Types and Math

Java has eight primitive types used in basic operations like arithmetic and logic. Types also define what values can be assigned to variables and passed to methods.

Manager: Recorder:

Presenter: Reflector:

Content Learning Objectives

After completing this activity, students should be able to:

- Explain what values can be assigned to primitive type variables.
- Apply methods from the Math class based on the documentation.
- Evaluate boolean expressions that include &&, ||, and ! operators.

Process Skill Goals

During the activity, students should make progress toward:

- Evaluating logic expressions based on operator precedence. (Critical Thinking)

Model 1 Primitive Types

Keyword	Size	Min Value	Max Value	Example
`byte`	1 byte	-128	127	`(byte) 123`
`short`	2 bytes	$-32,768$	$32,767$	`(short) 12345`
`int`	4 bytes	-2^{31}	$2^{31}-1$	`1234567890`
`long`	8 bytes	-2^{63}	$2^{63}-1$	`123456789012345L`
`float`	4 bytes	-3.4×10^{38}	3.4×10^{38}	`3.14159F`
`double`	8 bytes	-1.8×10^{308}	1.8×10^{308}	`3.141592653589793`
`boolean`	1 byte	N/A	N/A	`true`
`char`	2 bytes	0	65,535	`'A'`

Note that 1 byte is 8 bits, i.e., eight "ones and zeros" in computer memory. Since there are only two possible values for each bit, you can represent $2^8 = 256$ possible values with 1 byte.

Questions (15 min) Start time:

1. Which of the primitive types are integers? Which are floating-point?

2. Why do primitive types have ranges of values? What determines the range of the data type?

3. Why can't computers represent every possible number in mathematics? Will they ever be able to do so?

4. Since a `byte` can represent 256 different numbers, why is its max value 127 and not 128?

5. What is the data type for each of the following values?

1.14159	7.2E-4	-128
0	0.0	'0'
-1.0F	-13L	false
123	'H'	true

6. Based on the examples below, when does Java allow you to assign one type of primitive variable to another?

```
int int_ = 3;
long long_ = 3L;
float float_ = 3.0F;
double double_ = 3.0;

int_ = int_;
int_ = long_;      // illegal
int_ = float_;     // illegal
int_ = double_;    // illegal

long_ = int_;
long_ = long_;
long_ = float_;    // illegal
long_ = double_;   // illegal
```

```
float_ = int_;
float_ = long_;
float_ = float_;
float_ = double_;  // illegal

double_ = int_;
double_ = long_;
double_ = float_;
double_ = double_;

int_ = '0';
int_ = false;      // illegal
double_ = '0';
double_ = false;   // illegal
```

7. Given the following variable declarations, which of the assignments are not allowed?

```
byte miles;
short minutes;
int checking;
long days;
float total;
double sum;
boolean flag;
char letter;
```

```
checking = 56000;
total = 0;
sum = total;
total = sum;
checking = miles;
sum = checking;
flag = minutes;
days = '0';
```

Model 2 Math Methods

Consider the following methods defined in the `Math` class. (This list isn't exhaustive; the `Math` class has over 90 methods in total!)

Modifier and Type	Method and Description
`static int`	**`abs`**`(int a)` Returns the absolute value of an `int` value.
`static double`	**`log`**`(double a)` Returns the natural logarithm (base *e*) of a `double` value.
`static double`	**`pow`**`(double a, double b)` Returns the value of the first argument raised to the power of the second argument.
`static double`	**`random`**`()` Returns a `double` value with a positive sign, greater than or equal to `0.0` and less than `1.0`.
`static int`	**`subtractExact`**`(int x, int y)` Returns the difference of the arguments, throwing an exception if the result overflows an `int`.

The code for these methods is written in a file named *Math.java*. Here is what the definition of the `abs` method looks like:

```
public static int abs(int a) {
    // code omitted
}
```

To use a method from another source file (like *Math.java*), you must first specify the class name:

```
value = abs(-5);       // Error: cannot find symbol
value = Math.abs(-5);  // correct
```

Questions (15 min) Start time:

8. What type of value does `Math.random()` return? Give an example of what a random value might look like.

9. When *defining* a method (like `abs` or `log`), what do you need to specify before the method name and after the method name?

10. Define a method named `average` that takes two integers named `x` and `y` and returns a `double`. Don't write any semicolons or braces.

11. When *using* a method, what do you need to specify before the method name and after the method name?

12. For each method in Model 2, write a Java statement that uses the method and assigns the result to a variable.

*What you wrote for Question #10 is called the method's **signature**. The variables declared inside the parentheses are called **parameters**. When invoking a method, the values you provide are called **arguments**. Since arguments will be assigned to parameters, their types must be compatible.*

13. In the table below, how many parameters and arguments does each method have? What is the relationship between the last two columns?

Method	# Params	# Args
`abs`		
`log`		
`pow`		
`random`		
`subtractExact`		

14. Consider the statement `System.out.println("Price: " + price);` where the value of `price` is 9.99. What is the argument that `println` receives?

15. Consider the statement `System.out.printf("Price: %f", price);` where the value of `price` is 9.99. Why does `println` use *plus* and `printf` use *comma* to specify the arguments?

IMPORTANT: Never use + (string concatenation) with printf. You might accidentally add values to the format string itself, rather than substitute them.

Model 3 Conditional Operators

Boolean expressions, like `written > problem` and `teamwork < 75.0`, can be combined using the ***conditional operators***:

Operator	Meaning
`!`	not
`&&`	and
`||`	or

For example, `written > problem && teamwork < 75.0` is false, because `teamwork` is not less than `75.0`. (Both conditions need to be true in order for `&&` to be true.)

The following table summarizes the result of `&&`, `||`, and `!` for all possible inputs. The variables `p` and `q` represent conditions like `written > problem` and `teamwork < 75.0`.

`p`	`q`	`p && q`	`p || q`	`!p`
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Questions (15 min) Start time:

16. Consider the following variables:

```
double initiative = 74.2;
double analytical = 71.9;
double workEthic  = 70.8;
boolean hired = true;
boolean fired = false;
```

What are the results (true or false) of the following expressions?

Expression	Result
`!fired`	
`!(workEthic < initiative)`	
`workEthic < 71.0 && 71.0 < initiative`	
`initiative < 70.0 || workEthic > 70.0`	
`fired || workEthic < 50.0`	
`analytical < initiative && fired`	
`hired && !fired`	

17. Write a boolean expression that . . .

a) uses `initiative`, `analytical`, and `!`, and evaluates to false.

b) uses `analytical`, `workEthic`, and `!`, and evaluates to true.

c) uses any variable(s), and evaluates to false.

d) uses any variable(s), and evaluates to true.

18. Using your answers to the previous question, write a boolean expression "`p && q`" where `p` is your answer to part a) and `q` is your answer to part b).

a) Your expression:

b) Result of `p && q`:

Relational operators (<, >, and ==) are evaluated before conditional operators (!, &&, and ||). When multiple conditional operators are used, Java evaluates ! first, then &&, and finally ||.

19. Show the intermediate result of each operator below. In other words, show your work as you evaluate the code in the same order that Java would.

```
!(initiative < analytical) && workEthic > analytical
```

	Operator	Expression	Result
1st	`<`	`initiative < analytical`	false
2nd			
3rd			
4th			

20. Change the parentheses in the original expression (from the previous question) so that the `&&` is evaluated before the `!`. Then remove any unnecessary parentheses.

a) Expression:

b) New result:

21. Review the table from Model 3 for evaluating `&&` and `||`. Looking only at the `p` and `&&` columns, when is it necessary to examine `q` to determine how `p && q` should be evaluated?

22. Review the table from Model 3 for evaluating `&&` and `||`. Looking only at the `p` and `||` columns, when is it necessary to examine `q` to determine how `p || q` should be evaluated?

23. In Java, `&&` and `||` are ***short circuit*** operators, meaning they evaluate only what is necessary. If the expression `p` is more likely to be true than the expression `q`, which one should you place on the left of each operator to avoid doing extra work?

a) left of the `&&` expression:

b) left of the `||` expression:

24. What is the result of the following expressions?

a) `1 + 0 > 0 && 1 / 0 > 0`

b) `1 + 0 > 0 || 1 / 0 > 0`