

ICS 111

Introduction to Computer Science I

Nikki Manuel

Leeward Community College

A dark blue diagonal gradient bar that starts from the bottom left corner and extends towards the top right corner, covering the lower half of the slide.

Variables, Data Types, Arithmetic Operators

Week 2
Fall 2019

We know how to print out statements...

We already learned the syntax for `print` and `println`:

- `System.out.print("Hello!");`
- `System.out.println("Goodbye!");`

The text between the quotation marks is called a **string**.

- string: a sequence of characters
- strings must ALWAYS be between quotation marks

Printing Complex Messages

What if we want to print a long message?

What if we want to print many short messages?

What if we want to print a message with special characters in it?

Printing Long Messages

Coding standard: 80-132 characters per line

What if we want to print out a much longer message?

If you want to build a ship, don't drum up the men to gather wood, divide the work, and give orders. Instead, teach them to yearn for the vast and endless sea.

Printing Long Messages – Will this work?

```
System.out.println("If you want to build a ship, don't drum  
up the men to gather wood, divide the work, and give  
orders. Instead, teach them to yearn for the vast  
and endless sea.");
```

Printing Long Messages – Will this work?

```
System.out.println("If you want to build a ship, don't drum  
up the men to gather wood, divide the work, and give  
orders. Instead, teach them to yearn for the vast  
and endless sea.");
```

Error! The line cannot end like this



Printing Long Messages – jGrasp Color Code

```
System.out.println("If you want to build a ship, don't drum  
up the men to gather wood, divide the work, and give  
orders. Instead, teach them to yearn for the vast  
and endless sea.");
```


String Concatenation

Concatenate: to combine/connect/link things together

In Java, we use the plus sign **+** as a *string concatenation operator*
– Think of it as *adding* strings together

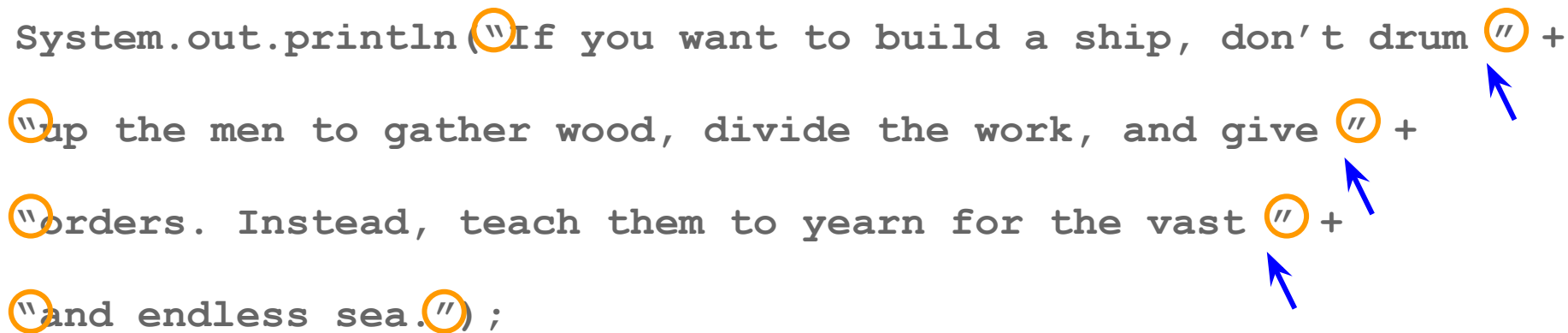
This allows us to combine strings together to form a longer one

Printing Long Messages – Concatenation

```
System.out.println("If you want to build a ship, don't drum " +  
"up the men to gather wood, divide the work, and give " +  
"orders. Instead, teach them to yearn for the vast " +  
"and endless sea.");
```

Printing Long Messages – Concatenation

```
System.out.println("If you want to build a ship, don't drum  
up the men to gather wood, divide the work, and give  
orders. Instead, teach them to yearn for the vast  
and endless sea.") ;
```



Printing Many Short Messages

Let's say "Hello!" in different languages.

```
System.out.println("Hello!");
```

```
System.out.println("Hola!");
```

```
System.out.println("Bonjour!");
```

```
System.out.println("Konnichiwa!");
```

```
System.out.println("Marhaba!");
```

```
System.out.print("Privet!");
```

Using \n

In Java, instead of using `println`,
we can use `\n` to move to the next line
(like pressing the *Enter* key):

```
System.out.println("Hello! \nHola! \nBonjour! \nKonnichiwa");
```

Escape Sequences

`\n` is an example of what we call an **escape sequence**

Escape sequences let us print characters we normally cannot print.

For example, we cannot simply press the *Enter* key, so instead we type `\n` to move to the next line.

Escape Sequences : Rules

Escape sequence rules:

- Begins with a backslash \
- No space between the backslash and the next character
- Must always be inside quotes

Printing Special Characters

Escape sequences can also be used to print special characters

Let's print this quote:

"Do or do not -- there is no try."

-- Yoda

Printing Special Characters – Will this work?

```
System.out.println(“Do or do not – there is no try.”);
```

ERROR!

Remember, when we print a message, Java looks for the message to be between two quotation marks.

In the above example, it sees the first quotation mark, then sees the second one, and expects there to be a message between them.

Escape Sequences : \' and \"

We can't print " -- so we need an escape sequence!

To print a single quote ' or double quote "

– Use \' or \"

```
System.out.println("\Do or do not - there is no try.\"");
```

Escape Sequences : \t

Now let's try to attribute the quote to Yoda. We want our output to look like this:

```
Do or do not - there is no try.  
                -- Yoda
```

However, we can't just press the *Tab* key, we need an escape sequence. To insert a "tab" we use **\t**:

```
System.out.print("\t\t\t-- Yoda");
```

Escape Sequences: Printing the \ character

Since we use \ for escaping, how can we print the \ character?

```
System.out.println("\\");
```

So what happens if we do this?

```
System.out.println("\\\\");
```

Escape Sequences Review

Begin with a backslash `\`

No space between the backslash and the next character

To print ' or " → use `\'` or `\"`

To print a tab → use `\t`

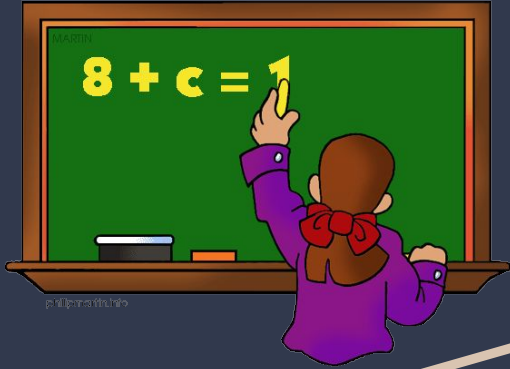
To skip a line → use `\n`

To print `\` → use `\\`

Variables



What is a variable?



$$a + b = 10$$

A **variable** stores values that will be used later in the program.

We call it a variable because the value that it represents can be changed - it varies!

Variable Rules

- A Java variable should be of a certain *data type*
- A Java variable should be *declared* before it is used
- A Java variable may be *initialized* to a convenient initial value.

A Java variable should be of a certain data type.



A java variable should be of a certain data type.

A **data type** specifies the kind of information that the variable will hold.

Java provides simple data types for representing integers, real numbers, characters, and Boolean types.

These types are known as **primitive data types**.

Primitive Data Types

Primitive data types are the building blocks of Java.

Here are a few we'll be learning about soon:

int

double

char

boolean

A Java variable should be **declared**
before it's used.



A Java variable should be declared before it's used.

To use a variable, you **declare** it by telling the compiler its name as well as what type of data it can store.

This tells the compiler to allocate the necessary memory space for the variable based on its data type.

Syntax for Declaring a Variable

The syntax for declaring a variable is:

```
datatype variableName;
```

Here are some examples of variable declaration:

```
int age;           // Declare age to be an integer variable
```

```
double wage; // Declare wage to be a double variable
```

Syntax for Declaring a Variable

If variables are of the same type, they can be declared together!

```
datatype variable1, variable2, variable3;
```

The variables are separated by commas. For example:

```
int a,b,c;           // Declare a, b, c to be integer variables
```

variableName : Rules for Naming Variables

- Variable names should not begin with a number, but they can contain numbers: `abc123`
- Variable names may contain underscores: `abc_123`
- Lower camel case: `alphaBravoCharlie`
- Nouns - give them a meaningful name: `interestRate`

A Java variable may be initialized to a convenient initial value.



A Java variable may be **initialized**. . .

Variables often have **initial values**.

This means we state what the variable stands for.

For example:

```
int age = 100;
```

We are saying that `age` is an integer with the value of 100.

Syntax for Initializing Java Variables

Note how we can declare and initialize the variable in 1 step:

```
int age = 100;
```

This is the same as writing these two statements:

```
int age;           // Declare age to be an integer value.  
age = 100;         // Assign age the value 100.
```

Syntax for Initializing Java Variables


Remember how we could do this?

```
int a,b,c;
```

We can also do this!

```
int a=1, b=2, c=3;
```

Data Types



Data Types: Numbers

For numbers, we can use:

- **short** (16 bits)
- **int** (32 bits)
- **float** (32 bits)
- **double** (64 bits)

int and **double** are the most commonly used

Data Types: Characters

`char`

- A single letter or number
- An escape sequence
(this includes the \)
- A space
- Symbols like: `+` `-` `$` `!`

Data Types: Boolean

boolean

- true
- false

A boolean data type can have only one out of two values: true or false.

Let's Recap: Data Types

`int`

`double`

`char`

`boolean`

Remember: datatype variableName = value;

```
int x = 7;
```

```
double y = 7.7;
```

```
char letter = 'a';
```

```
char newline = '\n';
```

```
boolean isCorrect = true;
```

Constants



What is a constant?

A **constant** represents a permanent value.

We use them when we want to refer to some value that we don't expect to change.

$$a + b - 1 = 10$$

Benefits of Using a Constant

- You don't have to repeatedly type the same value if it is used multiple times.
- If you have to change the constant's value, you need to change it only in a single location in the source code.
- A descriptive name for a constant makes the program easy to read.

Constant Rules

- Like variables, we need to specify the type
- When naming a constant, you must write it in all UPPERCASE
- A constant must be declared and initialized in the same statement.

Syntax for Declaring a Constant

The syntax for declaring a constant is:

```
final datatype CONSTANTNAME = value;
```

Here are some examples of constant declaration:

```
final double PI = 3.14159265359;
```

```
final int STUDENTS_IN_ICS_111 = 20;
```

Now let's try it out!



Create a Program

- Declare and print different variable types and constants.
- Introduce errors by initializing variables with wrong values.

Using Variables : `int`

```
1  public class usingVariables {  
2      public static void main(String[] args) {  
3  
4          // Declare the variable x as an integer (2)  
5          int x = 2;  
6  
7          // Print the value of variable x  
8          System.out.print(x) ;  
9      }  
10 }
```

Using Variables : int

```
1  public class usingVariables {  
2      public static void main(String[] args) {  
3  
4          // Declare the variable x as an integer (2)  
5          int x = 2;  
6  
7          // Tells us the value of variable x is 2  
8          System.out.print("x =" + x);  
9      }  
10 }
```



concatenation

Using Variables : `int` + Introducing Errors

```
1  public class usingVariables {
2      public static void main(String[] args) {
3
4          // Declare the variable x as a decimal
5          int x = 2.5;
6
7          // Tells us the value of variable x
8          System.out.print("x =" + x);
9      }
10 }
```

Using Constants : `final datatype NAME = value;`

```
1  public class usingConstants {
2      public static void main(String[] args) {
3
4          // Declare the constant PI
5          final double PI = 3.14159;
6
7          // Print the value of PI
8          System.out.print("Pi is equal to:" + PI);
9      }
10 }
```

String Variables



What is a string?

A **string** is a sequence of characters or words.

It is a data type, but it is NOT a primitive data type. It is made up of a primitive data type: characters!

A string is made up of characters

W	h	a	t		t	i	m	e		i	s		l	u	n	c	h	?
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

“What time is lunch?”

This string is made up of 19 characters

Syntax for Declaring a String

The syntax for declaring a declaring a string is:

```
String stringName = "Type your string here";
```

Here are some examples of string declaration:

```
String greeting = "Welcome to Java!";
```

```
String myDogsName = "Snoopy";
```

Using Strings : `String variableName = "";`

```
1  public class usingStrings {
2      public static void main(String[] args) {
3
4          // Declare the string
5          String myDogsName = "Snoopy";
6
7          // Print out the value of the variable
8          System.out.print(myDogsName) ;
9      }
10 }
```

Using Strings : `String variableName = "";`

```
1  public class usingStrings {
2      public static void main(String[] args) {
3
4          // Declare the string
5          String myDogsName = "Snoopy";
6
7          // Print out a message using the variable
8          System.out.print("I love " + myDogsName);
9      }
10 }
```

Assignments vs. Comparisons



Assignments vs. Comparisons

In programming, = does not mean "is equal to"

= means **assignment**

example: `int x = 30`

== means **comparison**

It compares both sides to see if they are equal, then it returns a boolean: true/false

Comparison Operators

<i>Comparison</i>	<i>Symbol</i>
Greater than	>
Less than	<
Greater than or equal to	>=
Less than or equal to	<=
Equal	==

Using Assignments and Comparisons

```
1  public class ComparingValues {
2      public static void main(String[] args) {
3
4          // Declare variables
5          int x = 1;
6          int y = 2;
7
8          // Print if x is equal to y
9          System.out.print(x == y);
10     }
11 }
```

Arithmetic Operators



Arithmetic Operators

<i>Operand</i>	<i>Meaning</i>	<i>Example</i>	<i>Result</i>
+	Add	$24 + 1$	25
-	Subtract	$10.0 - 2.5$	7.25
*	Multiply	$4 * 3$	12
/	Divide	$100 / 25$	4
%	Modulo	$20 \% 3$	2

What is modulo?

Modulo is the remainder after you divide.

For example, when you see:

$$20 \% 3$$

You are looking for the remainder of: $20 / 3$

20 divided by 3 is equal to 6, remainder 2

So, $20 \% 3$ is 2

Let's Practice Modulo

28 % 5

10 % 4

15 % 3

-7 % 6

13 % 1

Modulo is important!

Modulo is very useful in programming!

For example, an even number $\% 2$ is always 0, whereas an odd number $\% 2$ is always 1. So, you can use this to determine whether a number is odd or even.

Example of Using Modulo in Programming

If today is Saturday, it will be Saturday again in 7 days.

If your mom's birthday is in 10 days, what day will it be?

Let's start by thinking of the days as numbers:

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	1	2	3	4	5	6

Example of Using Modulo in Programming

Let's start by thinking of the days as numbers:

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	1	2	3	4	5	6

$$6 + 10$$

Saturday is day 6

We want the day 10 days later

Example of Using Modulo in Programming

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	1	2	3	4	5	6

$$(6 + 10) \% 7$$

Saturday is day 6

We want the day 10 days later

A week has 7 days

Example of Using Modulo in Programming

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
0	1	2	3	4	5	6

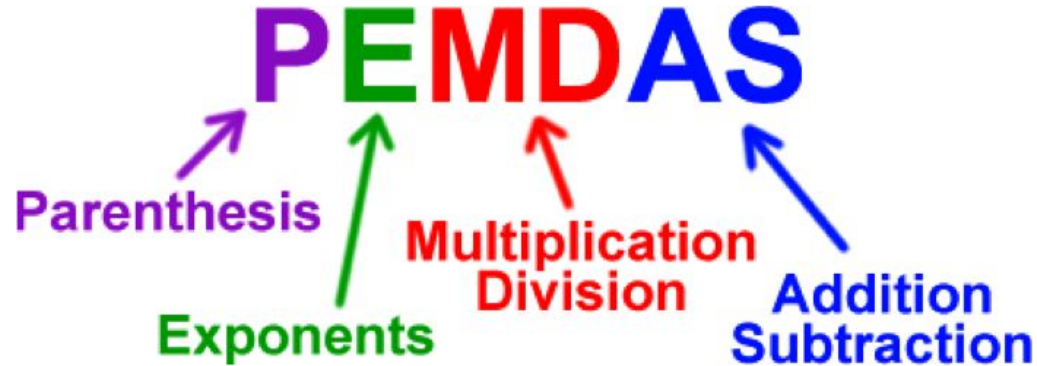
$$(6 + 10) \% 7 \text{ is } 2$$

Saturday is day 6

We want the day 10 days later

A week has 7 days

Arithmetic expressions follow the PEMDAS rule.



Modulo has the same priority as $*$ and \backslash

Important Note: Integer Division

When both numbers being divided are integers, the result is going to be an integer ... which might result in a wrong answer!

For example:

$5 / 2$ will show as 2 (not 2.5)

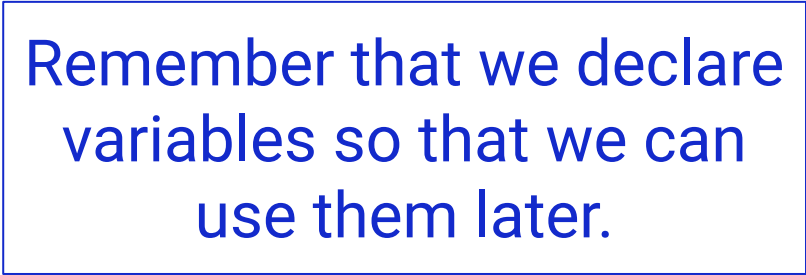
To show the correct answer, one number must be a decimal:

$5.0 / 2$ will show as 2.5

Variable Declaration and Arithmetic Operators

```
int x = 20  
int y = 4  
int add = 0;  
int subtract = 0;  
int multiply = 0;  
int divide = 0;
```

```
add = x + y;  
subtract = x - y;  
multiply = x * y;  
divide = x / y;
```



Remember that we declare variables so that we can use them later.

Augmented Operators



Augmented Operators

The *arithmetic operators* $+$, $-$, $*$, $/$, $\%$ can be combined with the *assignment operator* $=$ to form **augmented operators**.

Augmented Operators

Very often the current value of a variable is used, modified, and then reassigned back to the same variable:

```
x = x + 1;
```

Java allows us to combine arithmetic operators and assignment operators to instead rewrite that as:

```
x += 1;
```

Augmented Operators

<i>Operator</i>	<i>Example</i>	<i>Equivalent</i>
+=	x += 8	x = x + 8
-=	x -= 8	x = x - 8
*=	x *= 8	x = x * 8
/=	x /= 8	x = x / 8
%=	x %= 8	x = x % 8

Increment & Decrement Operators



Increment & Decrement Operators

The **increment operator** `++` and **decrement operator** `--` are two shortcuts used to add or subtract **1** from a variable.

Example:

```
x++;
```

```
y--;
```

You will find this to be very useful as you program!

Using Increment/Decrement Operators: Part 1

```
1  public class usingVariables {
2      public static void main(String[] args) {
3
4          // Declare the variable myAge
5          int myAge = 100;
6
7          // Print out my age
8          System.out.print("I am " + myAge + " years old.");
9          myAge++; // increment
10         System.out.print("\nNext year I'll be " + myAge);
11     }
12 }
```

Postfix vs. Prefix Operators

Postfix operators come *after* the variable:

```
x++;
```

Prefix operators come *before* the variable

```
++y;
```

Postfix vs. Prefix Operators

```
int x = 3, y = 3;
```

```
x++;           // x becomes 4
```

```
y--;           // y becomes 2
```

```
int x = 3, y = 3;
```

```
++x;           // x becomes 4
```

```
--y;           // y becomes 2
```

Postfix vs Prefix: Seems the same? Yes, but...

<i>Operator</i>	<i>Description</i>	<i>Example, if x = 1</i>
<code>++variable</code>	Increment <i>variable</i> by 1, and use the new value in the statement	<pre>int y = ++x; // y is 2, x is 2</pre>
<code>variable++</code>	Increment variable by 1, but use the old value in the statement	<pre>int y = x++; // y is 1, x is 2</pre>
<code>--variable</code>	Decrement variable by 1, and use the new value in the statement	<pre>int y = --x; // y is 0, x is 0</pre>
<code>variable--</code>	Decrement variable by 1, and use the old value in the statement	<pre>int y = x--; // y is 1, x is 0</pre>

Using Increment/Decrement Operators: Part 2

```
1  public class incrementDecrement {
2      public static void main(String[] args) {
3
4          int x = 10;
5          int newX = 10 * x++;
6          System.out.print("x is " + x + " newX is " + newX);
7
8          int y = 10;
9          int newY = 10 * (++y);
10         System.out.print("y is " + y + " newY is " + newY);
11     }
12 }
```

Using Increment/Decrement Operators : Part 3

```
1  public class incrementDecrement {
2      public static void main(String[] args) {
3
4          double x = 1.0;
5          double y = 5.0;
6          double z = x-- + (++y);
7
8          System.out.println("x: " + x);
9          System.out.println("y: " + y);
10         System.out.print("z: " + z);
11     }
12 }
```