Republic of the Philippines
City of Olongapo
GORDON COLLEGE
Olongapo City Sports Complex, Donor St., East Tapinac, Olongapo City
www.gordoncollege.edu.ph

Computer Programming 2

Module 2: Java Built-in Data Types

Name (LN,FN,MN): _________________________________ Program/Yr/Block: ___________

I. Introduction

This module will cover the built-in data types of Java which consists of primitive data types and the String class. In computer programming, a data type or simply type is an attribute of data which tells the compiler or interpreter how the programmer intends to use the data. Data types are important because it tells the compiler/interpreter how much memory will be allocated to the data that you will use in your code. It is also important for you to know the different operations that you can perform for a specific data type.

It is intended that you will perform all programming tasks in this module so that you can appreciate and experience how data types are used. Please feel free to use other references to learn more about Java data types. Enjoy coding!

II. Learning Objectives

After completing this module, you should be able to:

1. Understand the use and importance of data types in programming.
2. Declare and assign values to variables using built-in data types in Java.
3. Perform different operations to values assign to variables of a specific type.
4. Perform type casting to variables in your code.

III. Topics and Key Concepts

A. What is a Data Type?

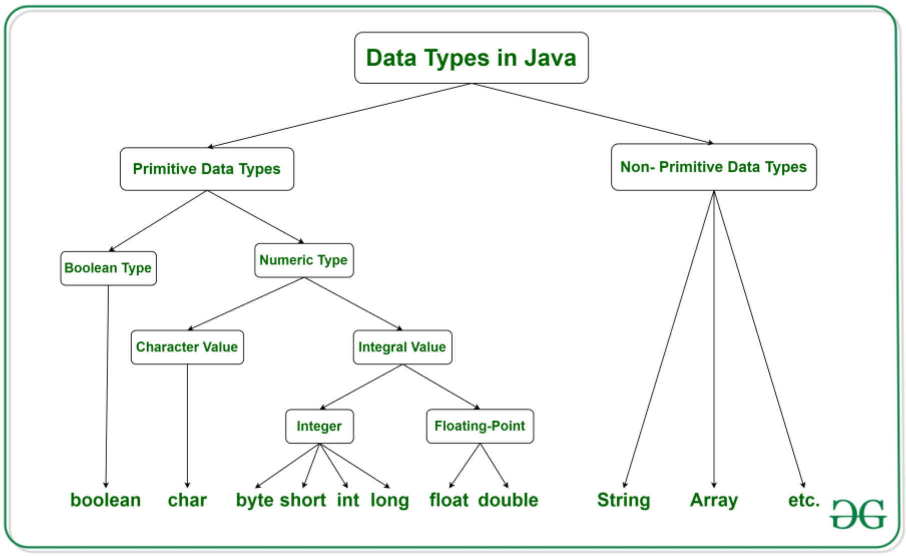
A data type is a set of values and a set of operations defined on them. For example, we are familiar with numbers and with operations defined on them such as addition and multiplication. Java has eight primitive data types that are used to store data during a program's operation.

Primitive data types are a special group of data types that do not use the keyword new when initialized. Java creates primitives as automatic variables that are not references and are stored in stack memory with the name of the variable. The most common primitive types used in this course are int (integers) and double (decimals).

We will also include the String type in this discussion. A String is an object that contains a sequence of characters. Declaring and instantiating a String is much like any other object variable. However, there are differences:

- They can be instantiated without using the new keyword.

- They are immutable.
- Once instantiated, they are final and cannot be changed.



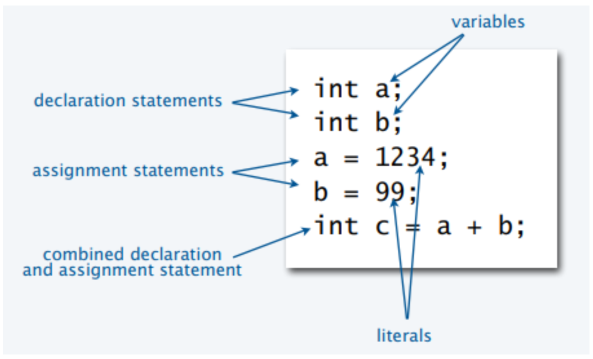
Java Data Types (© Geeksforgeeks.com)

type	*set of values*	*examples of values*	*examples of operations*
`char`	characters	`'A'` `'@'`	compare
`String`	sequences of characters	`"Hello World"` `"CS is fun"`	concatenate
`int`	integers	17 12345	add, subtract, multiply, divide
`double`	floating-point numbers	3.1415 6.022e23	add, subtract, multiply, divide
`boolean`	truth values	true false	and, or, not

Common Java built-in types

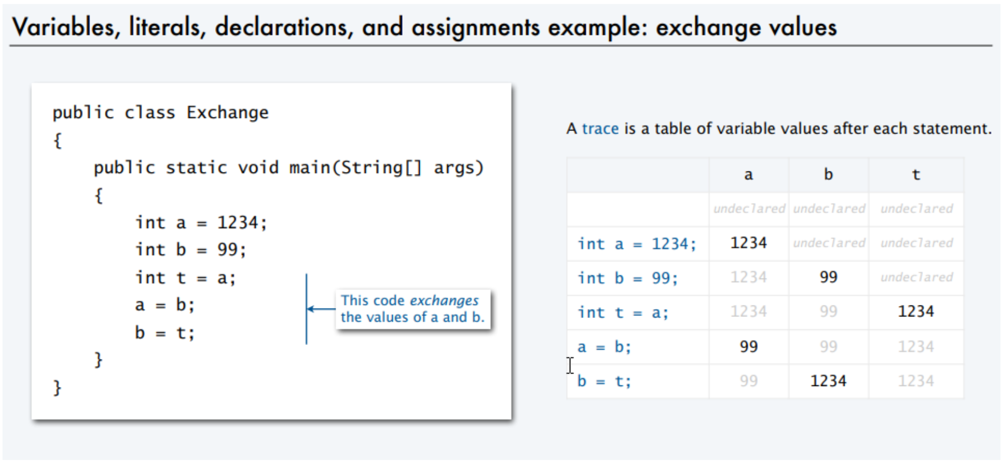
B. Basic Definitions

We use the following code fragment to introduce some terminology:



```
int a;
int b;
a = 1234;
b = 99;
int c = a + b;
```

- A **variable** is a name that refers to a value.
- A **literal** is a programming-language representation of a value.
- A **declaration statement** associates a variable with a type.
- An **assignment statement** associates a value with a variable.



Variables, literals, declarations, and assignments example: exchange values

```
public class Exchange
{
    public static void main(String[] args)
    {
        int a = 1234;
        int b = 99;
        int t = a;
        a = b;
        b = t;
    }
}
```

This code *exchanges* the values of a and b.

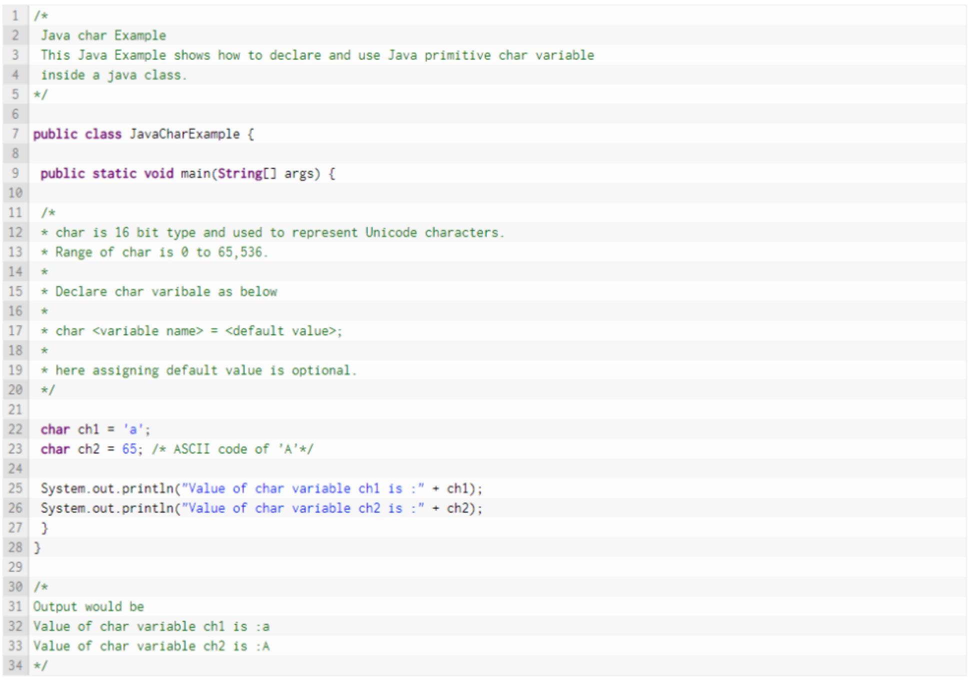
A trace is a table of variable values after each statement.

	a	b	t
	undeclared	undeclared	undeclared
int a = 1234;	1234	undeclared	undeclared
int b = 99;	1234	99	undeclared
int t = a;	1234	99	1234
a = b;	99	99	1234
b = t;	99	1234	1234

(Include statement to display variables a and b in Exchange.java)

C. Characters and Strings

A char is an alphanumeric character or symbol, like the ones that you type. We usually do not perform any operations on characters other than assigning values to variables.



```
/*
 Java char Example
 This Java Example shows how to declare and use Java primitive char variable
 inside a java class.
*/

public class JavaCharExample {

 public static void main(String[] args) {

 /*
 * char is 16 bit type and used to represent Unicode characters.
 * Range of char is 0 to 65,536.
 *
 * Declare char varibale as below
 *
 * char <variable name> = <default value>;
 *
 * here assigning default value is optional.
 */

 char ch1 = 'a';
 char ch2 = 65; /* ASCII code of 'A'*/

 System.out.println("Value of char variable ch1 is :" + ch1);
 System.out.println("Value of char variable ch2 is :" + ch2);
 }
}

/*
Output would be
Value of char variable ch1 is :a
Value of char variable ch2 is :A
*/
```

A sample Java code using the char data type

A character preceded by a backslash (\) is an escape sequence and has special meaning to the compiler. The following table shows the Java escape sequences:

Escape Sequences

Escape Sequence	Description
`\t`	Insert a tab in the text at this point.
`\b`	Insert a backspace in the text at this point.
`\n`	Insert a newline in the text at this point.
`\r`	Insert a carriage return in the text at this point.
`\f`	Insert a formfeed in the text at this point.
`\'`	Insert a single quote character in the text at this point.
`\"`	Insert a double quote character in the text at this point.
`\\`	Insert a backslash character in the text at this point.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes. To print the sentence:

```
She said "Hello!" to me.
```

you would write,

```
System.out.println("She said \"Hello!\" to me.");
```

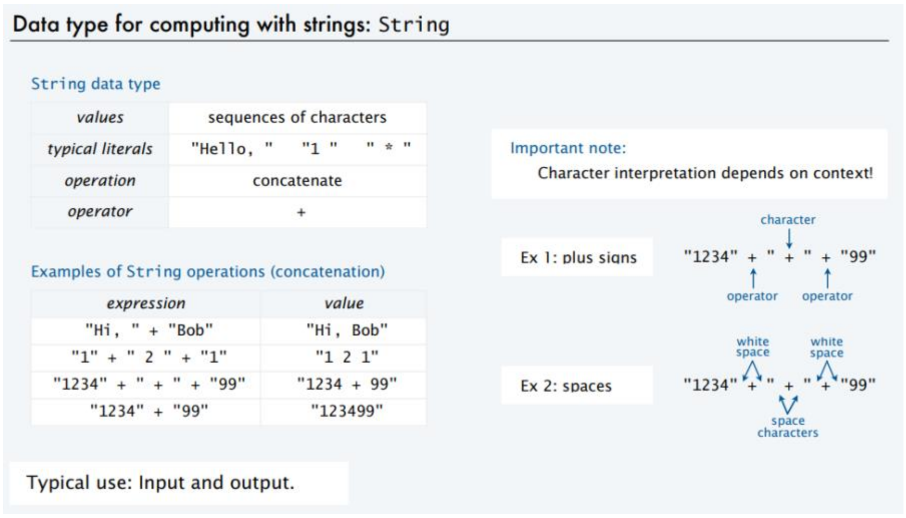
Strings, which are widely used in Java programming, are a sequence of characters. In the Java programming language, strings are objects.

The Java platform provides the String class to create and manipulate strings.

The most common operation that we perform on strings is known as concatenation: given two strings, chain them together to make a new string. For example, consider the following Java program fragment:

```
String a, b, c;
a = "Hello,";
b = " Bob";
c = a + b;
```

The first statement declares three variables to be of type String. The next three statements assign values to them, with the end result that c has the value "Hello, Bob".



Data type for computing with strings: String

String data type

values	sequences of characters
typical literals	`"Hello, "` `"1 "` `" * "`
operation	concatenate
operator	+

Examples of String operations (concatenation)

expression	*value*
`"Hi, " + "Bob"`	`"Hi, Bob"`
`"1" + " 2 " + "1"`	`"1 2 1"`
`"1234" + " + " + "99"`	`"1234 + 99"`
`"1234" + "99"`	`"123499"`

Important note:
Character interpretation depends on context!

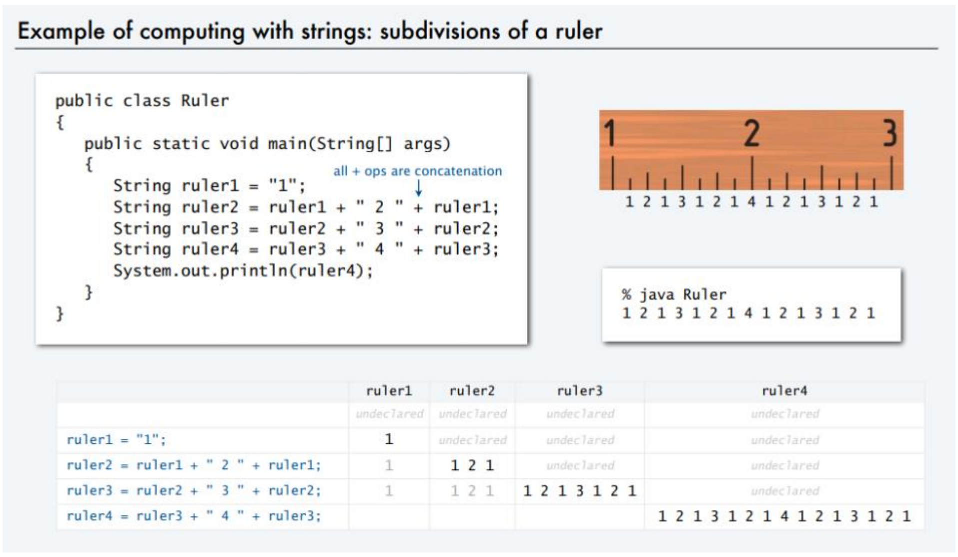
Ex 1: plus signs

`"1234" + " + " + "99"`

Ex 2: spaces

`"1234" + " + " + "99"`

Typical use: Input and output.



Example of computing with strings: subdivisions of a ruler

```
public class Ruler
{
   public static void main(String[] args)
   {
      String ruler1 = "1";
      String ruler2 = ruler1 + " 2 " + ruler1;
      String ruler3 = ruler2 + " 3 " + ruler2;
      String ruler4 = ruler3 + " 4 " + ruler3;
      System.out.println(ruler4);
   }
}
```

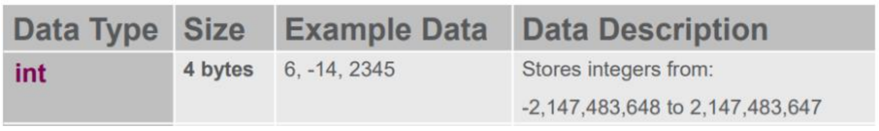
```
% java Ruler
1 2 1 3 1 2 1 4 1 2 1 3 1 2 1
```

	ruler1	ruler2	ruler3	ruler4
	undeclared	*undeclared*	*undeclared*	*undeclared*
`ruler1 = "1";`	1	*undeclared*	*undeclared*	*undeclared*
`ruler2 = ruler1 + " 2 " + ruler1;`	1	1 2 1	*undeclared*	*undeclared*
`ruler3 = ruler2 + " 3 " + ruler2;`	1	1 2 1	1 2 1 3 1 2 1	*undeclared*
`ruler4 = ruler3 + " 4 " + ruler3;`				1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Using string concatenation, Ruler.java prints the relative lengths of the subdivisions on a ruler.

D. Integers

An int is an integer (whole number) between −2,147,483,648 to 2,147,483,647. Standard arithmetic operators for addition, multiplication, and division, for integers are built into Java.



Data Type	Size	Example Data	Data Description
int	4 bytes	6, -14, 2345	Stores integers from: -2,147,483,648 to 2,147,483,647

In general, the int data type is the preferred data type when we create variables with a numeric value.

expression	*value*	*comment*
99	99	*integer literal*
+99	99	*positive sign*
-99	-99	*negative sign*
5 + 3	8	*addition*
5 - 3	2	*subtraction*
5 * 3	15	*multiplication*
5 / 3	1	*no fractional part*
5 % 3	2	*remainder*
1 / 0		*run-time error*
3 * 5 - 2	13	** has precedence*
3 + 5 / 2	5	*/ has precedence*
3 - 5 - 2	-4	*left associative*
(3 - 5) - 2	-4	*better style*
3 - (5 - 2)	0	*unambiguous*

Integer operations

D.1 Integer Types

The **byte** data type can store whole numbers from -128 to 127. This can be used instead of int or other integer types to save memory when you are certain that the value will be within -128 and 127:

```
byte myNum = 100;
System.out.println(myNum);
```

The **short** data type can store whole numbers from -32768 to 32767:

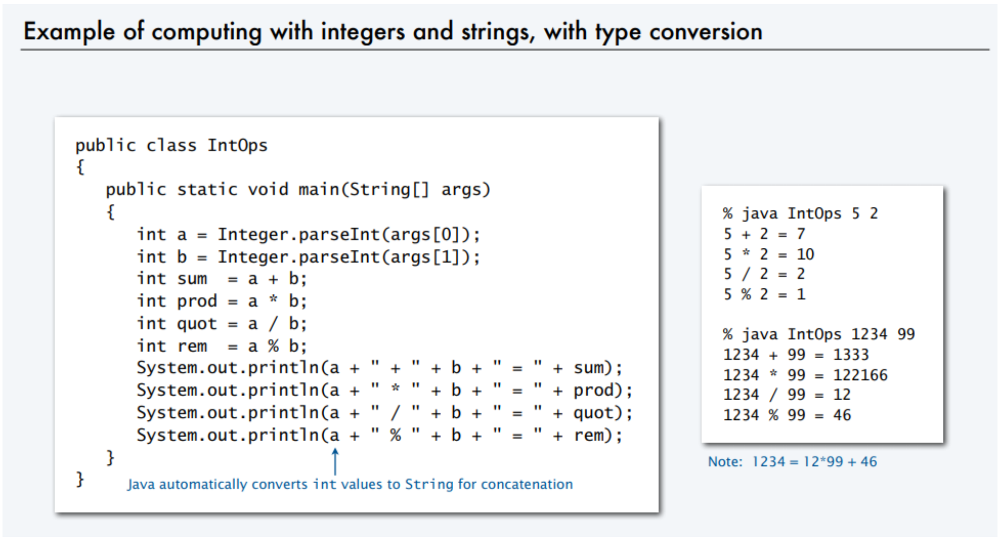
```
short myNum = 5000;
System.out.println(myNum);
```

The **int** data type can store whole numbers from -2147483648 to 2147483647:

```
int myNum = 100000;
System.out.println(myNum);
```

The **long** data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

```
long myNum = 15000000000L;
System.out.println(myNum);
```



Example of computing with integers and strings, with type conversion

```
public class IntOps
{
   public static void main(String[] args)
   {
      int a = Integer.parseInt(args[0]);
      int b = Integer.parseInt(args[1]);
      int sum  = a + b;
      int prod = a * b;
      int quot = a / b;
      int rem  = a % b;
      System.out.println(a + " + " + b + " = " + sum);
      System.out.println(a + " * " + b + " = " + prod);
      System.out.println(a + " / " + b + " = " + quot);
      System.out.println(a + " % " + b + " = " + rem);
   }
}
```

Java automatically converts int values to String for concatenation

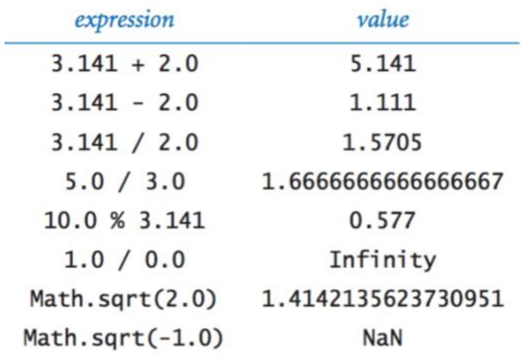
```
% java IntOps 5 2
5 + 2 = 7
5 * 2 = 10
5 / 2 = 2
5 % 2 = 1

% java IntOps 1234 99
1234 + 99 = 1333
1234 * 99 = 122166
1234 / 99 = 12
1234 % 99 = 46
```

Note: 1234 = 12*99 + 46

E. Floating-point Types

You should use a floating-point type whenever you need a number with a decimal, such as 9.99 or 3.14515. Standard arithmetic operators for addition, multiplication, and division, for doubles are built in to Java.



expression	*value*
3.141 + 2.0	5.141
3.141 - 2.0	1.111
3.141 / 2.0	1.5705
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
1.0 / 0.0	Infinity
Math.sqrt(2.0)	1.4142135623730951
Math.sqrt(-1.0)	NaN

E.1 Double data type

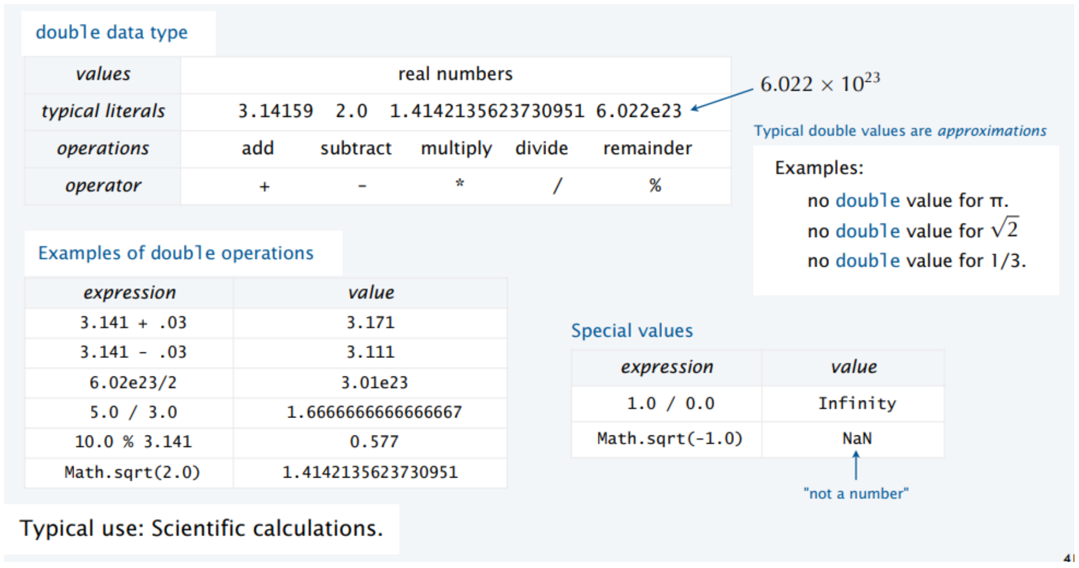
The double data type can store fractional numbers from 1.7e−308 to 1.7e+308. The maximum and minimum values are 17 followed by 307 zeros. Putting a "d" at the end of a double literal is optional.

```
double myNum = 19.99d;
System.out.println(myNum);
```

E.2 Float data type

The float data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that you should end the value with an "f":

```
float myNum = 5.75f;
System.out.println(myNum);
```



double data type

values	real numbers
typical literals	3.14159 2.0 1.4142135623730951 6.022e23
operations	add subtract multiply divide remainder
operator	+ - * / %

6.022×10^{23}

Typical double values are *approximations*

Examples:
no double value for π.
no double value for $\sqrt{2}$
no double value for 1/3.

Examples of double operations

expression	*value*
3.141 + .03	3.171
3.141 - .03	3.111
6.02e23/2	3.01e23
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
Math.sqrt(2.0)	1.4142135623730951

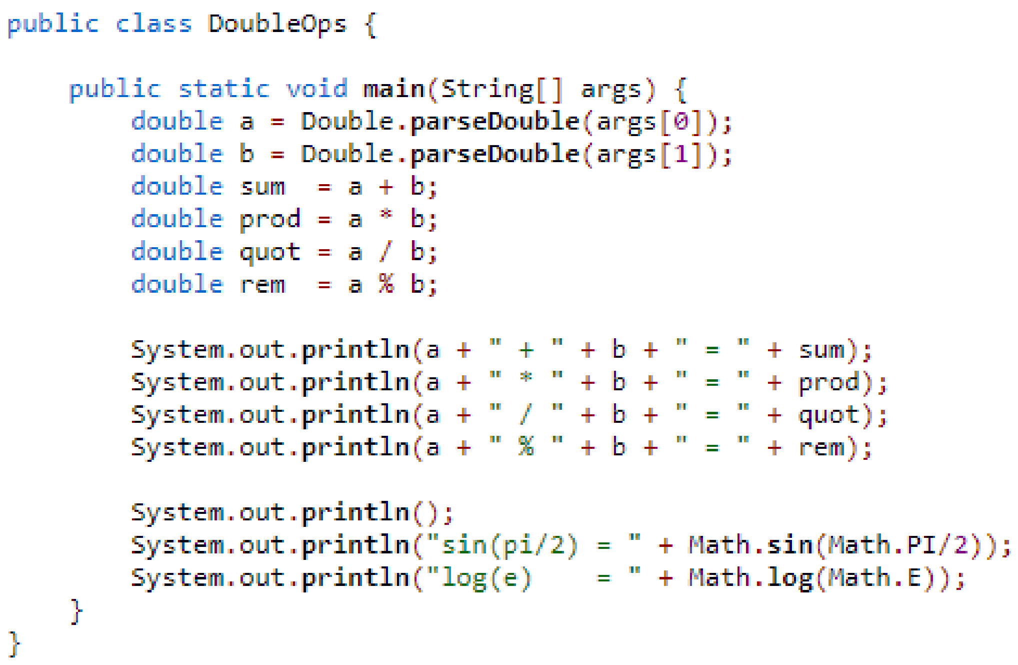
Special values

expression	*value*
1.0 / 0.0	Infinity
Math.sqrt(-1.0)	NaN

"not a number"

Typical use: Scientific calculations.

41



```
public class DoubleOps {

    public static void main(String[] args) {
        double a = Double.parseDouble(args[0]);
        double b = Double.parseDouble(args[1]);
        double sum  = a + b;
        double prod = a * b;
        double quot = a / b;
        double rem  = a % b;

        System.out.println(a + " + " + b + " = " + sum);
        System.out.println(a + " * " + b + " = " + prod);
        System.out.println(a + " / " + b + " = " + quot);
        System.out.println(a + " % " + b + " = " + rem);

        System.out.println();
        System.out.println("sin(pi/2) = " + Math.sin(Math.PI/2));
        System.out.println("log(e)    = " + Math.log(Math.E));
    }
}
```

A sample Java code that uses operations allowed for double data types

Additional Information for Numeric Data Types

1. Default Values

It's not always necessary to assign a value when a field is declared. Fields that are declared but not initialized will be set to a reasonable default by the compiler. Generally speaking, this default will be zero or null, depending on the data type. Relying on such default values, however, is generally considered bad programming style.

The following chart summarizes the default values for the above data types.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Local variables are slightly different; the compiler never assigns a default value to an uninitialized local variable. If you cannot initialize your local variable where it is declared, make sure to assign it a value before you attempt to use it. Accessing an uninitialized local variable will result in a compile-time error.

2. Numeric Literals

An integer literal is of type long if it ends with the letter L or l; otherwise, it is of type int. It is recommended that you use the upper-case letter L because the lower-case letter l is hard to distinguish from the digit 1.

Values of the integral types byte, short, int, and long can be created from int literals. Values of type long that exceed the range of int can be created from long literals. Integer literals can be expressed by these number systems:

- **Decimal**: Base 10, whose digits consists of the numbers 0 through 9; this is the number system you use every day
- **Hexadecimal**: Base 16, whose digits consist of the numbers 0 through 9 and the letters A through F
- **Binary**: Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later)

For general-purpose programming, the decimal system is likely to be the only number system you'll ever use. However, if you need to use another number system, the following example shows the correct syntax. The prefix 0x indicates hexadecimal and 0b indicates binary:

```
// The number 26, in decimal
int decVal = 26;
//  The number 26, in hexadecimal
int hexVal = 0x1a;
```

```
// The number 26, in binary
int binVal = 0b11010;
```

A floating-point literal is of type float if it ends with the letter F or f; otherwise, its type is double and it can optionally end with the letter D or d.

3. Underscores in Numeric Literals

In Java SE 7 and later, any number of underscore characters (_) can appear anywhere between digits in a numerical literal. This feature enables you, for example, to separate groups of digits in numeric literals, which can improve the readability of your code.

For instance, if your code contains numbers with many digits, you can use an underscore character to separate digits in groups of three, similar to how you would use a punctuation mark like a comma, or a space, as a separator.

The following example shows other ways you can use the underscore in numeric literals:

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi =      3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

You can place underscores only between digits; you cannot place underscores in the following places:

- At the beginning or end of a number
- Adjacent to a decimal point in a floating-point literal
- Prior to an F or L suffix
- In positions where a string of digits is expected

F. Boolean Type

The Boolean type has just two values: true or false. The apparent simplicity is deceiving—Booleans lie at the foundation of computer science. The most important operators defined for the Boolean are: and, or, and not.

- and: a && b is true if both a and b are true, and false otherwise.
- or: a || b is true if either a or b is true (or both are true), and false otherwise

- not: !a is true if a is false, and false otherwise.

Although these definitions are intuitive and easy to understand, it is worthwhile to fully specify each possibility for each operation in a truth table.

a	!a
true	false
false	true

a	b	a && b	a || b
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

The comparison operators are mixed-type operations that take operands of one type (e.g., int or double) and produce a result of type Boolean. These operations play a critical role in the process of developing more sophisticated programs.

op	meaning	true	false
==	equal	2 == 2	2 == 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

```java
public class LeapYear {
    public static void main(String[] args) {
        int year = Integer.parseInt(args[0]);
        boolean isLeapYear;

        // divisible by 4
        isLeapYear = (year % 4 == 0);

        // divisible by 4 and not 100
        isLeapYear = isLeapYear && (year % 100 != 0);

        // divisible by 4 and not 100 unless divisible by 400
        isLeapYear = isLeapYear || (year % 400 == 0);

        System.out.println(isLeapYear);
    }
}
```

The LeapYear.java program that uses Boolean operations

G. Type Conversion

We often find ourselves converting data from one type to another using one of the following approaches.

- **Explicit type conversion.** Call methods such as Math.round(), Integer.parseInt(), and Double.parseDouble().
- **Automatic type conversion**. For primitive numeric types, the system automatically performs type conversion when we use a value whose type has a larger range of values than expected.
- **Explicit casts.** Java also has some built-in type conversion methods for primitive types that you can use when you are aware that you might lose information, but you have to make your intention using something called a cast.
- **Automatic conversions for strings.** The built-in type String obeys special rules. One of these special rules is that you can easily convert any type of data to a String by using the + operator.

expression	*expression type*	*expression value*
`(1 + 2 + 3 + 4) / 4.0`	`double`	`2.5`
`Math.sqrt(4)`	`double`	`2.0`
`"1234" + 99`	`String`	`"123499"`
`11 * 0.25`	`double`	`2.75`
`(int) 11 * 0.25`	`double`	`2.75`
`11 * (int) 0.25`	`int`	`0`
`(int) (11 * 0.25)`	`int`	`2`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`Integer.parseInt("1234")`	`int`	`1234`

```java
public class RandomInt {
    public static void main(String[] args) {
        // a positive integer
        int n = Integer.parseInt(args[0]);

        // a pseudo-random real between 0.0 and 1.0
        double r = Math.random();

        // a pseudo-random integer between 0 and n-1
        int value = (int) (r * n);

        System.out.println(value);
    }
}
```

RandomInt.java reads an integer command-line argument n and prints a "random" integer between 0 and n−1.

H. Library Methods and APIs

Many programming tasks involve using Java library methods in addition to the built-in operators. An application programming interface is a table summarizing the methods in a library.

- Printing strings to the terminal window.

`void`	`System.out.print(String s)`	*print s*
`void`	`System.out.println(String s)`	*print s, followed by a newline*
`void`	`System.out.println()`	*print a newline*

- Converting strings to primitive types.

`int`	`Integer.parseInt(String s)`	*convert s to an `int` value*
`double`	`Double.parseDouble(String s)`	*convert s to a `double` value*
`long`	`Long.parseLong(String s)`	*convert s to a `long` value*

- Mathematical functions.

`public class Math`

`double`	`abs(double a)`	*absolute value of a*
`double`	`max(double a, double b)`	*maximum of a and b*
`double`	`min(double a, double b)`	*minimum of a and b*
`double`	`sin(double theta)`	*sine of theta*
`double`	`cos(double theta)`	*cosine of theta*
`double`	`tan(double theta)`	*tangent of theta*
`double`	`toRadians(double degrees)`	*convert angle from degrees to radians*
`double`	`toDegrees(double radians)`	*convert angle from radians to degrees*
`double`	`exp(double a)`	*exponential (e^a)*
`double`	`log(double a)`	*natural log ($\log_e a$, or ln a)*
`double`	`pow(double a, double b)`	*raise a to the bth power (a^b)*
`long`	`round(double a)`	*round a to the nearest integer*
`double`	`random()`	*random number in [0, 1)*
`double`	`sqrt(double a)`	*square root of a*
`double`	`E`	*value of e (constant)*
`double`	`PI`	*value of π (constant)*

You can call a method by typing its name followed by arguments, enclosed in parentheses and separated by commas. Here are some examples:

method call	*library*	*return type*	*value*
`Integer.parseInt("123")`	`Integer`	`int`	`123`
`Double.parseDouble("1.5")`	`Double`	`double`	`1.5`
`Math.sqrt(5.0*5.0 - 4.0*4.0)`	`Math`	`double`	`3.0`
`Math.log(Math.E)`	`Math`	`double`	`1.0`
`Math.random()`	`Math`	`double`	*random in* [0, 1)
`Math.round(3.14159)`	`Math`	`long`	`3`
`Math.max(1.0, 9.0)`	`Math`	`double`	`9.0`

IV. Teaching and Learning Materials Resources

- PC Computer | Laptop | Android Phone
- Gordon College LAMP
- Google Meet
- Facebook Messenger

V. Learning Tasks

A. Explore (60 points)

You are tasked to create/edit, compile and run the following sample codes that were used in this module:

1. Exchange.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

2. Ruler.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

3. IntOps.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

4. DoubleOps.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

5. LeapYear.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

6. RandomInt.java

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

B. Explain (50 points)

1. Give the type and value of each of the following expressions:
   a. (7 / 2) * 2.0
   b. (7 / 2.0) * 2
   c. "2" + 2

d. "2.0" + 2

2. What do each of the following print? Explain each outcome.
   a. System.out.println(2 + "bc");

   b. System.out.println(2 + 3 + "bc");

   c. System.out.println((2+3) + "bc");

   d. System.out.println("bc" + (2+3));

   e. System.out.println("bc" + 2 + 3);

3. A physics student gets unexpected results when using the code

```
double force = G * mass1 * mass2 / r * r;
```

to compute values according to the formula ***F = Gm1m2 / r***^{2}. Explain the problem and correct the code.

C. Engage

1. Modify MaxVariablesDemo.java to show minimum values instead of maximum values. You can delete all code related to the variables aChar and aBoolean. Show the output.

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

2. Write a program (SumOfInts.java) that accepts any number of integer values(arguments) from the command line and computes their sum (add them together). For example, suppose that you enter the following:

```
java SumOfInts 1 3 2 10
```

The program should display 16 and then exit. You can base your program on ValueOfDemo.java.

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

3. Write a program (SumOfFloats.java) that is similar to SumOfInts.java but has the following differences:

   - Instead of accepting integer values, it accepts floating-point arguments.
   - The sum to be displayed must have exactly two decimal digits.

   For example, suppose that you enter the following:

   ```
   java SumOfFloats 1 1e2 3.0 4.754
   ```

   The program would display 108.75.

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

4. Write a program WhatDayIsIt.java that accepts a date as input and displays the day (Sunday-Saturday) that date falls on. Your program should accept three integer values from the command-line: m (month), d (day), and y (year). For m use 1 for January, 2 for February, and so forth. For output print 0 for Sunday, 1 for Monday, 2 for Tuesday, and so forth. Use the following formulas, for the Gregorian calendar (where / denotes integer division):

$$y_0 = y - (14 - m) / 12$$

$$x = y_0 + y_0 / 4 - y_0 / 100 + y_0 / 400$$

$$m_0 = m + 12 \times ((14 - m) / 12) - 2$$

$$d_0 = (d + x + 31m_0 / 12) \bmod 7$$

For example, on which day of the week did February 14, 2000 fall?

```
y0 = 2000 - 0 = 1999
x = 1999 + 1999/4 - 1999/100 + 1999/400 = 2483
m0 = 2 + 12*1 - 2 = 12
d0 = (13 + 2483 + (31*12) / 12) mod 7 = 2528 mod 7 = 1  (Monday)
```

Your code (provide screenshot)

Sample Run: your terminal window (provide screenshot)

VI. References

- Oracle. nd. “Oracle Java Documentation”. https://docs.oracle.com/javase/tutorial/
- Oracle Academy. 2020. “Java Programming Instructor Resources”. https://academy.oracle.com
- Sedgewick Robert, Princeton University, Wayne, Kevin. 2017. “Introduction to Programming in Java: An Interdisciplinary Approach, 2nd Edition”. https://introcs.cs.princeton.edu/java/home/