CS3300 Java Project Exercises

1) Introductory “hello world” program

Objective: User must write a program that simply prints “Hello World”.
Intended output: The string “Hello world”.

Hello! Welcome to your first exercise!
Here, we’ll cover the basics of a Java program, and even allow you to write some code. Let’s get started! Go ahead and examine the code below.

// This line is called a comment, and begins with “//”. A comment is purely for the person reading or writing the code, and typically explains a certain part of your code and what it does.

//This is our class. Don’t worry too much about what a class is right now. Just know that this is where the majority of your code needs to run, and it all needs to be contained within a pair of curly bracers, ie: { }.

```
Public void playerMain() {



//Here, the system outputs “Hello World!”. When writing output, it follows the format of
//System.out.println(“Your text here!”);
//Your text must be contained within double quotes, and this statement must end with a semicolon to indicate the end of this portion of code.

    System.out.println("Hello, World!");

//This backet closes our class.
}
```

Prompt: Go ahead and try this out for yourself! Write a program that prints “Hello World!”.

2) Variables

Objective: User must write a program that calculates the area of a circle with a diameter of 5.
Intended output: 9.42

The user must create a program that accepts some basic information (name, age, location) and stores it in pre-named variables. The program will feature pre-written print statements to output the values of the variables.
Intended output: Three variables that are not null.

```
Public void playerMain() {


//Let's talk about variables. A variable in programming is essentially just a piece of data that's associated with another name. Let's look at some examples!



//You'll see below that every variable declaration follows the same structure:

// [Data type] [Variable name] = [Value of the variable];

//A data type is simply a specific kind of information. For example, a whole number like 5 is an integer, and 'c' is a character in the alphabet. They are different types of information, and thus diffferent data types.


//For the sake of simplicity, you could look at a string as a sentence.

String sentence = "Here's a sentence stored inside a variable!";

//The "int" data type stores positive or negative whole numbers, as well as zero.

Int number_of_cows = 5;


//Float variables store decimal values. If you enter a whole number, such as 4, the variable will store 4.0. You can enter an integer into a float variable, but not the other way around! Similarly to ints, these values can be zero and negative as well.
```

Float hours_spent_mining = 3.2;

//A character stores a single character value, enclosed in single quotes. They do not necessarily have to be letters – they can store numbers (1, 7, 0), special characters (%, ?), as well as lowercase and uppercase letters. The important thing is that the value inside the char data type is one character long.

Char steve_first_initial = 'S';

//Finally, we have booleans. A boolean stores a value of true or false. If you for example only want a door to open if the player is holding some object, then you could have a boolean value that is set to true when they have it, and false when they do not. The game would only open the door if that boolean value was true.

Boolean ender_dragon_slain = false;

}

Prompt: Go ahead and try this out for yourself! Write a program that prints the circumference of a circle with radius 5.
Reminder: Circumference = 3.14 * 5

3) More on variables and arithmetic

Objective: User must write a program that creates a variable with a value of 5, prints it, and then increments it by 1, and prints it again.
Intended output: 56

```
Public void playerMain() {



//Let's say you have a variable, but you want to change its value. Well, good news: you can!


//Here, we're creating a variable "a" with a value of 5.
Int a = 5;
A = 7;
//Since we are not creating the variable "a", and only altering it, we do not need to include the data type. We simply set a equal to a new value.

//Let's look at a neat trick for incrementing a value. If you wanted to increase the value of a by 1, you could type:
// a = a +1;
//Here, we are setting a equal to the value of itself plus one, replacing the initial value. If a was 7 before, it is now 8. You can also write "a = a +1" as:
//a++;
//If you wanted to de-increment a, in other words, "a = a –1", you could write:
//a--;

}
```

Prompt: Go ahead and try this out for yourself! Create an integer variable with a value of 5. Print that value, and then increment the variable by 1. You can do this however you want, but when you have done so, print the variable again. Your program should print two total values when it is run.

4) Arrays

Objective: User must write a program that creates an array with the string “My first array”, and then print the array value at index 0.
Intended output: My first array

```
Public void playerMain() {

    //Lets discuss arrays. An array is one way in Java to hold and organize multiple
    different variables of the same type. Let’s create some!


    //Here, we are creating an array that contains various colors.
    //”String[]” indicates that this is a string array, meaning that the variables inside the
    array are all strings, which is an important point: Arrays can contain multiple values,
    but they must be of the same data type. Note that the strings are also contained
    within double quotes.
    String[] colors = {“Blue”, “Red”, “Green”};

    //Here, we have an array of type int[], which contains integers.
    Int[] numbers = {5, 10, 15, 20};


    //Now that we’ve created some arrays, how do we access or modify their values?

    //Arrays have what are called “indices”, and each index corresponds to a value
    within the array. For example, in our integer array declared above,

    // Int[] numbers = {5, 10, 15, 20};
    //We have four total members of the array, and thus four indices. While it might
    seem intuitive to say that 5 is at index 1, it actually begins at index 0. If we wanted to
    access 5, we would write:
```

```
//System.out.println(numbers[0]);
//This accesses the numbers array at index 0, which holds the value 5.

//If we wanted to modify the value 10, we could write:

//numbers[1] = 16;

//This would access the numbers array at index 1 and replace the value with 16.

}
```

Prompt: Go ahead and try this out for yourself! Create a string array containing only one entry: “My first string”. Once you have created this array, go ahead and print it.

5) Arraylists

Objective: User must write a program that creates an empty String arraylist, and then print its length. They must then use the .add functionality and put “Steve” at index 0.
Intended output: 0Steve

Public void playerMain() {

//Lets talk about another data storage structure, similar to arrays, called arraylists.
//An arraylist is similar to arrays, however it has two big differences: 1) the size of an arraylist can be changed after it’s been created, and 2) we are required to import a class.

//Importing a class simply means you are importing some sort of functionality into your program that is not there by default. These functionalities are defined by Java, or by users. We’ll talk more about classes later, but just understand classes help you use things that are not available normally when you begin a program.
import java.util.ArrayList;

//Let’s look at an example.

//Here, we are creating our arraylist. Let’s go piece by piece:
//”ArrayList<String>” specifies that this is an arraylist containing strings.
//”colors” is the name of the arraylist.
//”new” indicates that we are creating a brand new arraylist.

```
ArrayList<String> colors = new ArrayList<String>();
//You may notice a lack of indices being specified as we modify our array. When we create our array and it's empty, it effectively has a length of zero. We can now add values and increase the length of our arraylist.
Colors.add("red");
//The arraylist cars now has the entry "red", and looks like this.
//["red"]
//It also now has a length of 1, instead of zero.
Colors.add("blue");
Colors.add("green");
//Our array should now look like this:
//["red", "blue", "green"]
//We can also still access our arraylist entries by index! The following code would ouput "blue".
System.out.println(colors[1]);

//Finally, you can check the size of your arraylist by using the colors.size() function, which returns the length of the arraylist.
System.out.println(colors.size());
}
```

Prompt: Go ahead and try this out for yourself! Create an empty string arraylist, and then print its size. Once you've done that, use .add to place "Steve" inside the arraylist, and then print his name!

6) Input

Objective: User must write a program that accepts three user inputs and then outputs them. The inputs should be integers checking the number of chickens, cows, and pigs. The integers must add up to 15.
Intended output: Either the three integers, or 15.

Public void playerMain() {

// We've talked about output in the first lesson, let's talk about input! Specifically, take input from the user.

//Let's revisit a variable from the previous lesson.

String sentence = "Here's a sentence stored inside a variable!";

//What if we wanted to output the information contained in this variable? Well, you can!

System.out.println(sentence);

//This will print out the data contained in the sentence variable. Since we're passing a variable to System.out.println(), we don't need to put it in quotes. The output of this program should be:
// Here's a sentence stored in a variable!

//We didn't cover this in the previous lesson, but it's good for situations like this where we'll be taking user input.

Float hours_spent_mining;

// You will probably recognize this from our previous lesson, but you might notice it's missing some information at the end! What we're doing is called "initializing" the variable. Here, we essentially create a variable with a name, but it doesn't actually contain anything. It follows the format

//[Data type] [Data name];

//Well, what if we want to output a combination of text *and* variables in our program? Here's how!

System.out.println("I have spent " + hours_spent_mining + " hours spent mining!");

//This allows us to concatenate our text and variables. We utilize a total of two spaces and one plus sign (+) between variables and text to combine the two.

}

Prompt: Go ahead and try this out for yourself! Imagine you're designing a barn for your animals, and only have enough space for 15 animals (pigs, cows, and chickens). Taking user input, store the number of each animal in a variable (number_of_cows, number_of_chickens, number_of_pigs), and ensure they add up to 15. Finally, add the total number of each animal together, and print it.

7) Arithmetic

Objective: User must write a program that performs a variety of calculations specified by the prompt.
Intended output: 0

```
Public void playerMain() {

//Let's talk a little bit about math in programming. Don't panic! Not only is this section brief, but you can write programs that do math for you.
//We'll cover how to implement a couple of operators.

//Below, we have some basic math implemented. As you can see, it's rather intuitive if you understood the previous sections about variables. Operations typically follow the format:
Int addition_result = 3 + 5;
Float subtraction_result = 4.3 - 1.1;
Int multiplication_result = 12 * 15;
//Be careful! Attempting to divide by zero will throw an error in your program and could lead to some serious problems.
Float division_result = 100 / 5;

//Let's cover some operations that might not be so intuitive.
// % is the Modulo operator, and basically calculates the remainer of a division problem.
Int modulo_result = 25 % 4;
//Here, 4 goes into 25 eight times, and has a remainder of one. So, the result of 25 % 4 would be 1.

//You're likely familiar with exponents, but not written the way it is here. If your operation is x^y, then in Java, it would be written at x**y.
Int exponent_result = 2**4;
//The output here would be 16, since 2^4 = 16.
```

//You can also combine operators. In these cases, ensure you utilize parentheses to reinforce order of operations. Here's an example.

```
Float parentheses_are_important = (3 * 13.2) + (21 / 3) - 2;
```

}

Prompt: Go ahead and try this out for yourself! Write a program that does the following:
Begin by creating an integer variable with a value of 5. Add 7 to it, subtract 9, multiply it by 10, and then divide it by 6. Finally, calculate the remainder of the variable when dividing it by 5.

8) Logical operators

Objective: User must write a program that generates one false statement and one true statement.
Intended output: truefalse or falsetrue

Public void playerMain() {

//Let's take a look at some more operators. Instead of arithmetic operators, we call these logical operators. They produce a boolean value (true or false), and you've probably seen them before. Let's look at examples for each logical operator.

Int a = 5;
Int b = 10;
Int c = 10;

// This is the less than operator, which you're likely familiar with. Since a = 5 and b = 10, we have 5 < 10, which evaluates to true.
Boolean test1 = a < b;

//This is the greater than operator, and below it says 5 > 10. This however is incorrect, so it evaluates to false!

```
Boolean test2 = a > b;

//Here, we have the less than or equal to operator. While b is neither less than or greater than c, it is certainly equal to it, as they are both 10. This equates to true.
Boolean test3 = b <= c;

//Here, we have the greater than or equal to operator! Once again, b = 10, and c = 10. Therefore, b >= c is true.
Boolean test4 = b >= c;

//Beware! New programmers may believe “a == c” is the same as “a = c”. It is not!
//”a == c” returns a true or false value after checking to see if a is equal to c. “a = c” assigns the value of c to a.
//Here, a = 5, and b = 10. Since 5 and 10 are not equal, a == c evaluates to false.
Boolean test5 = a == c;

//Finally, we have the “not equals” operator, !=. Here, if a is not equal to b, the expression evaluates to true. A = 5, and b = 10, so a != b is true.
Boolean test6 = a != b;
}
```

Prompt: Go ahead and try this out for yourself! Write a program that generates a true value and a false value, and then prints them out.

9) Logical and/or

Objective: User must write a program that produces two boolean outputs using predetermined values.
Intended output: 2 5 5, and then true true, true false, false false, or false true

```
Public void playerMain() {

//Let's continue our lesson on logic. What if we wanted to measure whether multiple things are true or false? Well, we can.

Int a = 5;
Int b = 10;
Int c = 10;

//Here, we introduce the && operator.
//&& effectively means "and". The English translation of the code below is:
//"If a is less than seven, and b is greater than nine, then the statement is true."
//Indeed, the above statement is true, so test1 will hold the value "true"!
Boolean test1 = (a < 7 && b > 9);

//Below, you can see the || operator.
//This is the "or" operator. Only one condition needs to be met for the statement to be true. The English translation for the below statement is:
//"If a is less than eight, or b is greater than eleven, then the statement is true."
//If we look at our variable declarations above, b is 10, which is certainly not greater than 11! That portion is false. However, a is 5, and definitely less than 8, which is true. So since at least one of the values was true, the entire statement is true.
Boolean test2 = (a < 8 || b > 11);

//You can build more complex logical statements using parentheses.
//Here, the statement ( a == 5 && b == 10) is evaluated separately first as true or false, as it's contained in parentheses! You can use parentheses in logic similarly to how you might use them in mathematics (sometimes, not always).
//This evaluates to true, since all statements below are true.
```

```
Boolean test3 = (( a == 5 && b == 10) && c == 5);

//Finally, you can also combine the && and || operators.
//Here, the output is true.
Boolean test4 = ((a != 6 || b == 10) && c == 5);
}
```

Prompt: Go ahead and try this out for yourself! Write a program where you have three integer values, each holding the values 2, 5, and 5 respectively. Print the values of these three variables. Then, creating an and statement checking to see if 2==5 and 5==5. Then, create an or statement checking the same conditions. Print the true or false values of these and or statements.

10) If/else/if else statements

Objective: User must write a program that checks an integer value against some if statements.
Intended output: 7 is less than 10.

```
Public void playerMain() {

//Let's continue our journey into the world of logic!

//So, we know how to evaluate logical expressions as true or false. That's all fine and dandy, but what if we want to actually do something when a condition is met? That's where if statements come in.

Int a = 5;
Int b = 10;
Int c = 10;

//Let me explain the syntax of the if statement.
//First, we have "if", followed by some logical statement. The program basically says, "if this statement is true, then we will do something". That something is included inside curly brackets { }.
if (b == c) {
        // Since b == c is true, the program moves further into the code and prints the following statement.
        System.out.println("b is equal to c!");
}

//Can you guess what would happen here, since a = 5 and c = 10? Well, a == c evaluates to false. Since the condition is not met, the code inside the if statement will never run! The program will perform its check with the if statement and then simply move on.
```

```
if (a == c) {
        // Since a == c is false, this part of the code never ran.
        System.out.println(“b is equal to c!”);
}

//Similarly to our previous lessons, we can check multiple things with if statements. We do this by utilizing “else if” and “else”. We’ll explain those briefly.

/This if statement is false, so it skips this part, and goes to the “else if” portion.
If (a == b) {
        System.out.println(“a is equal to b!”);
}

//Else if effectively does the same thing, checking the logical outcome of what’s contained in the parentheses. However, the else if portion will only run if the if statement before it is false.
//So, the first if statement was false, therefore this else if statement runs. However, it was also false!
Else if (a == c) {
System.out.println(“a is equal to c!”);
}
//Since the else if statement was false, we move to this. You may notice however that there’s no parentheses! This is because the “Else” condition simply means “if nothing in any previous if/else if statements was true, we will simply run this code”. You could also imagine it as an if statement that always evaluates to true. Each if statement conditional can only have one else statement, and it is always placed at the end.
Else {
System.out.println(“a is neither equal to b or c.”);
}

//Just remember: If, else if, and else all do the same thing. They check to see if a condition is true or false. If it’s true, it does some extra things.
//”If” must always be used first. “Else if” can be used if you want to test multiple conditions, and “else” can be used as a “catch all” if no previous conditions were met.
}
```

Prompt: Go ahead and try this out for yourself! Write a program that has the following:

An integer variable, a, which has a value of 7.

An if statement, which checks to see if a < 5.

An else if statement, which checks to see if a < 10.

An else statement, which checks to see if a >= 10.

}

11) Functions

Objective: User must write a program that has a function which calculates 2^10.
Intended output: 1024

The user must take the previous code and create a function that prints the user input in a readable way. IE: “Your name is X, you are Y years old, and you are from Z”.
Expected output: “Your name is X, you are Y years old, and you are from Z”.

```
Public void playerMain() {

//Lets talk about methods, also called functions in other programming languages!
//Methods are essentially portions of your code that do specific things and help to simplify your code by making certain parts of it reusable.

//This is our function. Let’s step through each part of it. Don’t worry about the “static” portion for now, however.
//Notice the function name has “int” before it, which is a data type. This means that the function will produce an integer as its output. It also means that by defining this, you will need to ensure an integer is output.
//myMultiplicationMethod is the name of the function.
//(int x, int y) specifies how many variables will be sent to this function as input (separated by a comma), and what data type they will be. You can send multiple data types to a single function. Here, we know we will send two variables to the function, and that they are both of type int.
static int myMultiplicationMethod(int x, int y) {
//The “return” portion simply means that “x * y” is the output of the function. Once the function reaches the part that says “return”, it will return the value and stop running the function.
```

```
// The “x *y” portion is what we are returning.
   return x * y;
 }
```

//As you can see, there’s a lot of details for such a small amount of code! This is because methods are incredibly important, and often complex.

```
Int a = 15;
Int b = 20;

//Here, we are using our function to multiple int x and int y. We refer to this as “calling” our function.

Int multiplicationResult = myMultiplicationMethod(a, b);

//Notice: The function returned (or output) a value, and that value was stored in multiplicationResult. If we want to view the result of multiplicationResult, we will have to print it.
      System.out.println(“The result of multiplying a and b is: “ +
multiplicationResult);
}
```

Prompt: Go ahead and try this out for yourself! Write a program containing your own exponential calculator function. Once you’ve created it, call your function and output the result of 2^10.

12) Recursive functions

Objective: User must write a program that has a function which utilizes the function in the explanation when passed a value of 10.
Intended output: 55

```
Public void playerMain() {



//Let's continue our discussion on the power of functions! Take a look at this function below.
//This one might be a little beefy, since it combines a lot of things we've discussed so far. Let's step through it piece by piece to try to understand it.

//Here, we're declaring a function named "sum". We know it returns an int, and also takes in one input, which is also an int.
  public static int sum(int k) {
//We run an if statement, checking to see if the input value is greater than zero.
   if (k > 0) {
//If k was greater than 0, we return k + sum(k-1). Remember this part!
     return k + sum(k - 1);
//If k was not greater than 0, we simply return 0. We haven't talked about this much, but simply returning 0 is an easy way to exit a function.
  } else {
     return 0;
  }
//Notice anything strange? Well, in that part you were told to remember, you can see that the function actually calls itself! We call this a recursive function.
//Basically, the function above adds up itself and all the values less than it via recursion. For example, if we entered 5 on the function call:
```

```
        //We send the value 5 to the function
  int result = sum(5);

        //If you step through the code we have once again posted below, you can see
the function does the following:

        //5 + 4 + 3 + 2 + 1

        //So, the output is 15.
  System.out.println(result);
 }
 public static int sum(int k) {
  if (k > 0) {
   return k + sum(k - 1);
  } else {
   return 0;
  }
}
```

//Recursive functions are incredibly useful, and not so complicated once you understand how they work. They can be used to compute factorials, and even the Fibonacci sequence!

Prompt: Go ahead and try this out for yourself! Write a program with the same function as the one created in the explanation. Call the function and pass it a value of 10, and output the result.

13) While loops

Objective: User must write a program that utilizes a while loop to increment an integer from 5 to 10, printing each time it does so. Once the number reaches 5, a boolean should be set to true and exit the loop.
Intended output: 5678910

Public void playerMain() {

//As we saw in our if statement conditionals in a previous lesson, the program will check the if conditional, and then the remaining else ifs, as well as a final else. What if you wanted to check multiple values though? What if you wanted to iterate through items in a list, or increase the value of a number every time a certain condition is met? Well, that's where loops come in!

//A loop is somewhat similar to if statements. A condition is checked, and if it's met, some code is ran. However, unlike if statements, loops continue to check conditions and execute until a certain condition is met! Let's look at our first kind of loop, called a while loop.

Int count = 0;

//Here, we've define an integer called "count" and set it to zero.
//The while loop, similarly to if statements, will check to see if the following statement in parentheses is true or false. If the statement is true, the loop will execuse, and the code within the curly bracers will run. Since the value of count is less than 3, the code executes.
while (count < 3) {
//When the code executes, count is incremented by 1. On the first loop, count goes from 0 to 1. On the second loop, it increments again, going from 1 to 2. Finally, on

the third loop, since count = 2, and 2 < 3, the condition executes, and count is incremented again, from 2 to 3. When the loop begins again, since count = 3, and 3 < 3 is false, the loop will not be triggered, and its contents will be ignored, causing the program to skip the loop and its contents, printing “loop completed”.

```
  count++;
}
System.out.println(“Loop completed.”);

}
```

Prompt: Go ahead and try this out for yourself! Write a program using while loops that count from 5 to 10. During each loop, print the number prior to incrementing it.

14) For loops

Objective: User must write a program that utilizes an existing list of colors and iterates through it using a for loop.
Intended output: redgreenblue

```
Public void playerMain() {

//Let's talk about another kind of loop: the for loop.


//Just like with if statements and while loops, the "for" portion will evaluate a true and false value within the parentheses. Let's examine the structure of the for loop.
//A for loop looks a bit more complicated than the other conditionals we've seen before, but it's actually quite simple.
//We can see "int i = 0" is declaring an integer, i, with value 0. This is followed by a semicolon.
//Then, we can see "i < 5", which you likely recognize as a true or false statement. This is what will be evaluated by the for loop.
//Finally, we see "i++", indicating that i will be incremented by 1 at some point.
//When the for loop first runs, it creates the variable i, which is equal to 0. If the "i < 5" portion is true, then it will execute the code within the curly braces. Once the code is finished executing, and the loop is about to execute again, i will be incremented by 1. The for loop will then continue to loop and increment i until the statement "i < 5" is false.
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
System.out.println("Loop completed.");
//If we step through the loop, we can see that the program will output the following:
//0
//1
//2
```

```
//3
//4
//Loop completed.

//For loops can be especially useful when iterating through items in a list or array.

}
```

Prompt: Go ahead and try this out for yourself! Write a program using for loops to iterate through the following array and print each value:

String[] colors = {“red”, “green”, “blue”}