Variables & Data Types

Today's Objectives

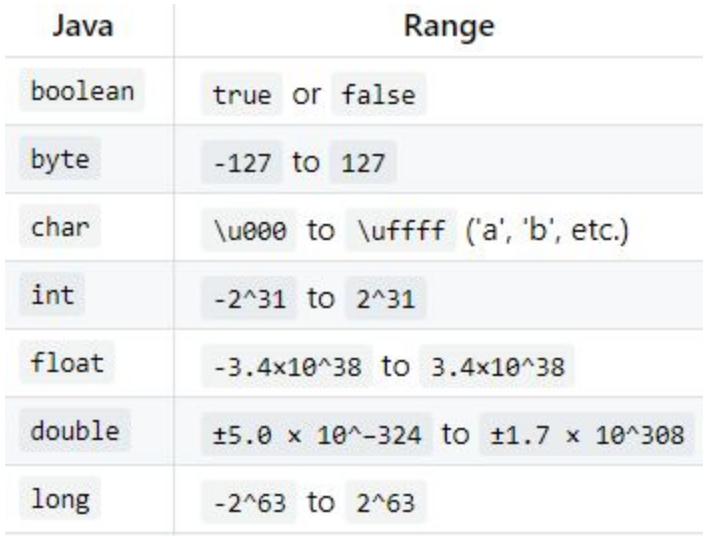
- Fundamental concepts & components of Java.
- Reading and writing code that uses variables.
- Declaring a variable and assigning a value to it.
- Industry accepted naming conventions.
- Choosing an appropriate primitive type to represent varying data.
- Using arithmetic operators to form mathematical expressions.
- Casting/data conversion: when it occurs, and why it’s used.
- Literal suffixes: what are they and when to use them.
- Code and test(debug) a simple program in the IDE.
- Perform simple IDE tasks such as:
    - Organize code into Projects
    - Understand basic syntax error feedback
    - Use Intellisense

HelloWorld: The Developer's Right of Passage

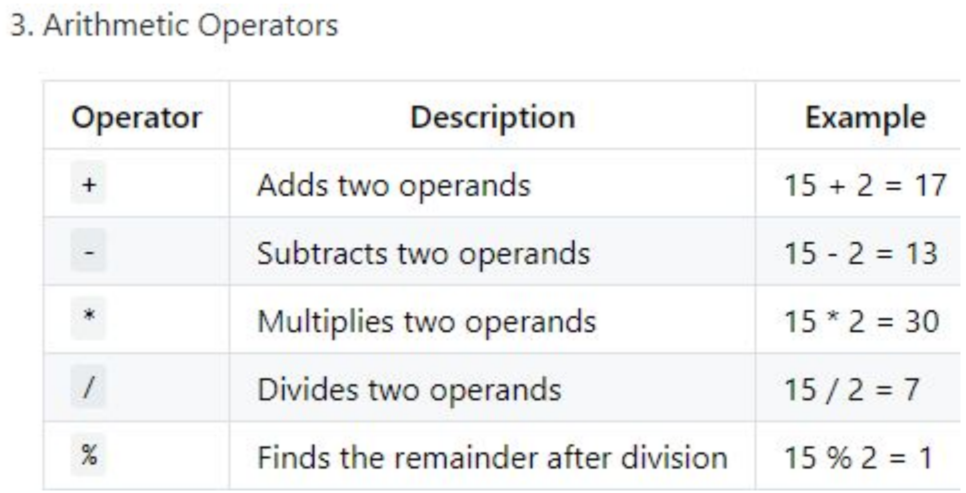
```java
package com.techelevator;

 public class HelloWorld {
     public static void main(String[] args) {
         // Prints out Hello World
         System.out.println("Hello, World!");
    }
}
```

Datatype and Operator Quick Reference



Java	Range
`boolean`	`true` or `false`
`byte`	`-127` to `127`
`char`	`\u000` to `\uffff` ('a', 'b', etc.)
`int`	`-2^31` to `2^31`
`float`	`-3.4x10^38` to `3.4x10^38`
`double`	`±5.0 × 10^-324` to `±1.7 × 10^308`
`long`	`-2^63` to `2^63`



3. Arithmetic Operators

Operator	Description	Example
`+`	Adds two operands	15 + 2 = 17
`-`	Subtracts two operands	15 - 2 = 13
`*`	Multiplies two operands	15 * 2 = 30
`/`	Divides two operands	15 / 2 = 7
`%`	Finds the remainder after division	15 % 2 = 1

History of Java

- Java is an object oriented language (you will learn what this means later!) developed at Sun Microsystems by James Gosling in 1995 and originally called Oak. Sun Microsystems was acquired by Oracle Corporation in 2010.
- Designed as a Write Once, Run Anywhere (WORA) language.
- It it is one of the most widely used languages. It consistently ranks in the top 2 in terms of popularity.
  (Source:
  https://stackify.com/popular-programming-languages-2018/)

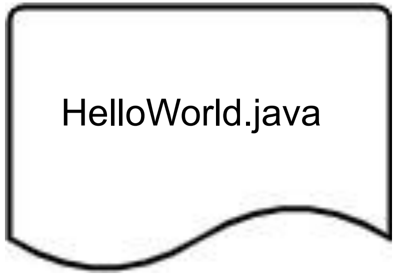
Java Features

- Syntax is similar to C/C++
- Solves some of the issues that made C/C++ harder to work with.
- Can be used to create desktop, mobile, and web applications.
- Unlike other languages, Java is not run natively on a given device, it is executed in a Java Virtual Machine (JVM) / Java Runtime Environment (JRE).... think of this as a virtual computer running inside your computer.
- Advantages of this model:
  - Oracle (not you 😉) is responsible for the lifecycle of the JRE / JVM.
  - Developers are therefore freed from the idiosyncrasies of each individual platform! (i.e. pc’s, macs, mobile devices, refrigerators, etc)

JVM Architecture

- JVM (Java Virtual Machine)
- A part of the JRE (Java Runtime Environment)
- Java programs run in a JVM
- Multiple implementations of JRE to allow WORA/platform independence
- javac (pronounced "java-see") is the primary Java compiler included in the Java Development Kit (JDK)
- The javac tool reads Java code (human readable) and compiles it into bytecode class files.

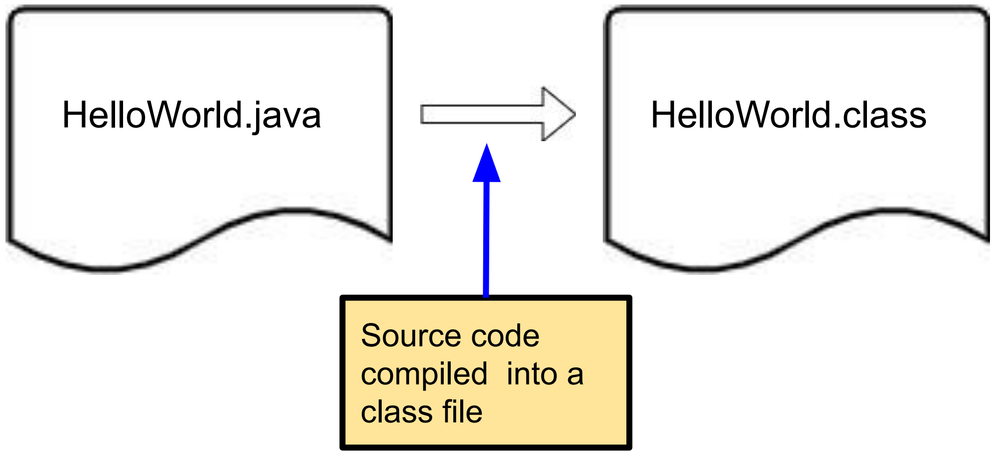
Java Development Workflow



Compiling is the process by which source code (in this case the file HelloWorld.java) is transformed into a language the computer can understand.

In the past, the command **javac** was used to compile code, since the advent of modern development tools, manually typing this command is no longer needed.

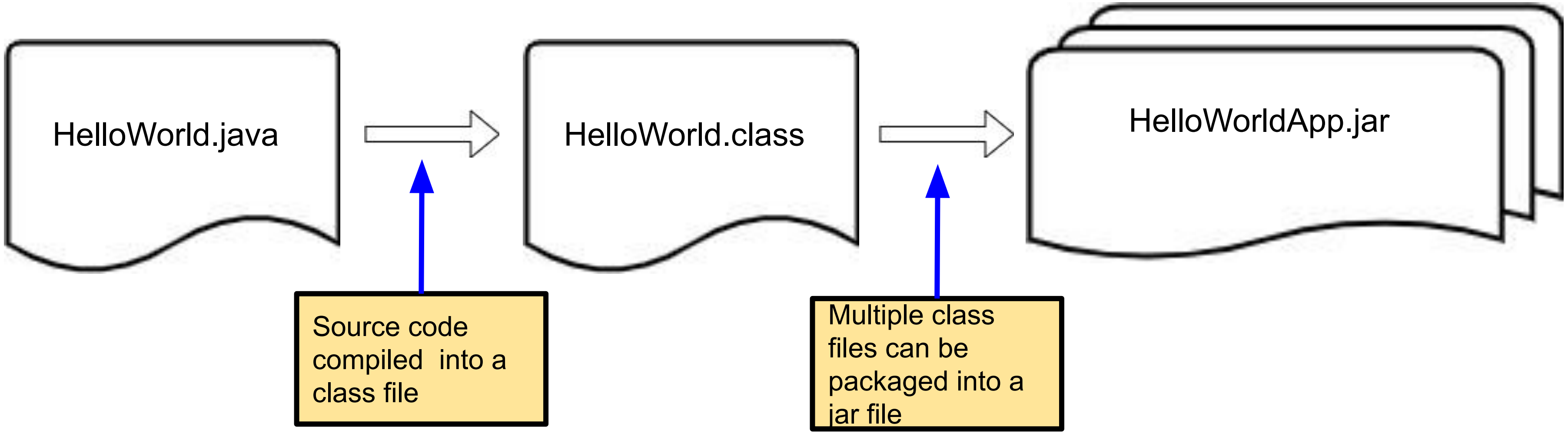
Java Development Workflow



Compiling is the process by which source code (in this case the file HelloWorld.java) is transformed into a language the computer can understand.

In the past, the command **javac** was used to compile code, since the advent of modern development tools, manually typing this command is no longer needed.

Java Development Workflow


HelloWorld.java
HelloWorld.class
HelloWorldApp.jar
Source code compiled into a class file
Multiple class files can be packaged into a jar file

Compiling is the process by which source code (in this case the file HelloWorld.java) is transformed into a language the computer can understand.

In the past, the command **javac** was used to compile code, since the advent of modern development tools, manually typing this command is no longer needed.

Java Bytecode Example

```
public static void main(String[] args) {
      System.out.println("Hello, world!");
}
```

This is what we will write

```
public static void main(String[] args) {
        Code:
        0: getstatic     #2                  // Field java/lang/System.out:Ljava/io/PrintStream;
        3: ldc           #3                  // String Hello, world!
        5: invokevirtual #4                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
        8: return
}
```

This is what the java compiler translates it to

Source: https://en.wikipedia.org/wiki/Java_bytecode

IDEs & IntelliJ IDEA

- Organize code into projects
  - A project includes a reference to one or more code files that belong to a module.
- Increase Efficiency – faster coding with less effort
  - Immediate feedback for syntax errors
  - Code assistance through ***intellisense***
- Debugging
  - Allows us to “step through” a program
- Cannot automatically fix errors, still need knowledge to code efficiently

Java Program Structure

- The main unit of organization in Java is a class. Here is a simple example:

Source code for this class is in the file **`FirstExample.java`**

```
public class FirstExample {

    public static void main(String[] args) {
        int i = 0;
    }

}
```

Java Program Structure

- The main unit of organization in Java is a class. Here is a simple example:

Source code for this class is in the file **`FirstExample.java`**

Class name (**`FirstExample`**) must match the filename minus the file extension.

```
public class FirstExample {

    public static void main(String[] args) {
        int i = 0;
    }

}
```

Java Program Structure

- The main unit of organization in Java is a class. Here is a simple example:

Source code for this class is in the file **`FirstExample.java`**

Class name (**`FirstExample`**) must match the filename minus the file extension.

This a method called **`main`**… **`main`** is a special method that determines what gets run when the program executes

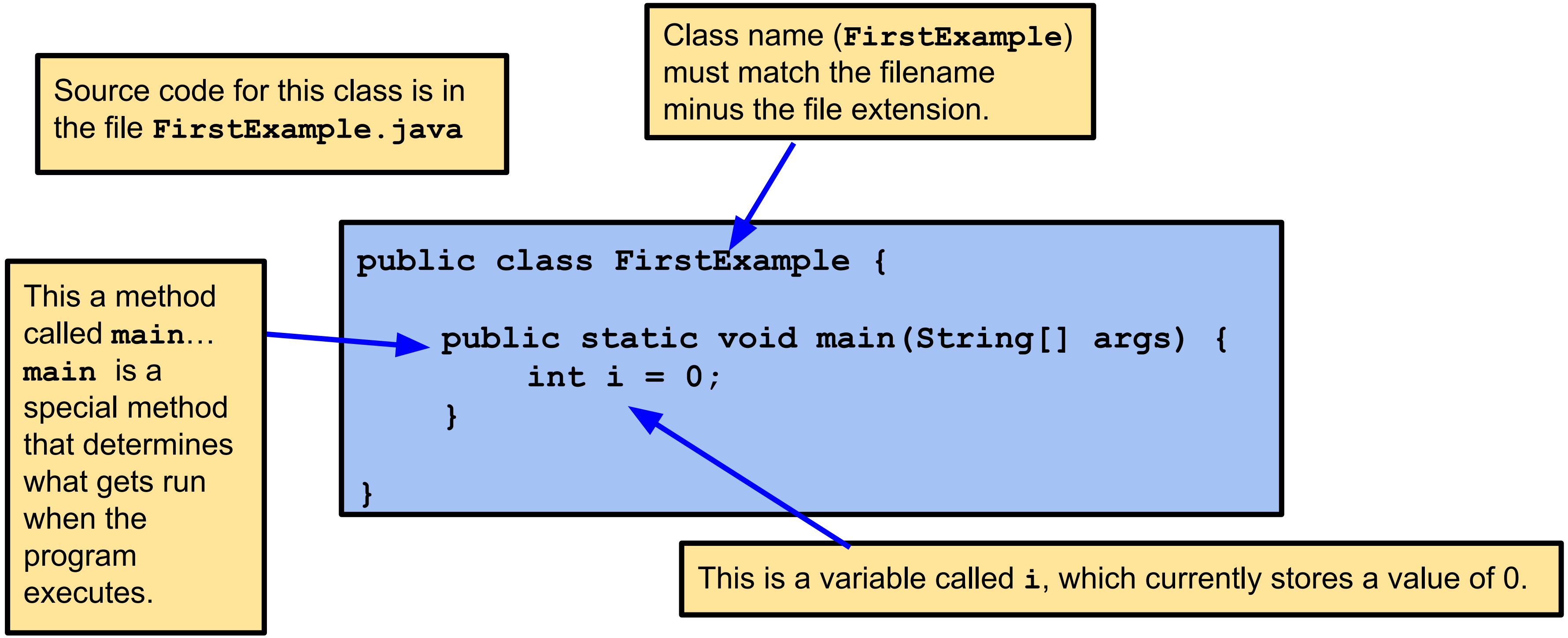
```
public class FirstExample {

    public static void main(String[] args) {
        int i = 0;
    }

}
```

Java Program Structure

- The main unit of organization in Java is a class. Here is a simple example:



Source code for this class is in the file **`FirstExample.java`**

Class name (**`FirstExample`**) must match the filename minus the file extension.

```
public class FirstExample {

    public static void main(String[] args) {
        int i = 0;
    }

}
```

This a method called **`main`**… **`main`** is a special method that determines what gets run when the program executes.

This is a variable called **`i`**, which currently stores a value of 0.

Java Program Structure

- The main unit of organization in Java is a class. Here is a simple example:

Source code for this class is in the file **`FirstExample.java`**

Class name (**`FirstExample`**) must match the filename minus the file extension.

All java statements end with a semicolon.

```
public class FirstExample {

    public static void main(String[] args) {
        int i = 0;
    }

}
```

This a method called **`main`**… **`main`** is a special method that determines what gets run when the program executes.

This is a variable called **`i`**, which currently stores a value of 0.

Java Variables: Declaring and Assigning Values

- In math, a variable is a representation for something that might change.
- In programming, it is thought of as a container
- This is what a java variable declaration looks like:

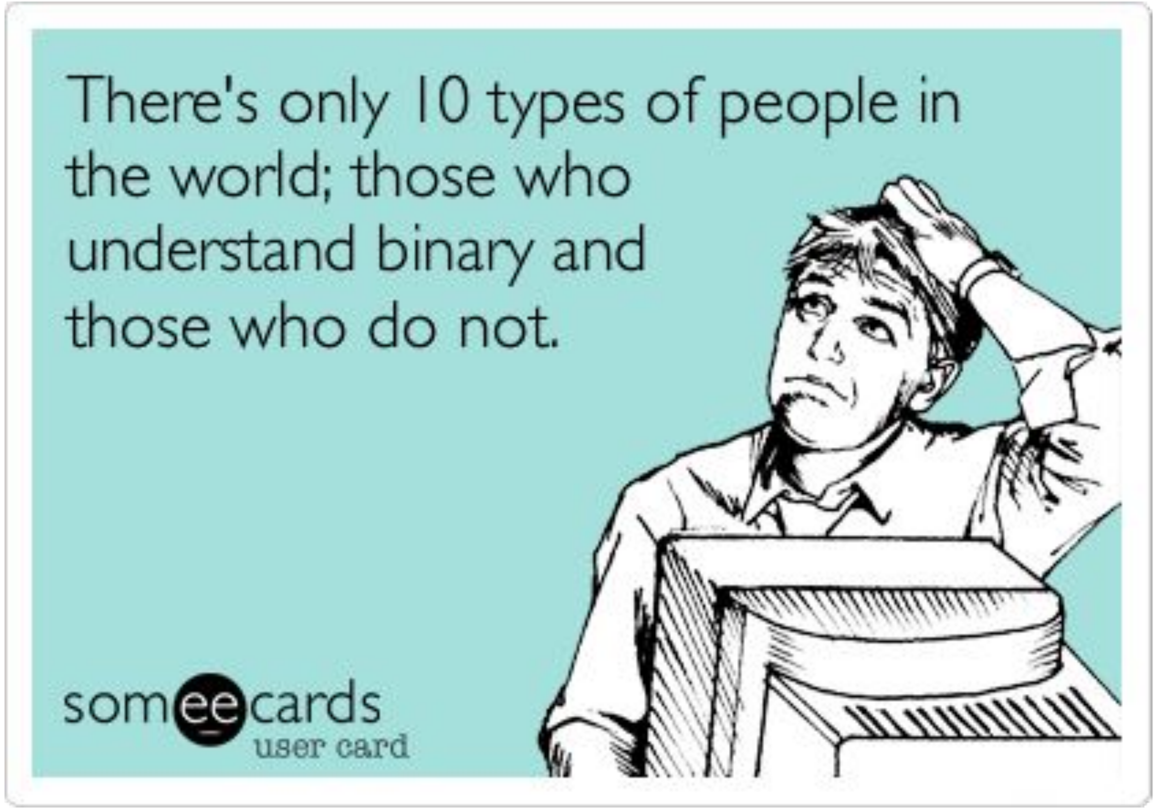
```
int i = 0;
```

Here, we have declared a variable called **`i`** of type integer, we have also given it an initial value of zero. Assigning values is accomplished using the equal sign (=).

- Consider this:

```
int i = 0;
i = 1;
```

Here, we have declared a variable called **`i`** of type integer, we gave it an initial value of zero, but then changed the value of the variable to 1.


There's only 10 types of people in the world; those who understand binary and those who do not.
someecards
user card

Example:

What value does the number 10 represent?

Example:

What value does the number 10 represent?

The value we know is base ten (**10**)

Base ten means that the first digit represents the number of '**10**'s and the second represents the number of '**1**'s

Example:

What value does the number 10 represent?

The value we know is base ten (**10**)

Base ten means that the first digit represents the number of '**10**'s and the second represents the number of '**1**'s

Base TWO mean that the first digit represents the number of '**2**'s and the second represents the number of '**1**'s

So in base two: **10** = (1 x 2) + (0 x 1) = **2**

Base two is known as **Binary**

Example:

What value does the number 10 represent?

The value we know is base ten (**10**)

Base ten means that the first digit represents the number of '**10**'s and the second represents the number of '**1**'s

Base TWO mean that the first digit represents the number of '**2**'s and the second represents the number of '**1**'s

So in base two: **10** = (1 x 2) + (0 x 1) = **2**

Base two is known as **<u>Binary</u>**

You must tell Java what the **TYPE of variable** you are using is just as we need to know the BASE in this example.

Java Variables: Data Types

Data Type	Description	Example
int	Integers (whole numbers)	int i = 1;
long	Long integers (whole numbers) that can hold larger numbers than int	long l = 5000000L; // Note that to declare a long you must explicitly state that the number is a long by appending the L.
double	Decimals	double x = 3.14;
float	Also decimals, but older format. Avoid, use doubles instead.	float x = 3.14f; // Note that to declare a float you must explicitly state that the number is a float by appending the f.
char	One character. **Note the single quotes**.	char myChar = ‘a’;
String	A bunch or characters. **Note the double quotes**.	String myName = “Horatio”; String mySentence = “This is a beautiful day.”;

boolean	true or false	boolean isItTurnedOn = true;
		boolean paidBills = false;

Java Variables: Strings

A String represents a sequence of zero or more Unicode characters.

- Declaring a String

```
String myString = "My String Data";
```

- Escape Characters:
    - Like `\n` and `\t`.
    - Needed because you can't really type a tab or return in a string directly.

Java Variables: Rules & Conventions

Conventions:

- Ideally, variables should be named using “Camel Case.”
  - Examples of variable names:
    - playerOneScore
    - cityTemperature
    - shirtSize
- Ideally, variables should never start with an upper case.
- Variable names should be descriptive and of reasonable length.

Rules:

- Variables can begin with an underscore (_), a dollar sign ($), or a letter.
- Subsequent characters can be letters, numbers, or underscores.
- Variable names cannot be the same as java keywords. https://www.w3schools.in/java-tutorial/keywords/

Let's
Try It!!!

Introducing: Expressions

Introducing Expressions...

An ***expression*** is a statement of code which can be evaluated to produce a result.

We use the result and often assign it to another variable or as the input to another expression.

```
int sum = 5 + 6;
```

5 + 6 is an expression. It evaluates to 11 and then is stored in the variable sum.

Working with Numbers: Basic Operators

- Math operators can be used to perform basic arithmetic between two numeric variables.
  - From the previous slides: variables of type int, float, and double are examples.
- These are the basic operators:
  - + (addition)
  - - (subtraction)
  - * (multiplication)
  - / (division)
  - % (modulo, aka remainder)
- The basic operators can be combined with the assignment operator to store result calculations, for example: int i = 4 + 6;
- Operator/assignment operators:
  - i += 5; i -= 5; i *= 5; i /= 5; i %= 5;

Working with Numbers: Order of Operations

For now, order of operations is the same as normal arithmetic:

- **P**lease **E**xcuse **M**y **D**ear **A**unt **S**ally (this will get more complex as we introduce more Java concepts)

    - Anything inside **p**arentheses first.
    - **E**xponents
    - **M**ultiplication
    - **D**ivision
    - **A**ddition
    - **S**ubtraction

Note that even though the acronym suggests that multiplication outranks division, and addition outranks subtraction, multiplication and division have the same priority, likewise addition and subtraction have the same priority.

Working with Numbers: Order of Operations

For now, order of operations is the same as normal arithmetic:

- **P**lease **E**xcuse **M**y **D**ear **A**unt **S**ally (this will get more complex as we introduce more Java concepts)
    - Anything inside **p**arentheses first.
    - **E**xponents
    - **M**ultiplication
    - **D**ivision
    - **A**ddition
    - **S**ubtraction

$\boxed{(4 + 6) * 5 = 50}$

$\boxed{4 + 6 * 5 = 34}$

Note that even though the acronym suggests that multiplication outranks division, and addition outranks subtraction, multiplication and division have the same priority, likewise addition and subtraction have the same priority.

Working with Numbers: Example 1

- Express the following English statement in Java: I paid for an item that cost $8.50 with a $10.00 bill, how much change would I get in return?

```
public class MyClass {

    public static void main(String[] args) {




    }
}
```

Working with Numbers: Example 1

- Express the following English statement in Java: I paid for an item that cost $8.50 with a $10.00 bill, how much change would I get in return?

```
public class MyClass {

    public static void main(String[] args) {

        double price = 8.50;



    }
}
```

Working with Numbers: Example 1

- Express the following English statement in Java: I paid for an item that cost $8.50 with a $10.00 bill, how much change would I get in return?

```
public class MyClass {

    public static void main(String[] args) {

        double price = 8.50;
        double payment = 10.00;


    }
}
```

Working with Numbers: Example 1

- Express the following English statement in Java: I paid for an item that cost $8.50 with a $10.00 bill, how much change would I get in return?

```
public class MyClass {

    public static void main(String[] args) {

        double price = 8.50;
        double payment = 10.00;
        double change = payment - price;

    }
}
```

Working with Numbers: Example 1

- Express the following English statement in Java: I paid for an item that cost $8.50 with a $10.00 bill, how much change would I get in return?

```
public class MyClass {

    public static void main(String[] args) {

        double price = 8.50;
        double payment = 10.00;
        double change = payment - price;
        System.out.println(change);
    }
}
```

Working with Numbers: Example 2

- Let’s try something a little bit more complex: We can convert degrees in Fahrenheit to Celsius using the following formula: (TF - 32) x (5/9) = TC. How much is 98.6 degrees Fahrenheit in Celsius?

```
public class MyClass {

    public static void main(String[] args) {





    }
}
```

Working with Numbers: Example 2

- Let’s try something a little bit more complex: We can convert degrees in Fahrenheit to Celsius using the following formula: (TF - 32) x (5/9) = TC. How much is 98.6 degrees Fahrenheit in Celsius?

```
public class MyClass {

    public static void main(String[] args) {

        double tempInF = 98.6;



    }
}
```

Working with Numbers: Example 2

- Let’s try something a little bit more complex: We can convert degrees in Fahrenheit to Celsius using the following formula: (TF - 32) x (5/9) = TC. How much is 98.6 degrees Fahrenheit in Celsius?

```
public class MyClass {

    public static void main(String[] args) {

        double tempInF = 98.6;
        double tempInC = (tempInF -32.0) *
(5.0/9.0);

    }
}
```

Working with Numbers: Example 2

- Let’s try something a little bit more complex: We can convert degrees in Fahrenheit to Celsius using the following formula: (TF - 32) x (5/9) = TC. How much is 98.6 degrees Fahrenheit in Celsius?

```
public class MyClass {

    public static void main(String[] args) {

        double tempInF = 98.6;
        double tempInC = (tempInF -32.0) *
(5.0/9.0);
        System.out.println(tempInC);
    }
}
```

Working with Numbers: Type Conversion

ints, doubles and floats can be used together in the same statement, but Java will apply certain rules:

- Bytes and Shorts are automatically promoted to an int, when used together. Consider this code:

```
public class MyClass {

     public static void main(String[] args) {

          byte myByte = 4;
          byte myOtherByte = 1;

          // Wont'work, Java will complain!
          byte firstAttempt = myByte + myOtherByte;
     }
}
```

The result of **`myByte`** and **`myOtherByte`** is promoted to an **`int`**, it will no longer fit in **`firstAttempt`**, which is a byte

Working with Numbers: Type Conversion

- We can overcome this type incompatibility by doing a cast:

```
byte secondAttempt = (byte) (myByte + myOtherByte);
```

- We can also overcome type incompatibility in this code by making the variable secondAttempt an int:

```
int secondAttempt = myByte + myOtherByte;
```

Working with Numbers: Type Conversion

Two additional rules:

- A long with anything else will become a long.
- A double with anything else becomes a double.

```
public class MyClass {

     public static void main(String[] args) {
          long thisVariable = 4;
          int anotherVariable = 7;

          // This won't work:
          int someVariable1 = thisVariable + anotherVariable;

          // This is perfect:
          long someVariable2 = thisVariable + anotherVariable;
     }
}
```

Working with Numbers: Type Conversion

Remember this problem?

```
public class MyClass {

    public static void main(String[] args) {

        double tempInF = 98.6;
        double tempInC = (tempInF -32.0) * (5/9);
        System.out.println(tempInC);
    }
}
```

Note that instead of 5.0/9.0, it is now 5/9, and when run, the result is 0.0. Using the rules we just discussed, can you figure out why?

Combining Strings

The plus sign can also be used with Strings:

```
public class MyClass {

     public static void main(String[] args) {

          String firstName = "Bob";
          String lastName = "Ross";

          String combinedName = lastName + ", " + firstName;
          System.out.println(combinedName);
     }
}
```

The above code will print ***Ross, Bob***. This process is known as ***concatenation***.

Let's Try Using What We Learned...

DEBUGGER DEMO

Cheat Sheets & References

- IntelliJ IDEA Cheat Sheet:
  - https://resources.jetbrains.com/storage/products/intellij-idea/docs/IntelliJIDEA_ReferenceCard.pdf
- Java Data Type Conversion
  - https://drive.google.com/open?id=0Bz4DHj0l-C66MnV4cnRiUm1UYWc
- Java Operator Summary
  - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html