JAVA MODULE 1

REQUIRED KNOWLEDGE TO SOLVE THE PROBLEMS

- Java Fundamentals
  - Variables
  - Arithmetic
  - Comments
- Selection Control Structures
  - if-statements
  - if-else statements
  - switch statements
- Repetition Control Structures
  - for loops
  - while loops
  - do-while loops

REVIEW OF REQUIRED KNOWLEDGE

In this section, let's do a review and see some code samples so you understand what you need to solve the problem.

JAVA FUNDAMENTALS

In order to do anything in a programming language like Java, you should be familiar with fundamental Java programming constructs, such as variables, arithmetic and comments.

VARIABLES

A **variable** is a named storage location to hold some kind of data. You can think of a variable as a box that holds something in it. This "box" exists in the computer's **memory**, often called **main memory**, or **random access memory** (**RAM**.)

A variable has a **data type**, which is, not surprisingly, the type of data that the variable can hold. Some commonly used data types are as follows:

Data type	Name/Description	Example
int	Integer. The counting numbers and their opposites.	0, 45, -3, -2, -100, 57, 1680
float	Floating point number. Can hold real numbers. (32 bit)	0.5, 6.4576, -3.5, 3.14159, 623.78

double	Floating point number. Can hold small or very large real numbers. (64 bit)	Same as above, and can hold very large numbers – more precision
char	Character data. (16 bit Unicode characters)	'a', 'b', 'j', '4', '$'
String	Text string data	"John", "Sam", "The dog."
boolean	Boolean data.	`true` and `false` are the only two values that a Boolean variable can take

When you give a variable a value, it's called **assigning** the variable a value. When you assign a variable a value when it's declared, or whenever the first time it is given a value, this is called **initialization** because you're giving the variable an initial value. Assigning a value is done with the **assignment operator**, the equals sign: `=`.

Here's an example of initializing a variable and printout the value to the console:

```
public static void main(String[] args)
{
	int a = 165;
	System.out.println("The value is " + a);
}
```

ARITHMETIC

When you have a programming expression such as the following:

```
b + 2;
```

We call the + symbol an **operator** and the `b` and the `2`, the **operands**.

We can perform arithmetic on values and variables using arithmetic operators:

Operator	Name/Description	Example
+	Addition operator. Returns the sum of its two operands.	5 + 7; a = 2 + someNum;
-	Subtraction operator. Returns the difference between its two operands.	6 – 4; 17.3 – 4.56; someValue – 14.3;
*****	Multiplication operator. Returns the product of its two operands.	7 * 54; 8.5 * 3; someNumber * 2.75;

/	Division operator. Returns the quotient of a division operation.	6 / 7; 6.45 / 2; 17.8 / someValue;
%	Modulus operator. Returns the remainder (residue) of a division operation.	7 % 4; 5 % 2;

The above operators should be straightforward, except for possible the **modulus operator**. This returns the remainder of a division operation. For example:

7 % 3 = 1 //this is because 7 / 3 = 2 with *a remainder of 1*
4 % 2 = 0 //this is because 4 / 2 = 2 with *a remainder of 0*
15 % 12 = 3 //this is because 15 /12 = 1 with a remainder of 3

Note that when saying these operations allowed, you say "7 mod 3" or "7 modulus 3" or "7 modulo 3", for example.

COMMENTS

Comments are an important part of programming, even though the compiler ignores them. They are a fundamental part of good software engineering. They allow you to write words and sentences in a **natural language** (such as English, as opposed to a **programming language** like Java or C++.) Comments serve to help remind yourself (or others) and to document the code you are writing so that it is more clear what a particular set of instructions is doing.

Single line comments:

```
//  this is a comment about something
```

Multi-line comments:

```
/*   Here is a multiple line comment
We can continue this until the star (*) and
forward slash (/) is encountered.
*/
```

SELECTION CONTROL STRUCTURES

Selection control structures, also called **conditional control structures** are used for the purpose of allowing your program to make decisions based upon a particular condition.

There are three selection control structures in Java:

- `if` single selection control statement
- `if-else` double selection control statement
- `switch` multiple selection control statement

The `if` statement is concerned with executing a block of code if a particular condition is true, or not executing that particular block of code if the particular condition is false.

```
import java.util.Scanner;

public class Test1 {

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

		Scanner keyboard = new Scanner(System.in);

		System.out.println("Please enter your age");
		age = keyboard.nextInt();

		if(age >= 18)
		{
			System.out.println("Congratulations!  You can vote!");
		}

		System.out.println("Done with program!");
	}
}
```

The `if-else` statement is concerned with executing a block of code if a particular condition is true, or executing *a different* block of code if that particular condition is false. Also, it should be noted that if-else statements can be cascaded, in which we have a formation of if-else-if… statements.

```
import java.util.Scanner;


public class Test1 {

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

		Scanner keyboard = new Scanner(System.in);

		System.out.println("Please enter your age");
```

```
                age = keyboard.nextInt();

                if(age >= 18)
                {
                        System.out.println("Congratulations!  You can vote!");
                }
                else
                {
                        System.out.println("You cannot vote, sorry!");
                }

                System.out.println("Done with program!");
        }
}
```

The `switch` statement is designed for multiple scenarios based on different conditions.

Let's say we tell the user to only enter 18-21 as values (for the sake of simplicity):

```
import java.util.Scanner;


public class Test1 {

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

                Scanner keyboard = new Scanner(System.in);

                System.out.println("Please enter your age (between 18 and 21)");
                age = keyboard.nextInt();

                switch(age)
                {
                        case 21:
                                System.out.println("You can drink AND vote legally!");
                                break;
                        case 20:
                        case 19:
                        case 18:
                                System.out.println("You can vote legally!");
                                break;
                        default:
                                System.out.println("You cannot drink or vote legally!");
                                break;
                }

                System.out.println("Done with program!");
        }
}
```

Even though we ask the user nicely not to enter values outside the range, if they do enter a value below 18, it will respond with "You cannot drink or vote legally!" Also, there is a logic bug in the program, technically. See if you can figure out what it is. (Hint: What happens if the user enters 22?)

REPETITION CONTROL STRUCTURES

Whereas selection control structures allow the program to perform different actions based upon selection between options, or among multiple options, **repetition control structures** (also called **iterative control structures** or sometimes just **loops**) allow the program to repeat a particular block of code potentially multiple times.

We have three repetition control structures in Java:

- `while`
- `do-while`
- `for`

The `while` loop allows a user to use a condition (much like with an `if` statement), called the **loop continuation condition**, to determine whether the block of code continues to execute.

```
public static void main(String[] args)
{
	int count = 0;

	while(count < 10)
	{
		System.out.println("Counter is " + count);
		count++;
	}
}
```

The `do-while` loop is identical to the `while` loop except that it is post-test. This means that it is guaranteed to execute the block of attached code at least once, and perform the loop continuation test *after* the loop has executed.

Notice in the example above that even though the condition would be false immediately (the counter variable is initialized to 10), that the code will execute once, because the loop is post-test.

```
public static void main(String[] args)
	{
		int count = 10;

		do
		{
			System.out.println("Counter is " + count);
			count++;
		}
		while(count < 10);
	}
```

The `for` loop is designed with a place for initializing a counting variable, the loop continuation condition, and a place to increment (or decrement, or otherwise modify) the continuation variable. For loops have the tendency to be more compact than `while` and `do-while`. Like the `while` loop, the `for` loop is pre-test.

```
public static void main(String[] args)
{
	for(int i = 0; i < 10; i++)
	{
		System.out.println("Counter is " + i);
	}
}
```

THE PROBLEMS

1. Write a program that allows the user to enter a series of non-zero integers. If the user enters 0, it will exit the loop and display the largest and smallest numbers of the numbers that were entered.

2. Write a program that generates a triangle made of asterisks (*) starting at one asterisk at the top, and whatever the user enters as `numStars` at the bottom. So if the user enters 6, then the triangle will look like this:

```
*
**
***
****
*****
******
```

3. Write a program that takes 5 grades (0 to 100) on projects submitted by a student. Once the user has entered these grades, you display the percentage of the student and then tell them what letter grade they received.

The grades are as follows:

A	90 – 100
B	80 – 89
C	70 – 79
D	60 – 69
F	Below 60