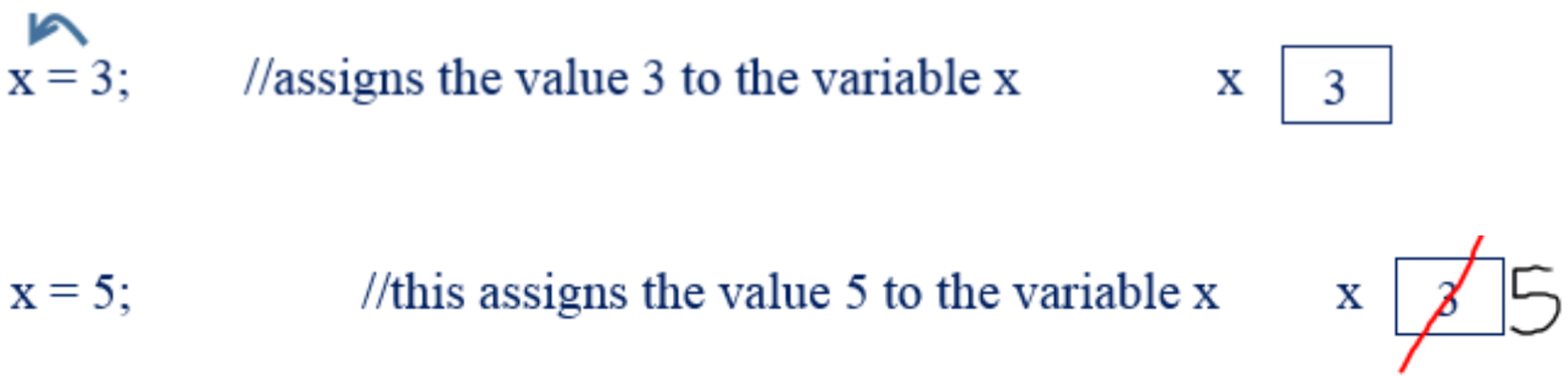
```
x = 3;        //assigns the value 3 to the variable x
```

x | 3

***Assignments go from right to left**

***Note that "=" means assigns not equals**

CHANGING THE VALUE OF A VARIABLE



- **Assignment**: when we **give a value** to a variable, it is called an assignment.
- **Another Example:**

Code

```
int i; // declare variable
i = 10; // initialize variable
System.out.println("The value of i is: " + i);
i = 5;  // assign new value
System.out.println("The value of i is now: " + i);
```

Output

```
The value of i is: 10
The value of i is now: 5
```

DECLARING AND INITIALIZING IN ONE STEP

Code

```java
// declaring and initializing
int id = 7286;
double hourlyWage = 25.55;
char middleInitial = 'C';

System.out.println("Employee Id: " + id + ", Hourly Wage: "
        + hourlyWage + ", Middle Initial: " + middleInitial);
// notice above how you can go to the next line since white space is ignored
```

Output

```
Employee Id: 7286, Hourly Wage: 25.55, Middle Initial: C
```

VARIABLES AND PRIMITIVE JAVA TYPES

Let's do the following:

1. Declare a variable of type **double** then assign it the value of **15.45**. Use two steps and one step.

2. Declare and assign a variable called letter that can hold the letter **T**.

3. Declare and assign a variable of type **int** then assign it the value **10**. Now change the value to **34**.

4. Declare a variable of type **double** and assign it **(3+2) * 2**.

VARIABLES AND PRIMITIVE JAVA TYPES

Let's do the following continued:

5. How can we write a program to find the **sum** of two variables, x and y? x has a value of **8.25** and y has a value of **2**.

VARIABLES AND PRIMITIVE JAVA TYPES

- Declare and initialize a **double** variable with the value **3.7** in one line. Now change the value to **–20.25**.
- Even though a String is NOT primitive, can you guess how you would declare and initialize a variable of type **String** with the text, “Hello World!”?
- Do you think the following is okay/possible?

```
int x = 5;
int y = 7;
y = x;
```

VARIABLES AND PRIMITIVE JAVA TYPES

- Do you think the following is okay/possible?

1. char c;
   char c2 = c;
2. int x = 5;
   int x = 7;
3. int y = 123;
   double z = y;
4. double a = 2.34;
   int b = a;

VARIABLES AND PRIMITIVE JAVA TYPES

- Do you think the following is okay/possible?

1. char c;
   char c2 = c;
   ✗ A variable must be initialized first before assigning it to another variable.
2. int x = 5;
   int x = 7;
   ✗ A variable name can only be declared once.
3. int y = 123;
   double z = y;
   ✓ An int can be a double…
4. double a = 2.34;
   int b = a;
   ✗ But a double cannot be an int.

VARIABLES AND PRIMITIVE JAVA TYPES

- Which of the following are allowed?

1. int x = 2.3;
2. double y = 4;
3. int a = ‘D’;

VARIABLES AND PRIMITIVE JAVA TYPES

- Which of the following are allowed?

1. int x = 2.3; ✗ A double can't be an int…
2. double y = 4; ✓ But an int can be a double.
3. int a = 'D'; ✓ All characters are assigned a decimal number ('D' = 68):

https://www.ascii-code.com/

THREE WAYS TO INCREMENT A VARIABLE

```
int x = 5;

x++;  //this way can only be used to increase by 1
x = x + 1; //this way and the one below can use ANY number
x+=1;
```

- All three ways are increasing x by 1.
- Use whichever you like best. However, **x++** can only be used for 1 increment.

THREE WAYS TO DECREMENT A VARIABLE

```
int x = 5;

x--;  //this way can only be used to decrease by 1
x = x - 1; //this way and the one below can use ANY number
x-=1;
```

- All three ways are decreasing x by 1.
- Use whichever you like best. However, **x- -** can only be used for 1 decrement.

OTHER OPERATIONS

- You can use the following as a shortcut for other operations too:

```
int x = 5;

x = x + 8; //addition
x+=8;

x = x -2; //subtraction
x-=2;

x = x * 7; //multiplication
x*=7;

x = x / 1; //division
x/=1;
```