Java Basics With Addison :3

Getting Started, Printing and Inputting

* The name of the first class in a Java file should be named the same way as the file;
* **public static void main(String[] args){}:** Starts the program;
  + Every command the program contains must be written inside of those brackets.
* The primitive data types are:
  + int: 0, 1, 3…;
  + float/double: 0.25, 3.46, 0.81 etc.;
  + boolean: true or false;
  + char: A single character;
    - Ex: “a”.
* Reference types: Anything defined by classes.
  + Store the location in memory where the information about that object is stored
  + Have methods that can be called;
  + String;
  + Reference types can use .equals() or .compareTo()
* **System.out:** Outputs something;
  + **print:** Prints anything you want;
  + **println:** Doesthe same as print, but adds a new line below.
* **System.err:** Prints something as an error**;**
  + Has the same methods as System.out.
* “try” and “catch”: Error handler
  + You wrap everything that may trigger an error inside of try{}, and catch{} will handle the error in case it’s triggered;
* Inputting stuff into the system: Scanner variableName = new Scanner(inputStream);
  + Initializes a Scanner;
  + E.g.: Scanner input = new Scanner(System.in);
  + **Scanner.nextLine():** In case the input must be a String;
  + **Scanner.nextInt():** In case the input must be an integer;
  + **Scanner.nextDouble():** In case the input must be a decimal number;
  + **Scanner.nextBoolean():** In case the input must be a true/false value;

Operations

* Additions: System.out.print(n + m);
* Subtractions: System.out.print(n - m);
* Multiplications: System.out.print(n \* m);
* Divisions: System.out.print(n / m);
  + Be careful with divisions between integers, if the result is meant to be a double, it’ll only print the integer part (before the dot);
  + System.out.print(**(double)**n / m): Divides both numbers as doubles;
  + System.out.print(n / **(double)**m): Divides both numbers as doubles;
  + System.out.print(**(double)**n / **(double)**m): Divides both numbers as doubles;
  + System.out.print(**(double) (**n / m**)**): Due to the parenthesis between the two numbers to be divided, they’re divided as integers and then the result is converted to double;
  + (double) can be used as (int) in case you want to transform a double into an integer;
* Modulus: System.out.print(n % m);
  + Negative Modulus: You’ll use the signal from the number on the left side as a reference.
    - Ex: 6 % -4 = 6 % 4.
* Order of operations:
  + Parentheses;
  + Method calls, like Math.sqrt() and Math.pow();
  + Multiplication and division;
  + Addition and subtraction;
* Boolean Operations: Comparison between two or more values;
* Order of boolean operations:
  + Parentheses;
  + Method calls;
    - Ex.: String#method().
  + Math operations;
  + Comparisons:
    - Greater than (>);
    - Lesser than (<);
    - Greater or equal than (>=);
    - Lesser or equal than (<=);
    - Equal (==);
    - Not equal (!=).
      * **Note: Only use comparisons on primitive types (int, double, Boolean, char).**
  + Not (!);
  + And (&&);
  + Or (||).
    - **Note: “&&” has priority over “||”.**
* Advanced operators:
  + **“varName = x”:** Assigns a value to a variable;
  + **“varName += x”:** The same as “varName = varName + x”;
  + **“varName -= x”:** The same as “varName = varName - x”;
  + **“varName \*= x”:** The same as “varName = varName \* x”;
  + **“varName /= x”:** The same as “varName = varName / x”;
  + **“varName %= x”:** The same as “varName = varName % x”;
  + **“someInt++”:** Increments a value and then displays it;
    - **The same idea works for “someInt--”.**
  + **“++someInt”:** Displays the integer and then increments it;
    - **The same idea works for “--someInt”.**

Classes

* Used to create new reference types;
* Must be in a file with the same name as the class;
  + No need to use the main method inside of those files, as they’ll be ran by a class in another file;
  + E.g.: Table must be in table.java.
* A class file must contain:
  + Instance variables: Data about the object;
    - Usually kept “private” for encapsulation;
  + Constructors: Build the class;
    - The only method types with no return type (int, boolen…);
    - public ClassName() {};
    - **“new Class()”:** Calls the constructor.
  + Getters (aka Accessors):They access private information to be used wherever they need to be used;
    - Must have a return type;
    - **“public dataType getVariable(){}”.**
  + Setters: Sets a new value to an existing variable;
    - public dataType setVariable(dataType variable){this.variable = variable};
  + toString: Converts a reference type to String so Java can print it when requested to;
* Overload: Multiple methods with a same name, but with different parameters;
  + E.g.: Table(){} and Table(int length, int height, int width){return variable};
* Signature: Name and data types of the constructor;
  + E.g.: **Table(int length, int height, int width)** is the signature of the constructor;
* **“this.”:** Differs the instance variables from the ones in the signature;
* **Methods:** Parts of code that you want to use frequently (function);
* Methods must have:
  + An access modifier (public or private);
  + A return type (reference type, a primitive or void (returns nothing));
  + A name to call the method;
  + The method body inside of braces ({});
* They can also have:
  + One or more parameters (things inside the parenthesis);
  + Other parameters, like “static” (Accesses a method using the class itself instead of an object related to this class);
    - E.g.: “public static int area 🡪 Table.area(){}”, while “public int area 🡪 table1.area(){}”;
* Examples of methods:
  + Constructors;
  + Getters;
  + Setters;

Conditionals and Loops

* Ifs and Whiles require a boolean inside of the parenthesis
* **“If(){}”:** What’s inside of the braces will be executed **if** the parameters are true;
  + They don’t need the argument “else{}” to work;
  + **“else if()”:** In case of the first if to be false, you can use else if() to make the program check if another conditionals are true instead of the first ones.
* **“while(){}”:** Executes what’s inside of the braces until the parameters become false;
  + Used when you **don’t know** how many times you want to execute a certain command;
  + If the conditions are already false, the piece of code doesn’t execute;
  + If there’s none of the values get changed, it’ll generate an infinite loop.
* **“for([starting point]; [conditional]; [command to increment the initial value]){}”:** Makes the same thing as the while loop, but it’s used when you know the amount of times you want to execute a certain command;
  + E.g.: for(int i = 0; i < 5; ++i){System.out.println(“Hello, world!”)}.

Arrays

* **Arrays:** Data structure used to store a list of information of the same data type;
  + To access or create an array, use brackets (“[]”);
  + Arrays have a predefined size established when they get created;
* Ways of creating an array:
  + dataType[] arrayName = new dataType[l];
    - No spaces between int and the brackets;
    - l stands for the length of the array;
    - Array created: [0, 0, 0, 0, 0, …].
  + dataType[] arrayName = {values};
    - You insert the values inside of the braces;
    - E.g.: int[] array1 = {1, 2, 3, 4, 5}.
* Printing an array: You’ll need to use the “#toString()” method;
  + System#out#println(Arrays#toString(arrayName));
    - E.g.: System.out.println(Arrays.toString(array1));
    - Array printed: [1, 2, 3, 4, 5].
* Index: Position where a certain element is in an array;
  + Starts in 0, not in 1;
* Length: The amount of elements an array has;
  + E.g.: An array with length 4, will have elements with index 0, 1, 2, and 3;
* **Enhanced For / For Each:** For each index value in an array, execute the command inside of the braces;
  + **“for (dataType x : arrayName){}”;**

String Methods

* **stringName#equals(anotherString):** Compares if two strings are equal, returning a Boolean value;
  + **Case sensitive;**
* **stringName#compareTo(anotherString):** Compares the Unicode values of the first non-equal characters in two Strings, returning an integer;
  + **Case sensitive;**
  + If the value is less than 0, stringName comes before anotherString alphabetically;
  + If the value is greater than 0, stringName comes **after** anotherString alphabetically;
  + If the value is equal to 0, both strings are equal;
* **stringName#length():** Returns the number of characters in a String;
* **stringName#substring(index1, index2):** Returns the characters from index1 to (index2 – 1);
  + E.g.: In a String “Addison”, String#substring(0,3) would return “Add”.
* **stringName#charAt():** Grabs the character of a given index in a String and returns it as a char;
  + E.g.: In the same String “Addison”, String#charAt(3) would return “i”.
* **stringName#indexOf():** Returns the first occurrence of a certain character;
  + E.g.: In the String “Addison”, String#indexOf(“A”) returns “0”.
* **stringName#lastIndexOf():** Returns the **last** occurrence of a certain character;
  + E.g.: In the String “Addison”, String#lastIndexOf(“d”) returns “2”.
* **stringName#toUpperCase():** Returns a new string with all the characters being in uppercase;
  + E.g.: The value returned when using stringName#toUpperCase() in the String “Addison” is “ADDISON”.
* **stringName#toLowerCase():** Does the same things as the command before, but turning all the chars to lowercase instead;
* **stringName#contains():** Returns a Boolean value about if a certain String contains a certain sequence of characters or not;
* **stringName#isEmpty():** Returns true if there’s absolutely **nothing** in a String;
  + Doesn’t consider white spaces;
* **stringName#isBlank():** Returns true even with the presence of white spaces;
* **stringName#trim():** Trims the white space existent before the first letter and after the last one;