**Java 1 Student Manual**

**Education Unlimited Summer 2017**

**Command Line Basics:**

Open your terminal. Here you can type commands. Commands can have parameters, which are optional letters you type after the command after a dash (-). Press “Tab” to complete the time of what you’re typing. “.” is the current folder, “..” is the parent folder, “\*” is all files in your location. Common commands:

* ls: list all files in the current location
* cd: change directories.
  + Example: if you have a folder called “stuff” in your location, typing “cd stuff” will move you into the folder “stuff.”
  + Typing “cd ..” will take you up to the parent folder
* mkdir folderName: makes a new folder with the name “folderName”
* rmdir folderName: removes the folder called “folderName”
* rm fileName: removes the file called “fileName”
  + Typing “rm \*” will remove all the files in your current directory
* nano fileName: very simple file creator/editor. Only supports typing. When done, type “control-x”, then “y”.
* clear: clears the screen
* javac fileName.java: compiles the Java file
* java fileName: runs the compiled Java file

**Git Basics:**

* Create a github account here: <https://github.com/>, download git to your computer.
* Creating a local repo: When creating a new project on your local machine using git, you'll first create a new repo. To use git you'll be using the terminal. To begin, open up a terminal and move to where you want to place the project on your local machine using the cd command. Then use the mkdir command to make a folder for your project. Move into the folder. To initialize a git repository in the folder, run “git init”
* Creating and committing a file: Go ahead and add a new file to the project, using any text editor you like. After creating the new file, you can use the “git status” command to see which files git knows exist. The response basically says is, "Hey, we noticed you created a new file, but unless you use the 'git add' command we aren't going to do anything with it." Add the file to the staging environment using the “git add” command. Run the command “git commit -m "Your message about the commit."” The message at the end of the commit should be something related to what the commit contains - maybe it's a new feature, maybe it's a bug fix, maybe it's just fixing a typo.
* Creating a github repo and connecting it to your local: To create a new repo on GitHub, login and go to the GitHub home page. You should see a green '+ New repository' button. After clicking the button, GitHub will ask you to name your repo and provide a brief description. When you're done filling out the information, press the 'Create repository' button to make your new repo. GitHub will ask if you want to create a new repo from scratch or if you want to add a repo you have created locally. In this case, since we've already created a new repo locally, we want to push that onto GitHub so follow the **'....or push an existing repository from the command line'** section. Now we'll **push** the commit in your branch to your new GitHub repo. To push changes onto a new branch on GitHub, you'll want to run “git push origin yourbranchname”. If you don’t know your branch name, run the command “git branch” and the name with an asterisk (\*) next to it is the branch you are on.

**NetBeans:**



The white rectangle with green plus is add new file, the yellow one next to it is add new project. The Green arrow is run, the smaller green arrow to the right of it is Step Through, or debug: goes through your code step-by-step.

**Java**

**Variables**

Most used types of variables:

* Int: an integer
* Double: a floating point value (ex. 1.25)
* Boolean: can only be true or false
* String: a string of characters. Always defined using quotation marks

Creating a variable:

* type name;

name = value;

* type name = value;

Names can be words and they always start with lowercase letter. The convention for multiword variable names is “firstSecondThird.”

Example of creation and use of variables:

int c=1;

double x,y,z;

boolean isSummer=true;

String morning = "Good morning, class!";

int d = c+1;

System.out.println("isSummer = " + isSummer);

Changing between types of variables:

String to int: Integer.parseInt(string). String to double: Double.parseDouble(string).

Int to String: Integer.toString(int). Double to String: Double.toString(double)

**Math Operations**

* The plus sign (+) will add integers or doubles. If used with Strings, it will concatenate (merge) them Strings together.
* The minus sign (-) will subtract one number from another. It works with two integers, doubles, or an integer and a double (in any order).
* An asterisk (\*) is used for multiplication. The order of operations (multiplication, division, addition, subtraction, left to right) is used by Java.
* A slash (/) is used for division. When an integer is divided by another integer, the result is an integer and double, so use doubles for any divisions that will not yield whole numbers.

**Math Shortcuts**

* += : This sets the variable equal to itself plus the following number. Ex. i += 4; is the same as i = i + 4;.
* -= : This sets the variable equal to itself minus the following number. Ex. i -= 2; is the same as i = i - 2;.
* ++ : This sets the variable equal to itself plus one. Ex. i++; is the same as i = i + 1;.
* -- : This sets the variable equal to itself minus one. Ex. i--; is the same as i = i - 1;.

**Logic Operators for Numbers**

* == : Equals. Returns true if the two numbers are equal, and false if the two numbers are not equal.
  + 4==4 returns true, 2==0 returns false.
* != : Not Equals. Returns true if the two numbers are not equal, and false if the two numbers are equal.
  + 6!=7 returns true, 3!=3 returns false.
* > : Greater than. Returns true if the first number is greater than the second, and false if it is not.
  + 7>4 returns true, 7>10 returns false, 7>7 returns false.
* < : Less than. Returns true if the first number is less than the second, and false if it is not.
  + 3<8 returns true, 3<1 returns false, 3<3 returns false.
* >= : Greater than or equal to. Returns true if the first number is greater than or equal to the second, and false if it is not.
  + 9>=1 returns true, 9>=9 returns true, 9>=12 returns false.
* <= : Less than or equal to. Returns true if the first number is less than or equal to the second, and false if it is not.
  + 2<=5 returns true, 2<=2 returns true, 2<=0 returns false.

**Logic Operators for Strings**

* .equals: used to see if two Strings are equal. Returns true if they are the same, including cases, punctuation, and space.
  + Usage: String1.equals(String2);
  + Example: “hello!”.equals(“hello!”); returns true
  + Example: “Hello”.equals(“hello”) returns false.
* .compareTo : used to see if one String comes before the other alphabetically. The result is a negative number if the first String comes before the second, it is zero if they are exactly the same, and it is positive if the first String comes after the second String.
  + Usage: String1.compareTo(String2);
  + Example: “hello!”.compareTo(“hello!”); returns 0 because they are exactly the same
  + Example: “hello!”.compareTo(“goodbye”); returns 1 because ‘g’ is behind ‘h’ in the alphabet
  + Example: “hello!”.compareTo(“hi!”); returns -4 because ‘e’ is ahead of ‘i’ in the alphabet

**Compound Boolean Logic**

* ! : “not” operator. Takes one boolean and reverses it. Returns true if the input is false, returns false if the input is true.
* && : “and” operator. Takes two booleans and returns true if they are both true, and false if either one or both of them is false.
* || : “or” operator. Takes two booleans and returns true if either of them are true, and false if both of them are false.

**Capturing User Input**

* If you are will be using keyboard input, you must put “import java.util.Scanner;” at the top of your Java file. Then, inside the class, create the scanner with this line of code: “Scanner keyboard = new Scanner(System.in);” ‘Keyboard’ is the name of the scanner and can be changed to whatever variable name you choose.
* You can capture user input the following ways:
  + keyboard.nextLine(); captures the next line the user inputs until they press enter.
  + keyboard.next(); captures the next word (no spaces) the user inputs.
  + keyboard.nextInt(); captures an integer if that is the next user input. This will cause an error if the user enters anything other than an integer.
  + keyboard.nextDouble(); captures a double if that is the next user input. This will cause an error if the user enters anything other than a double.

**Conditionals (If-Else Statements):**

* Conditionals allow certain statements in Java to only run under certain conditions.
* If Statements consist of the word ‘if’ followed by a boolean expression (something that evaluates to true or false, usually in parentheses). If the expression evaluates to true, the following code enclosed in curly braces ({ }) will execute. If the boolean expression is false, then the code with be skipped.
  + Example:

if (age >= 10) {

System.out.println("Congrats! You hit double digits!");

}

* + If you enter 7, then nothing will print. If you enter 11, then "Congrats! You hit double digits!" will print.
* If-Else statements have an If statement followed by an Else statement. The code block after the if will execute if the boolean expression is true, and the code block after the else will execute if the boolean expression is false.
  + Example:

if (age >= 10) {

System.out.println("Congrats! You hit double digits!");

}

else {

System.out.println(“You’re still a kid!”);

}

* + If you enter 7, then “You’re still a kid!” will print. If you enter 11, then "Congrats! You hit double digits!" will print.
* Optionally, there can be ‘Else if’ statements between the if and the else statements. In a string of these statements, only 1 will ever run.
  + Example:

if (age >= 10) {

System.out.println("Congrats! You hit double digits!");

}

else if (age > 3) {

System.out.println(“You’re a kid!”);

}

else {

System.out.println(“You’re a baby!”)

* + If you enter 7, then “You’re a kid!” will print. If you enter 11, then "Congrats! You hit double digits!" will print. If you enter 2, then “You’re a baby!” will print.

**While Loops**

* While loops are often used to keep doing a process until something happens - until a user enters the magic password, rolls doubles, etc. Sometimes they are used to do something a set number of times using a counter.
* While loops consist of the word ‘while’ followed by a boolean expression (something that evaluates to true or false, usually in parentheses). If the expression evaluates to true, the following code enclosed in curly braces ({ }) will execute and then it will check the boolean expression again. As long as the expression is true, the loop will keep repeating. If the boolean expression is false, then the code with be skipped and execution will move on to the next code outside of the while loop.
* Warning: Infinite loops! If you do not ever change the boolean expression and it stays true, your program will go into an infinite loop and you will have to manually stop it. Make sure there is always something in your while loop that will eventually cause the boolean expression to become false.
* Example:

int i=0;

while (i<5) {

System.out.println(i);

i++;

}

This will print the numbers 0 through 4 on separate lines.

**Random Numbers**

To generate a random number, use the Math.random() function. Math.random() will generate a random number between 0 and 1 (including 0 but not including 1). If you want a larger range, you can multiply the random number by a larger number. For example, for a random whole number between 1 and 50, use:

int random = (int )(Math.random() \* 50 + 1);

**Functions / Methods**

Functions are called Methods in Java. It is a set of code which has a name and can be called (used) at any point in a program by using its name. Methods are like subprograms that act on data and often return a value.

* Example of a void function with no input variables:

public static void sayHello() {

System.out.println("Hello, there! How are you today?");

}

* Example of a function which takes in 2 doubles and returns a double that is their sum:

public static double addNums(double n1, double n2) {

return n1+n2;

}

**Arrays**

An array is a collection of elements that all have the same type. The length of the array (the number of things in it) is set when you create the array, and cannot be changed after that. The square brackets [ ] are used to represent an array. The indices (singular index) for arrays start at 0, so if you have an array of size 10, the indices go from 0 to 9.

* Initializing an array: int[] myArray = new int[10]; creates an array of integers of size 10.
* Putting values into the array: myArray[0] = 2; myArray[1] = 4; makes the first 2 ints in the array 2 and 4.
* Getting array elements: System.out.println(“The first element in the array is: ” + myArray[0]); will print out the first element in the array.
* Getting the length of the array: myArray.length. So the indices go from 0 to length-1.

**For Loops**

* For loops are used to do something a certain number of times or to go through an array and do something to each element.
* For loops consist of the word ‘for’ followed by (in parentheses) a counter (usually an int) set to an initial value (usually zero), then a semicolon, then a boolean expression for when the for loop should stop (usually the counter variable less than another number), then a semicolon, then
* Example: If you have an array called myList, this will print all the elements of myList separated by spaces.

for (int i = 0; i < myList.length; i++) {

System.out.println(myList[i] + " ");

}

**Object Oriented Programming**

Object Oriented Programming (OOP) is a way of programming based on the concept of "objects", which contain data (fields, attributes), and procedures (methods). The idea is, if you make a class to represent a Person, which has attributes Name and Age and a method called getName(), then if you create a person object elseware you can use its getName method.

Example: Person has a getName() method that returns a String.

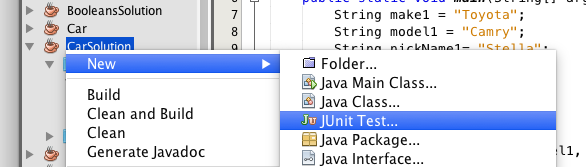
Person p = new Person();

String pName = p.getName();

**Unit Testing**

Unit tests are test individual parts of the code and make sure that all code works as intended. Test Driven Development (TDD) is the process of writing tests before writing the methods themselves. This way, you know exactly what the intended outcomes are before you start coding.

Junit is a testing framework for writing and running unit tests. In Netbeans, to create a new Junit test set, right click on the project you want to test, select ‘New’, then select ‘Junit test…’



Name your test class the name of the class you are testing, followed by the word Test. Uncheck all of the boxes and select the highest number version of Junit.

Tests look like this:

@Test

public void makeTest() {

assertEquals(c.getMake(), "Toyota");

}

There should be at least 1 unit test for each method, and for complicated methods there should be at least 1 test for each possible outcome. To run tests, right click on the file and click ‘Run File’.

**File I/O**

There are a lot of different ways to read and write files in Java. We just look at the simplest ways to do it, BufferedWriter and BufferedReader. Always enclose them in a try block and catch IOException after. Requires at the top of the class: import java.io.\*;

How to create and write to a file with pre-defined name FILENAME:

BufferedWriter writer = new BufferedWriter(new FileWriter(FILENAME));

writer.write(content); //content is a String

How to read from a file with name FILENAME:

BufferedReader reader = new BufferedReader(new FileReader(FILENAME));

line =reader.readLine(); //always check if the line is null, that means you have reached the end of the file.

**Polymorphism**

Polymorphism means many forms. It is the Object Oriented Programming concept that one object can have many forms and that one category can include many types of things. For example, if you want to make classes that represent different types of animals, there are some things they have in common. You can create an Animal interface, with many different implementations for the different types of animals.

**ArrayLists**

ArrayLists are very similar to arrays, because they are lists that contain the same type of object. Indices also start at 0 and go to size-1. However, unlike arrays, they change size depending on how many objects they contain.

Methods:

* Constructors
  + ArrayList a = new ArrayList(); creates a new empty ArrayList.
  + ArrayList(“a”, “b”, “c”) creates a new ArrayList containing “a”, “b”, and “c”.
* Modifying the ArrayList
  + a.add(“d”); adds “d” into the end of ArrayList a, so a = [“a”, “b”, “c”, “d”].
  + a.add(1, “d”); adds “d” into the index 1 in ArrayList a, so a = [“a”, “d”, “b”, “c”].
  + a.remove(“a”); will remove “a” if it is in ArrayList a.
  + a.remove(1); will remove the element at index 1 in ArrayList a. If previously a=[“a”, “b”, “c”, “d”], then after this it will be [“a”, “c”, “d”].
* Getting information about an ArrayList
  + Boolean b = a.contains(“c”); will return true if “c” is in ArrayList a, false if not.
  + String s = a.get(2); will get the element at index 2 of ArrayList a.
  + int i = a.indexOf(“d”); will give the index of “d” in ArrayList a.
  + int j = a.size(); will give the size of ArrayList a.
  + Printing an ArrayList will print all the elements surrounded by square brackets like this: [1,2,3].

**Other Programming Languages**

**Python**

Python is an interpreted, high-level programming language. It used general-purpose, so it can be used for a lot of different types of projects. Python differs from many other languages in its use of whitespace instead of punctuation, and its extremely efficient and succinct syntax.

**C**

C is an imperative programming language with a static type system. C has a memory-use system that uses pointers, and is very fast. C is often used in operating systems, compilers, editors, databases, and database management systems.

**C++**

C++ is built off of C, so it is compatible with most C programs. It’s a compiled language, and can be one of the fastest languages when used correctly. It is an object-oriented language. Like C, C++ has low-level memory manipulation (pointers).

**Racket**

Racket is a multi-paradigm programming language. It is related to well-known languages Lisp and Scheme. It is used as a platform to develop other languages, for scripting, and for graphics. It is considered a great learning language.