CISC 1115 (TY8) Exam 1 Review Sheet

***NOTE**: This review sheet covers most (but not all) topics for your upcoming exam. It is here to clear up any confusion you may have about a particular concept, however, please keep in mind that we cannot and will not provide you with all the answers. Programming is a hands-on skill, please take it upon yourself to understand these concepts AND put them to use practically. Please work on the sample exercises provided; the only way to get better at writing code is by **actually writing code**.*

Topics

- Chapter 2: Variables and operators
- Chapter 3: Input and output
- Chapter 5: Conditionals and Logic
- Chapter 7: Loops

Chapter 2: Variables and Operators

Identifiers and Keywords

- Keywords are predefined, reserved words used in Java programming with special meanings to the compiler.
    - **For example: int, char, double, new, return, static, etc.**
- You cannot use keywords like int, for, class, etc. as variable names (or identifiers) as they are part of the Java programming language syntax.
- Identifiers are the names given to variables, classes, methods, etc.
    - **For example: n, num, number, score, etc.**

Arithmetic Expressions

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

- Pre-increment (++x), i.e. System.out.println(++x); ➞➞➞ result: new value of x
- Post-increment (x++), i.e. System.out.println(x++); ➞➞➞ result: original value of x, store new value for next action
- (=): assignment vs (==): equivalence

Arithmetic Precedence

Java Operator Precedence Table

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Larger number means higher precedence.

Java.lang.Math class methods

- Math Class methods are built-in methods that make performing numeric operations like square, square root, cube, cube root, exponential and trigonometric operations, etc. easier.

- Math class methods include (but are not limited to):

    - **abs()**: java.lang.Math.abs(datatype arg) method returns the absolute value of any type of argument passed. This method can handle all the data types
        - **Parameters:**
            arg: the argument whose absolute value we need
        - **Returns:**
            the absolute value of the passed argument

    - **pow()**: java.lang.Math.pow(double b, double e) method returns the value as b^e
        - **Parameters:**
            b: base
            e: exponent
        - **Returns:**
            value as base^exponent

    - **sqrt()**: java.lang.Math.sqrt(double a) method returns the correctly rounded positive square root of a double value
        - **Parameters**:
            a - a value
        - **Returns:**
            the positive square root of a. If the argument is NaN or less than zero, the result is NaN

    - **random()**: java.lang.Math.random() method returns a pseudorandom double type number greater than or equal to 0.0 and less than 1.0
        - **Returns:**
            a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

        - **To generate a double value within a specific in range:**

        Math.random() * (max - min) + min

        Where min is inclusive and max is exclusive
        (i.e. to generate a number between 1-10, set min to 1 and max to 11)

    - **See the Java API for the full list of Math class methods:**
      https://docs.oracle.com/en/java/javase/22/docs/api/java.base/java/lang/Math.html

Chapter 3: Input and Output

Writing to the Screen (System.out)

- System.out represents the Standard Output Stream. That means that if we want to print any statement on the console, we should use the following statement:

System.out.print();

- There are three methods we use to print statements:
    - **print(String s)**: used to print on the console. It accepts a string as a parameter (we can concatenate parts to the string if necessary, i.e. “Hello” + “ World”, etc. however, it is typical convention to include all string literals in the same pair of quotations if they are consecutive) After printing the statement, the cursor remains on the same line.
    - **println(String s)**: upgraded version of the print() method. It is also used to display text on the console. After printing the statement, it throws the cursor at the start of the next line. It is the main() difference between the println() and the print() method.
    - **printf(String format, datatype args)**: It accepts two parameters:
        - format: It is a formatted String, uses placeholders with specific conversion characters to refer to the arguments that will be replaced when printing
        - args: It is an argument referenced by the format specifiers. If the number of arguments is more than the format specifiers, the other arguments are ignored. The number of arguments may be zero.
        - Conversion characters are only valid for certain data types. Here are some common ones:
            - **%s formats strings (String)**
            - **%d formats decimal integers (int)**
            - **%f formats floating-point numbers (double)**
        - Optional Modifiers:
            - **The [flags] define standard ways to modify the output and are most common for formatting integers and floating-point numbers.**
            - **The [width] specifies the field width for outputting the argument. It represents the minimum number of characters written to the output.**
            - **The [.precision] specifies the number of digits of precision when outputting floating-point values. Additionally, we can use it to define the length of a substring to extract from a String.**
        - %[flags][width][.precision]conversion-character (i.e. %2.2f would print a double value with two decimal places with left padding of two spaces)

        - printf() throws **NullPointerException** if the format is null, also throws the **IllegalFormatException** if a format string contains illegal syntax, and throws an **IllegalFormatConversionException** if the placeholder and corresponding argument are not of the same type.

Read Interactively from the Keyboard (System.in)

- All reading done in Java uses the Scanner class. Using this class, we can create an object to read input from the standard input channel System.in (the keyboard)

- To use the Scanner class, it is necessary to import the class Scanner from the library java.util by including the following line at the top of program:

import java.util.Scanner;

- We can declare a Scanner object by saying:

Scanner sc = new Scanner (System.in);

    - where:

        - sc is the name of the Scanner object (this can be changed and is entirely up to the person who writes the code)
        - and System.in connects our object to standard input (the keyboard)

- The Scanner class has multiple built-in methods that we can access to help us read data, for example: **we can use sc.nextLine() to read in a line of String(s) (spaces included)**

- Other methods include:

    - **nextBoolean(): reads a boolean value from the user**
    - **nextByte(): reads a byte value from the user**
    - **nextDouble(): reads a double value from the user**
    - **nextFloat(): reads a float value from the user**
    - **nextInt(): reads a int value from the user**
    - **next(): reads a complete token (String) from the user. A complete token is preceded and followed by input that matches the delimiter pattern, in most instances this is a space**

- **Sample exercises:**

    1. **Write a Java program that prompts the user for their first and last name and then prints out a message: "Hello, [FIRST NAME, LAST NAME]!"**

        <u>Sample Data (Input/Output)</u>:

        Enter your name: Jane Doe
        Hello, Jane Doe!

Reading and Writing Using Files

- As previously mentioned, we use a Scanner to read data whether or not the data is from the keyboard or a file. **To read from a file we only need to make one or two modifications from what we learned from reading from the keyboard.**

- First, since we are using files we need to allow our program(s) to use them by including:

import java.io.File; //import the File class

- Now we can add a **throws Exception** declaration to main and create the File object that we will be reading from:

File file= new File("[filename].txt");

- And create a Scanner like we have previously done, only changing our reference from the keyboard to the corresponding file object:

Scanner sc = new Scanner(file);

- It is important that you remember to **create the .txt file that you are trying to reading from and make sure that it is in the correct directory or to specify the path where the file is located when creating the File object** else you will receive a FileNotFoundException when you attempt to run your code.

- Writing to a file is slightly less work since we do not have to manually create the .txt file ourselves prior to running the program, the program will do the creation for us.

- Like before, we need to import the PrintWriter class in order to use it:

import java.io.PrintWriter;

- We will use the PrintWriter class to create our file:

PrintWriter pw = new PrintWriter(“[filename].txt”);

- It is important to **make sure that the filename that you are trying to create does not already exist in the directory that you are trying to create it in otherwise, you will overwrite the file and lose your original file data**

- Now we can use the PrintWriter object to write, the same way that we wrote to the screen only replacing System.out with our PrintWriter object, **i.e. pw.print(), pw.println(), pw.printf()**

- **Try modifying the programs that you may have previously done so that they write to both the screen AND to a file**

Chapter 5: Conditionals and Logic

If and if-else statements

- An if statement specifies a statement(s) to be executed only if a particular boolean expression is true.

- An if statement can be followed by an optional else statement, which executes when the boolean expression is false.

- An if statement can be followed by an else if statement to specify a new condition if the first condition is false. If you choose to include an else if statement it **MUST COME BEFORE** the else statement

- Syntax:

    - **if(condition1) {**
      //statement(s) to be executed if condition1 is true
      }
      else if(condition2) {
      //statement(s) to be executed if the condition1 is false and condition2 is true
      }
      else {
      //statement(s) to be executed if the condition1 is false and condition2 is false
      }

- **Sample exercises:**

    1. **Write a Java program that prompts the user for a number and prints whether it is even or odd.**

       <u>Sample Data (Input/Output)</u>:

       Enter number: 104

       104 is an even number.

    2. **Write a Java program that prompts the user for three numbers and prints out the largest of the three.**

       <u>Sample Data (Input/Output)</u>:

       Enter three numbers: 78 25 87

       The largest number is: 87

Nested if-else statements

- Nested if statements means an if statement inside an if statement

- Syntax:

    - **if (condition1) {**
      //statement(s) to be executed if condition1 is true
      if (condition2) {
      //statement(s) to be executed if condition2 is true
      }
      }

- **Sample exercises:**

    1. **Write a Java program that prompts the user for a year and prints whether that year is a leap year or not. (*Note: Leap Years are any year that can be evenly divided by 4. A year that is evenly divisible by 100 is a leap year only if it is also evenly divisible by 400. For example, the years 1700, 1800, and 1900 are not leap years, but the years 1600 and 2000 are.)***

       <u>Sample Data (Input/Output)</u>:

       Enter year: 2016

       2016 is a leap year.

Logical and Relational Operators

- Boolean expressions (evaluates to true/false):
    - Less than: **a < b**
    - Less than or equal to: **a <= b**
    - Greater than: **a > b**
    - Greater than or equal to: **a >= b**
    - Equal to: **a == b**
    - Not Equal to: **a != b**

Integer and Logical Values

- We can include integers in our boolean expressions and determine whether or not they are true or false. For instance, if we were to print out:
    - System.out.print(6 > 5) we would see a result value of ‘true’
    - while System.out.print(6 < 5) or System.out.print(6 == 5) would result in a value of ‘false’ for obvious reasons

Chapter 7: Loops

While Loops

- allows code to be executed repeatedly based on a given Boolean condition (T/F). Needs two parts: **condition/test expression and increment/decrement/update expression**

- Syntax:

    - ```
      int i = 1;
      while(i <= 10) {
          //statement(s) to be executed while the condition is true
          i++;
      }
      ```

    - where:

        - i <= 10 is a boolean expression. I**f the condition evaluates to true, we will execute the body of the loop and go to the update expression.** Otherwise, we will exit from the while loop without reaching the statements inside the body.
        - and i++ increments the variable i

- How many times does this loop run? What would happen if you forgot to include i++;? Try writing code that uses a while loop like this.

- **Sample exercises:**

    1. **Write a Java program that prompts the user for age and prints out "Sorry, you are ineligible to vote." and increments the age until the user is eligible to vote, and prints out "Congratulations, you are eligible to vote!" once they are 18+.**

       <u>Sample Data (Input/Output)</u>:

       Enter your age: 16

       Sorry, you are ineligible to vote.
       Sorry, you are ineligible to vote.
       Congratulations, you are eligible to vote!

    2. **Write a Java program that prompts the user to input an integer and then outputs the number with the digits reversed. For example, if the input is 12345, the output should be 54321.**

       <u>Sample Data (Input/Output)</u>:

       Enter a number: 12345
       Reverse of 12345 is: 54321
      ```

For Loops

- Unlike a while loop, a for statement has the **initialization, condition, and increment/decrement** in one line

- Syntax:

    - **for (int i = 1; i <= 10; i++) {**
      //statement(s) to be executed while the condition is true
      }

    - where:
        - int i = 1; is the initialization
        - i <= 10; is a boolean condition
        - and i++ increments the variable i

- How many times does this loop run? Try writing code that uses a for loop like this.

- **Sample exercises:**

    1. **Write a program to print numbers from 10 to 100 in increments of 10.**

       <u>Sample Data (Output only)</u>:

       10 20 30 40 50 60 70 80 90 100

    2. **Write a program that prompts the user to input a positive integer. It should then print the multiplication table of that number. (0 - 12)**

       <u>Sample Data (Input/Output)</u>:

       Enter a number: 5
       Multiplication Table of 5
       5 x 0 = 0
       5 x 1 = 5
       ....
       5 x 12 = 60

More Complex Loops

- **Nested loops**: if a loop exists inside the body of another loop, it's called a nested loop.

- **Loops with more than one condition**: a condition is a boolean expression and can have more than one part to be considered. For example, to vote you need to be 18+ AND an American citizen. In this case, only when both requirements are true can a person be eligible to vote. It works the same with loop conditions.

    - when using AND (&&), both parts of the condition must be true to evaluate to true
    - when using OR (||), only one part needs to be true to evaluate to true

- **Sample exercises:**

    1. **Write a Java program that prints out each day of the week for a month (approximately 4 weeks).**

       <u>Sample Data (Output)</u>:

       Week: 1
       Day: 1
       Day: 2
       Day: 3
       ....
       Week: 2
       Day: 1
       Day: 2
       Day: 3
       ....
       ....

    2. **Write a Java program that prompts the user for a number of rows and then prints out a half pyramid of asterisks (‘*’) with the same number of rows.**

       <u>Sample Data (Input/Output)</u>:

       Enter the number of rows: 5

       *****
       *** ***
       *** * ***
       *** * * ***
       *** * * * ***

    3. **Write a Java program that prompts the user for a number less than 100 and prints out all the even numbers starting from the number that they entered up to and including 100.**

       <u>Sample Data(Input/Output)</u>:

       Enter a number: 85

       86 88 90 92 94 96 98 100