**0: Getting Started**

* Use the Linux operating system to do basic things: view web pages, use editors
  + Opens the webpage in terminal **Open (webpage name)**
  + Editor for java in terminal **geddit \_\_\_\_\_.java &**
  + Other commands to know **cd, ls, mkdir**
* Compile and run simple programs written by others
* Write, compile and run simple programs you have written
  + To compile **javac \_\_\_\_\_\_.java**
  + To run **java \_\_\_\_\_\_\_**

**1: Following Commands**

* Explain what is meant by an algorithm and defensive programming
  + **Algorithm:** Series of instructions followed to achieve a task
  + **Defensive Program:** Writing code to improve the chances that it will work even though you would not in unforeseen circumstances
* Write programs that print out a message to the user
  + Printing to the terminal: **System.out.println();**
  + Printing in another window:
    - **import javax.swing.\***
    - **JOptionPane.showMessageDialog(null, “\_\_\_\_\_”)**
* Write programs that read input from the user and store it in a variable
  + Taking in input from the terminal
    - **import java.util.Scanner;**
    - **Scanner scanner = new Scanner(System.in);**
    - **(varname) = scanner.nextLine();**
  + Taking input from a pop up window
    - **import javax.swing.x;**
    - **JOptionPane.showInputDialog(“\_\_\_\_\_\_”);**
  + Writing methods that take in inputs that are STRINGS
    - **Import java.util.Scanner**
    - **public static String inputString(String message){**
    - **System.out.println(message)**
    - **Scanner scanner = new Scanner(System.in);**
    - **return scanner.nextLine();**
    - **}**
  + Writing a follow up method that takes an input of INTEGERS
    - **public static int inputInt(String message){**
    - **return Integer.parseInt(inputString(message));**
    - **}**
  + Calling the input methods created
    - **String (varname) = inputString(“\_\_\_(message)\_\_\_”)**
    - **int (varname) = inputInt(“\_\_\_\_(message)\_\_\_\_”)**
* Write programs that print out messages that include the contents of variables
  + Concatenating strings and contents of variables
    - **System.out.println(“\_\_\_(message)\_\_\_” + (variable))**
* Write programs that are split into methods in simple ways
* Write and use methods that both do and don’t return results
  + **Return:** The return statement allows the method to retake control of the program flow to continue where it left off
* Explain how your above programs work and the concepts involved
* Explain what is meant by a method, function and procedure
  + **Method**: This is a named operation – Example **main() system.out.println()**
  + **Function**: A function is a type of method that returns a value that can be used later in computation
  + **Procedure:** A procedure is a type of method that just does instructions but does not return a value.

**2: Manipulating Values, Using Types**

* Write, debug and run simple programs that manipulate data
* Write and debug programs that do calculations on values stored
  + **int num1 = 5;**
  + **int num2 = 10;**
  + **System.out.println(num1 \* num2);** (Output of 50)
* Write and debug programs that print out messages that include results of calculations
  + **System.out.println( 4 + 5);** (Output of 9)
* Write and debug programs that store and manipulate different types of values.
* Write and debug programs that create and use simple record types.
  + **Record:** A record is a data structure that binds different values together into a single combined value.
  + **Creating a new record:** You can create a new record by creating a new class outside of the curly brackets of the main. The class is like defning a blueprint for what a variable type of the record will be
    - **class mainprogram{**
      * **Main program code + main method**
    - **}**
    - **//Name, age and are instance identifiers which give the separate values of the record**
    - **class recordtype{**
      * **String Name**
      * **Int Age**
      * **<Type> Identifier**
    - **}**

**Student pupil1 = new Student();**

**Variable** called **pupil1** that has a **type Student**

Example of making a record

**class studentrecord {**

**public static void main(String []p){**

**Student pupil1 = new Student();**

**pupil1.name = “Paul Curzon”;**

**pupil1.id = “4509930”;**

**pupil1.mark = 89;**

**System.out.println(pupil1.id + “ “ + pupil1.name + “ “ + pupil1.mark);**

**}**

**}**

**class Student {**

**String name;**

**String id;**

**int mark;**

**}**

* Explain how your above programs work and the concepts involved.
  + Commenting code using
    - **// (Comment)**
    - **/\* (Comment over a few lines) \*/**
* Explain the importance of types
  + Grouping data into types with similar properties is beneficial because
    - The computer can make appropriate storage space for the data
    - Instructions deal with the data appropriately (**Eg Multiplying strings is pointless as it means nothing**)
  + **String:** Values that are in quotation marks (Eg **“Hello”)**
  + **Int:** Values that are whole numbers which can be positive or negative (Eg **55 -89 0**)
  + **Char:** Single characters (Eg **‘a’ ‘b’ ‘z’**)
  + **Boolean:** Only has two possible values **(True False)**
  + **Double:** Floating point numbers that can be positive or negative (Eg 3.14159)
  + You can also convert data types using:
    - **Integer.parseInt(\_\_\_\_)** – Converts strings to integers
    - **Double.parseDouble(\_\_\_\_\_)** – Converts string into doubles

3**: Making Decisions, more on methods and records**

* Write and debug programs using Boolean expressions
* Write and debug programs that make decisions
  + Method used to compare strings exactly
    - **variableToCompare.equals(“\_\_\_\_”)**
  + Method used to see if a variable contains something
    - **variableToCompare.contains(“\_\_\_\_\_”)**
* Explain the use of Boolean expressions
  + Creating Boolean expressions
    - **Equal ==**
    - **Not Equal !=**
    - **Less than <**
    - **Less than or equal to >**
    - **Greater than <=**
    - **Greater than or equal to >=**
    - **And &&**
    - **Or ||**
* Explain what is meant by if statements
  + If statements allows the program to make a decision on how to react in response to a situation (Eg User input, outcome of a calculation)
    - **if (<test>){**
      * **<statement>**
    - **}**
    - **else{**
      * **<statement>**
      * **}**
* Trace programs containing if statements
* Write and debug methods that take arguments and return results
  + The typical way a method is written is **public static void method1()**
  + However, when a variable needs to be returned back from the method to the main, the **void** needs to be replaced with the data type of the variable being returned
* Write and debug programs that use accessor methods
  + **Accessor Methods:** Accessor methods extract information from a record
  + **Abstraction:** Hiding the details of implementation from the user
    - **A new record has been made called Student**
    - **class Student{**
      * **String name;**
      * **int ID;**
      * **int age;**
      * **}**
  + **Setters:** A method to allow you to set the value of a field in a record
    - **//This method sets the name and takes the argument of the name of the new student which will be used in the main class.** 
      * **public static void setName(String name){**
        + **this.name = name;**
      * **}**
      * **public static void setID(Int ID){**
        + **this.ID = ID;**
      * **}**
      * **public static void setAge(int Age){**
        + **this.age = age;**
      * **}**
  + **Getters:** A method to allow you to get the value of a field in a record
    - **//This method has no arguments as it will just return a value**
      * **public static String getName(){**
        + **return name**
      * **}**
      * **public static String getID(){**
        + **return ID**
      * **}**
      * **public static String getAge(){**
        + **return age**
      * **}**
  + Using setters and getters to add data items to the record
    - **//Creating a new student using the Student class, setters and getters**
      * **public static void main(String args[]){**
        + **Student James = new Student();**
        + **James.setName(“James”);**
        + **James.setID(12345);**
        + **James.setAge(16);**

**Student Jane = new Student();**

**Jane.setName(“Jane”);**

**Jane.setID(23334);**

**Jane.setAge(“12”);**

**//Printing the values to the screen – using the getters**

**System.out.println(“James’ ID is ”+ James.getID());**

* + - * **} Output: Jame’s ID is 12345**
* Explain concepts related to and the use of methods

4**: Bounded For Loops**

* Write and debug programs that follow instructions a fixed number of times
* Explain what is meant by a for loop
  + **For Loop:** A counter controlled loop that used a counter to repeat code a fixed number of times.
  + **Counter:** Keeps track of how many times the loop block has been executed
  + **Terminating Variable:** Setting the Boolean expression that terminates the for loop
  + **Accumulator Variable:** A variable that gradually builds up a result a bit at a time
  + **Loop body:** The block that is repeatedly executed under the loop.
  + The general form of a loop is

**for (int i = <start>; i <= <finish>; i++){**

**<body>**

* Trace the execution of programs containing for loops

**Other Key Words**

**Algorithm:** Series of instructions that when followed achieve some task

**Program**: Algorithm that is written in a special language

**Return**: Gives control back to the method that called it so it can carry on from where it left off

**Method**: A named operation (**main() System.out.println()** )

**Class**: A name group of collected methods

**Keyword**: A reserved word that has a special meaning in Java and can’t be used for anything else

**Variable**: Variables are what languages use to store information in a way that it can be accessed later.

**Declaring Variables**: Writing a statement to introduce the variable (**String name**)

**Assigning Variables**: Putting something into the variable/ setting the contents of a variable (**name = “Fred”**)

**Function**: Returns a value that you can use later in the computation **scanner.nextLine()**

**Procedure:** A method that simply does something (no value is returned to put somewhere) **System.out.println(“Hello”)**

**Compile Time Errors:** Grammar and spelling mistakes that the compiler picks up. These errors will not allow the code to run. **(Eg forgetting to use a semi colon)**

**Run Time Errors:** Run time errors are when the program does run but does the wrong thing **(Eg. Printing out the wrong message)**

**Compiler:** A compiler is a program that converts instructions into machine code so that they can be read and executed by a computer.

**Value:** Another name for data

**Expressions**: A fragment of a program that evaluates to give a value Eg **(a+1)\*5** or **“Hello” + name + “.”**

**Block:** Group of simple statements denoted by {}

**{ System.out.println(“Line 1”)**

**System.out.println(“Line 2”)**

**}**

**Systematic Testing:** Testing each possible program behaviour not each possible input.