1.1 - Introduction

Computer science involves **problem-solving**, **hardware**, and **algorithms** that help people utilize computers and **incorporate multiple perspectives to address real-world problems** in contemporary life. As the study of computer science continues to evolve, the careful design of the AP Computer Science A course allows students to discover the power of computer science through **rewarding yet challenging** concepts.

1.2 - Coding Environment Setup

Install IntelliJ IDEA as we did in the class. Go this link if you get stuck.

1.3 - Why Programming? Why Java?

Origin of Programming

In the early 1840s, Charles Babbage proposed a machine called the Analytical Engine. It was only a proposal—no actual machine was built, but one inventive woman by the name of Ada Lovelace decided to write an article that provided detailed instructions on how to represent Bernoulli [ber-noo-li] numbers, a recursive equation based on number theory, on the Analytical Engine. This article is considered to be the very first computer program.

Since then the devices that can be programmed went from theoretical to physical, manual to automatic, analog to digital. With each evolutionary step, the way we program computers needed to evolve as well.

With the birth of mainframe computers, data processing required instructions to be sent to the machine to process and interpret the instructions from the programmer. This was then applied to data to organize and analyze the data. Instructions were entered through a keyboard, but without the benefit of a monitor, so everything was done through printouts on paper or storing data in the form of punched holes on cards that were stored in stacks. If you look carefully at text encodings and at some programming languages, you'll see things like "carriage return" or "print" that are carryovers from those printer days from decades ago.

Why Java?

Write once, run anywhere. Java is often described as "platform-free" or "platform-independent" because of its ability to run on any operating system or hardware platform without modification. This feature is achieved through the use of the Java Virtual Machine (JVM).

How Java Works?

The problem with compiled languages

A compiled language takes instructions written by a human and sends that code to something called a compiler. A compiler takes the program instructions and converts it to binary bytecode or native code for the hardware and creates a program called an executable. When you convert your code to an executable program, the compiler reads the code and creates a binary file that is specific to a processor architecture.

So, if I used a language like C or C++ to compile a program on my 32-bit Windows PC, that program can’t run on a Mac, nor can it run on a 64-bit Windows PC natively without added technologies to bridge the gap. It can only run on a 32-bit Windows PC. Every processor and operating system is expecting a program to meet extremely specific requirements. The way the program accesses the hardware is vastly different between Intel processors, ARM-based processors (like those found on phones and tablets), and other processor types like RISC architectures. (Older Macs used this type of architecture using chips called PowerPC chips like the G5 or G4.)

Even 32-bit Intel processors and 64-bit Intel processors operate completely differently. So 64-bit programs can’t run on 32-bit computers, and 32-bit programs can’t run natively on 64-bit computers without additional software to make it compatible. When you expand to other types of computers like servers, the number of processor types and architectures can get even more complex.

Web and application servers can use vastly different architecture types like Sparc, Xeon, Itanium, and other types over the last couple of decades.

So, the folks at Sun Microsystems, when they designed Java, wanted to get past this limitation, so they created an innovative way to get around it.

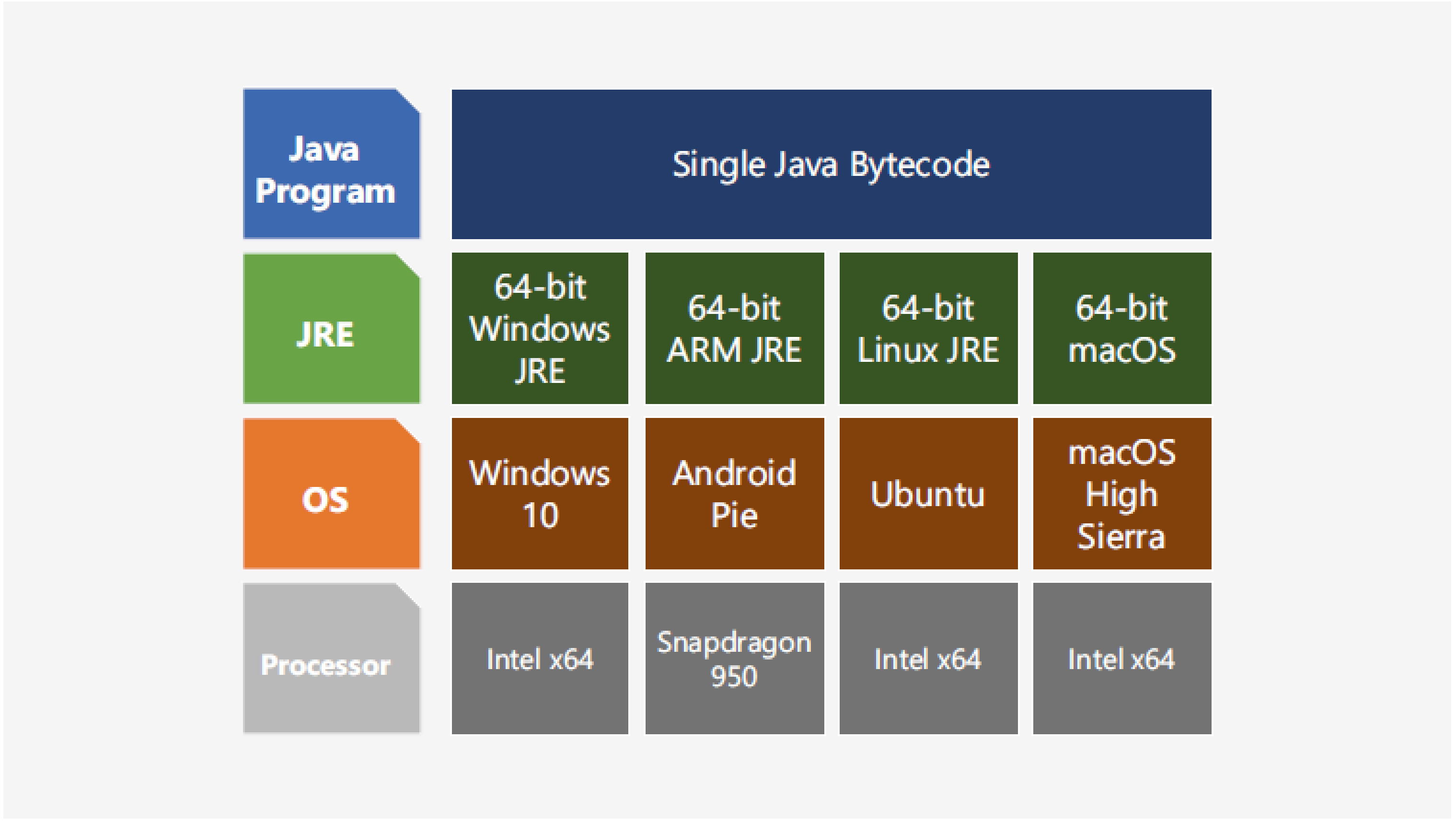
JVM and JRE

When you run a Java program, you aren’t just running the program, you are running a few things at the same time.

For multiple platforms, Oracle has created a set of technologies called the Java Runtime Environment and the Java Virtual Machine. These are built so that the specific architectures of the hardware including Intel, ARM, 64-bit, 32-bit, and more can run Java programs.

These runtimes are installed on the system and stay resident, waiting for a Java program to be executed.

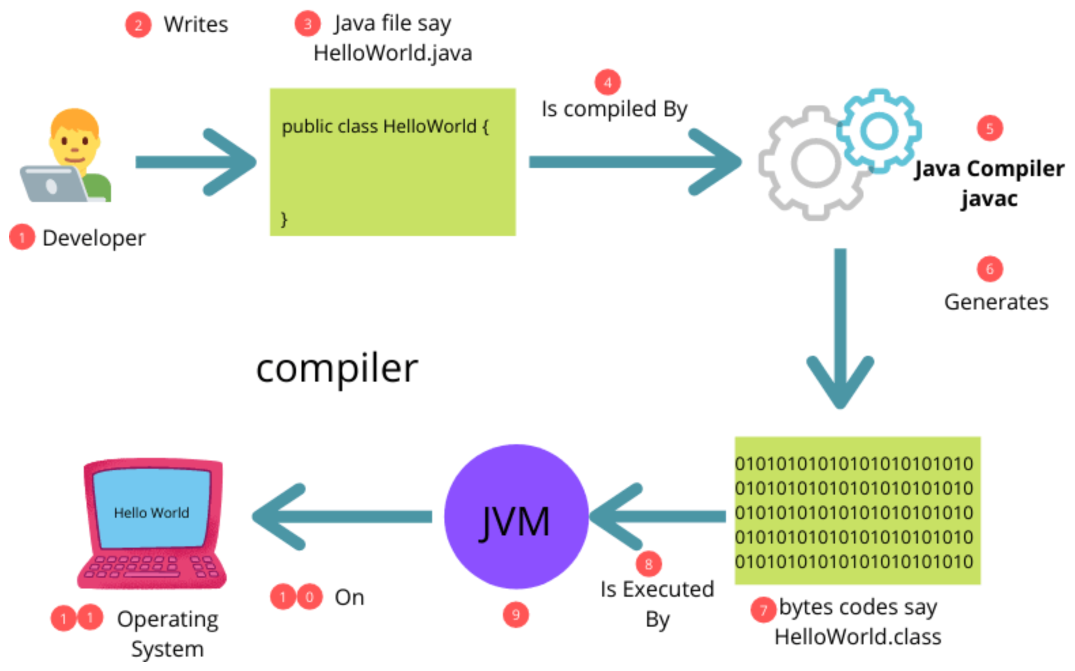
When a Java program is called to run, the Java Runtime Environment, or JRE, kicks in and opens the compiled program. The runtime environment then captures the instructions of the program and converts them to native executions on the computer.



This is what has made Java so universal. Wherever Oracle has created a JRE, you can run Java programs.

Compiling Java Bytecode

When you create your Java program, there are a few things in play when you create your code, called source code or source.



First, when you create your program, you are writing a plain text file. There are no formatting, special characters, graphics, or anything other than letters and numbers. When you save this file to your computer, it is a *.java* file, meaning that the file extension is .java. When you are finished coding your program, you take the file and send it to the compiler. The compiler takes the instructions you created and converts them to Java Bytecode, which the JRE can read and understand. The Java Bytecode (sometimes referred to as JBC) is a concise and compressed file that isn’t human readable but is designed to be understood and executed quickly by the JRE. The JRE then takes those bytecode instructions and sends them to the native hardware to execute the program on the machine. When you compile your program, you are creating these Java Bytecode files, called *.class* files, and they will run in the JRE against the native hardware of the architecture.

Binary System in Computing

In computing, all data, including numbers, text, images, and instructions, is represented in binary, which uses only the digits 0 and 1. For example, the letter 'A' is represented in binary as 01000001 in the ASCII encoding. Computers perform logical operations such as AND, OR, NOT, and XOR on binary data, which are essential for decision-making and processing. Memory and storage devices use binary to store data, with each cell holding a binary value of 0 or 1. Machine code, the lowest level of code executed by the CPU, is written in binary, with each instruction being a sequence of binary digits that the CPU interprets and executes.

First Java Program

Every program in Java is written as a class. Java is an object-oriented language and we'll learn more about classes and objects in Unit 2. Inside the class, there can be a main method that starts the program. When you ask the Java run-time to run a class, it will always start execution in the main method.

In [2]:
```
public class MyClass
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}

MyClass.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
Hello World!
```

- Java has two different methods to print output to the screen:
    - System.out.println(value) : prints the value followed by a new line (ln).
    - System.out.print(value) : prints the value without advancing to the next line.

`System.out.println("Hello World!");` prints out the characters between the first " and the second " followed by a new line. The "Hello World!" is called a string literal, and it can have zero to many characters enclosed in starting and ending double quotes.

- Special words —also called keywords— such as `public`, `class`, and `if` must be in lowercase, but class names such as `System` and `String` are

capitalized.

- Lines in a Java program that express a complete action such as assigning a value to a variable must end with a semicolon (;). Such a line is called a **statement**. ;s in Java are like .s in sentences. There is an exception, however, if the line starts a construct like an if statement, there is no semicolon before the opening { nor one after the closing }.

1.4 Variables and Data Types

- To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.
- A type is a set of values (a domain) and a set of operations on them. For example you can add 2 with 5 but can't add 2 and "slgjldsgjs". There are two types of variables in Java: **primitive** variables that hold primitive types and **object or reference** variables that hold a reference to an object of a class. A reference is a way to find the object (like a UPS tracking number helps you find your package). The primitive types on the Advanced Placement Computer Science A exam are:
    - `int` which can represent integers, i.e. numbers with no fractional part such as 3, 0, -76, and 20393.
    - `double` which can represent non-integer numbers like 6.3 -0.9, and 60293.93032. Computer people call these “floating point” numbers because the decimal point “floats” relative to the magnitude of the number, similar to the way it does in scientific notation like $ 6.5 \times 10^8 $
        - The name double comes from the fact that doubles are represented using 64 bits, double the 32 bits used for the type float which used to be the normal size floating point number when most computers did math in units of 32-bits. (float is rarely used these days and is not part of the AP curriculum.)
    - `boolean` which can represent only two values: true and false. (The data type is named for George Boole, a 19th century English mathematician who invented Boolean algebra, a system for dealing with statements made up of only true and false values.)

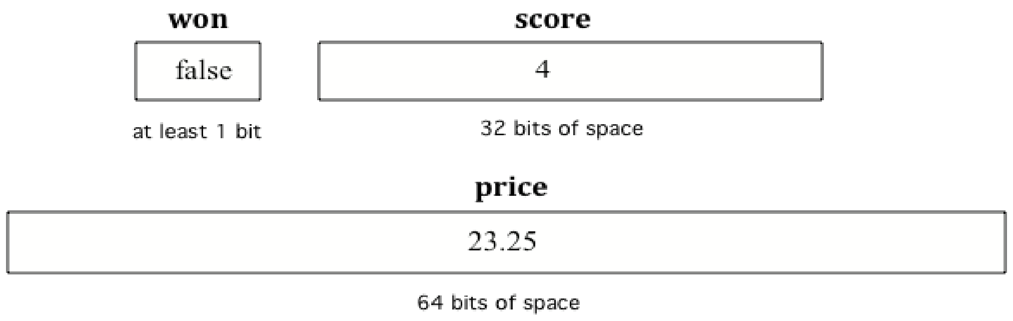
  This primitive data types used in this course define the set of operations for numbers and Boolean values.

    - `String` is one of the object types on the exam and is the name of a class in Java. A String is written in a Java program as a sequence of characters enclosed in a pair of double quotes - like "Hello".

How to declare variables in Java?

- To create a variable, you must tell Java its data type and its name. Creating a variable is also called declaring a variable. The type is a keyword like int, double, or boolean, but you get to make up the name for the variable. When you create a primitive variable Java will set aside enough bits in memory for that primitive type and associate that memory location with the name that you used.

- Computers store all values using bits (binary digits). A bit can represent two values and we usually say that the value of a bit is either 0 or 1. When you declare a variable, you have to tell Java the type of the variable because Java needs to know how many bits to use and how to represent the value. The 3 different primitive types all require different number of bits. An integer gets 32 bits of memory, a double gets 64 bits of memory and a boolean could be represented by just one bit.



How to name variables?

- A variable name should start with an alphabetic character (like a, b, c, etc.) and can include letters, numbers, and underscores _. It must be all one word with no spaces.
- You can't use any of the keywords or reserved words as variable names in Java (for, if, class, static, int, double, etc). For a complete list of keywords and reserved words, see https://docs.oracle.com/javase/specs/jls/se14/html/jls-3.html#jls-3.9.
- Use meaningful variable names!
- Start variable names with a lower case letter and use camelCase.
- Variable names are case-sensitive and spelling sensitive! Each use of the variable in the code must match the variable name in the declaration exactly.
- Never put variables inside quotes (" ")!

In [19]:
```java
public class Variables
{
    public static void main(String[] args)
    {
        int score;
        score = 0;
        System.out.print("The score is ");
        System.out.println(score);

        double price = 23.25;
        System.out.println("The price is " + price);

        boolean won = false;
        System.out.println(won);
        won = true;
        System.out.println(won);

        String name = "Jose";
        System.out.println("Hi " + name);
    }
}

Variables.main(null) /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
The score is 0
The price is 23.25
false
true
Hi Jose
```

- The memory associated with a variable of a primitive type holds the actual primitive value.
- The memory associated with a variable of a reference type holds a reference (or address) to the actual object in the heap memory. This means the variable itself does not contain the object's data directly but rather a pointer to the location in memory where the object is stored.

- When a variable is declared final, its value cannot be changed once it is initialized.

In [15]:

```
public class Variables2
{
    public static void main(String[] args)
    {
        final int x;
        x = 4;
        x = 10; // this gives an error because x is final.
    }
}
```

```
|           x = 10; // this gives an error because x is final.
variable x might already have been assigned
```

1.5 Expressions and Assignment Statements

- Assignment statements initialize or change the value stored in a variable using the assignment operator `=`. An assignment statement always has a single variable on the left hand side. The value of the expression (which can contain math operators and other variables) on the right of the `=` sign is stored in the variable on the left.

In [27]:

```
public class Assignment
{
     public static void main(String[] args)
    {
        int minute = 60;
        int hour = 60 * minute;
        int day = 24*hour;
        int year = 365*day + 6*hour;
        System.out.println("There are "+year+" seconds in a year.");

        double myHeight = 1.74;
        double friendHeight = 1.68;
        double lebronJamesHeight = 2.06;
        boolean meOrLebron = lebronJamesHeight > myHeight;
        String heights = "Me +"+" My Friend +"+" Lebron James";
        double total = myHeight + friendHeight+lebronJamesHeight;
        System.out.println(heights+" make a total of "+total+" meters.");
    }
}

Assignment.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
There are 31557600 seconds in a year.
Me + My Friend + Lebron James make a total of 5.48 meters.
```

In [30]:
```
public class Assignment2
{
    public static void main(String[] args)
    {
        int x = 3;
        int y = 6;
        int z = y/x;
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
        x = y;
        y = 23;
        System.out.println(x);
        System.out.println(y);
        System.out.println(z);
    }
}

Assignment2.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
3
6
2
6
23
2
```

- As we see above, we can set one variable’s value to a copy of the value of another variable like y = x.
- See that the value of z didn't change after we changed x and y since we calculated z using initial values of x and y.

Changing a variable using itself

In [76]:
```
public class Assignment3{
    public static void main(String[] args){
        int score = 3;
        score = score + 1;
        System.out.println(score);
        double score2 = 99.3;
        score2 = score2 + 5;
        System.out.println(score2);
        int score3 = 24;
        score3 = score3 * 12;
        System.out.println(score3);
    }
}

Assignment3.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
4
104.3
288
```

Input with Variables

In [49]:

```
import java.util.Scanner;

public class Main
{
    public static void main(String[] args)
    {
        System.out.println("Please type in a name in the input box below.");
        Scanner scan = new Scanner(System.in); //not tested on AP Exam.
        String name = scan.nextLine();
        System.out.println("Hello " + name);
        scan.close();
     }
 }

Main.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
Please type in a name in the input box below.
Hello Fatih
```

Expressions in Java

- In Java you can do these arithmetic operations:
  - addition using `+`
  - subtraction using `-`
  - multiplication using `*`
  - division using `/`
  - exponentiation using `Math.pow()` (not `^`)
  - taking remainder using `%`
  - more using `Math` class
  - the order of operations is `(), * , / , % , + , -` (remember PEMDAS?)
- Arithmetic expressions can be of type `int` or `double`. An arithmetic expression consisting only of `int` values will evaluate to an `int` value. An arithmetic expression that uses at least one `double` value will evaluate to a double value.
- Operators can be used to construct compound expressions.
- `+` is also used for string concatenation. More on that later.
- You can do this relational operations:
  - use `==` to check if two values are equal
  - use `!=` to check if two values are **not** equal
  - use `<,>,<=,>=` for the same mathematical purposes
- Note that using `==` and `!=` with `double` values can produce surprising results. Because double values are only an approximation of the real numbers even things that should be mathematically equivalent might not be represented by the exactly same `double` value and thus will not be `==`.

In [57]:
```java
public class Operations
{
    public static void main(String[] args)
    {
        System.out.println(2 + 3);
        System.out.println(2 - 3);
        System.out.println(2 * 3);
        System.out.println(2 / 3);
        // == is to test while = is to assign
        System.out.println(2 == 3);
        System.out.println(2 != 3);
    }
}
```

```
Operations.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
5
-1
6
0
false
true
```

- See that it gives 2/3 as 0. When you do any operation using just integers, Java assumes you want to get an integer result. So it just **truncates** the part after the decimal point. This is called **truncating division**. If you use doubles instead: 👇

In [59]:

```
public class Operations
{
    public static void main(String[] args)
    {
        System.out.println(2 + 3.0);
        System.out.println(2 - 3.0);
        System.out.println(2 * 3.0);
        System.out.println(2 / 3.0);
        // == is to test while = is to assign
        System.out.println(2 == 3);
        System.out.println(2 != 3);
        System.out.println(2 == 2.0); // see that it gives true
    }
}

Operations.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
5.0
-1.0
6.0
0.6666666666666666
false
true
true
```

- An attempt to divide an integer by zero will result in an `ArithmeticException` to occur. 👇

In [67]:
```
public class Expressions{
    public static void main(String[] args){
        System.out.println(2/0);
    }
}

Expressions.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
---------------------------------------------------------------------------
java.lang.ArithmeticException: / by zero
        at Expressions.main(#77:3)
        at .(#78:1)
```

- Now let's look at remainder operator `%`

In [75]:
```
public class Expressions2{
    public static void main(String[] args){
        System.out.println(11 % 10);
        System.out.println(3 % 4);
        System.out.println(8 % 2);
        System.out.println(9 % 2);
        //if there are negatives
        System.out.println(10 % -3);
        System.out.println(-53 % 5);
        System.out.println(-42 % -8);
        System.out.println(-42 % -7);
    }
}

Expressions2.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
1
3
0
1
1
-3
-2
0
```

- You might find the taking remainder with negative operands a bit of complicated. But recall that this is all about Euclidean Algorithm. That is, if $ a = b + qr $ where $0 < r < b$, `a % b` gives `r`.

1.5 - Compound Assignment Operators

- For any numeric variable `x`, `x++` is called post-increment. When we used this, the value of `x` is incremented by 1 **after** the current expression is evaluated. In contrast, `++x` is called pre-increment and here the value of `x` is incremented by 1 **before** the current expression is evaluated. Exactly same things go for `x--` and `--x` either.

In [79]:
```
public class Assignment4{
    public static void main(String[] args){
        int x = 5;
        int y = x++;
        System.out.println("x is: "+x);
        System.out.println("x increased by 1 but y is: "+y);
        int z = ++x;
        System.out.println("x is: "+x);
        System.out.println("x increased by 1 and z got that value: "+z);
    }
}

Assignment4.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
x is: 6
x increased by 1 but y is: 5
x is: 7
x increased by 1 and z got that value: 7
```

In [80]:
```java
public class Assignment5{
    public static void main(String[] args){
        int x = 5;
        int y = x--;
        System.out.println("x is: "+x);
        System.out.println("x decreased by 1 but y is: "+y);
        int z = --x;
        System.out.println("x is: "+x);
        System.out.println("x decreased by 1 and z got that value: "+z);
    }
}

Assignment5.main(null); /*This is just the code to see the output at the environment I create
                          the notebooks. Normally we will not do this in Java.*/
```

```
x is: 4
x decreased by 1 but y is: 5
x is: 3
x decreased by 1 and z got that value: 3
```

- **Pre-increment** and **pre-decrement** are **not tested** on AP Exam.

- Instead of `x = x + 3`, say, to increase `x` by 3, we can do `x += 3`. They are exactly the same. 👇

In [83]:
```java
public class CompoundAssignment{
    public static void main(String[] args){
        int x = 5;
        x += 100;
        System.out.println(x);
        x -= 20;
        System.out.println(x);
        x *= 10;
        System.out.println(x);
        x /= 12;
        System.out.println(x);
        x %= 4;
        System.out.println(x);
    }
}

CompoundAssignment.main(null); /*This is just the code to see the output at the environment I create
                    the notebooks. Normally we will not do this in Java.*/
```

```
105
85
850
70
2
```

THANK YOU!

You got this !
SHOW ME THOSE MUSCLES!!