**-Problem solving:** You have to understand the problem and have a carefully planned approach before writing a program. You should also understand available building blocks and proven program construction techniques.

1) Understand the problem thoroughly.

2) Formulate algorithm using pseudocode and top-down stepwise refinement. We should decide programming languages and data structures to use and checks we have to put to avoid errors before starting to code the program.

3) Write the program.

4) Test, debug, test. (Test with complete sets of data and cover all possibilities.)

**-Algorithm:** Any computing problem can be solved by executing a series of actions in a specific order. A procedure for solving a problem in terms of the actions to execute and the order in which these actions execute is called an algorithm.

Specifying the order in which statements(actions) execute in a program is called program control. We control the program using control statements.

There are different ways of specfying algorithms. Algorithms can be described in various ways, from pure mathematical formulas to complex graphs, more times than not, without pseudocode. Pseudocode describes how you would implement an algorithm without getting into syntactical details.

**-Pseudocode:**

Euclid’s algorithmfor computing gcd*(m, n)*:

Step 1If *n* = 0, return the value of *m* as the answer and stop; otherwise, proceed to Step 2.

Step 2Divide *m* by *n* and assign the value of the remainder to *r*.

Step 3Assign the value of *n* to *m* and the value of *r* to *n*. Go to Step 1.

**-Pseudocode that looks like code:** We dont obey to every single rule a programming language has. We use a standart “language” to display our algorithm

Is an informal language that helps you develop algorithms without having to worry about the strict details of a certain language syntax. Pseudocode can be anything but it should be something simple(no code knowledge required to understand) like english language. You can use pseudocode to explain your code to people who both know and dont know programming. Generally pseudocodes represent only actions in a program not variables.

Pseudocode does not execute on computers. Rather it helps you plan a program before starting to write it in a programming language.

Euclid’s algorithmfor computing gcd*(m, n)*

// Computes gcd(m, n) by Euclid’s algorithm

// Input: Two non-negative, not-both-zero integers m and n

// Output: Greatest common divisor of m and n

**while** n != 0 **do**

r 🡨 m mod n

m 🡨 n

n 🡨 r

**return** m

**-Flowchart:** This representation technique has proved to be inconvenient for all but very simple algorithms. It was dominant way of specifying algorithms in the earlier days of computing. [UML Notes.docx](../../../8-%20Java/3-%20Notes/Algorithm,%20UML/UML%20Notes.docx.lnk)

**-Top-down stepwise refinement:** Techniques to form out algorithm is needed for well structured programs.Top-down technique means we start from the biggest problem(whole software) and we break it into small pieces. Bottom-up approach is the opposite. Starts from small parts and works its way to whole software. Stepwise refinement stands for these steps we take from whole software to small pieces or from small pieces to whole software. Lets give an example to a “top” which is a single statement that conveys the overall function of the program.

“Determine the class average for the quiz”

The top rarely conveys sufficient detail from which to write a Java program. So we now begin the refinement process. We divide the top into a series of smaller tasks and list these in order in which they will be performed. This results in the following first refinement. The example below only uses sequence structure.

Initialize variables

Input, sum and count the quiz grades

Calculate and print the class average

Each refinement as well as the top itself is a complete specification of the algorithm. Only the level of detail varies.

Many programs can be divided logically into three phases. An initialization phase that initializes variables. A processing phase that inputs data values and adjuts program variables accordingly. And a termination phase that calculates and outputs the final results.

In the second refinement we commit to specific variables. For our quiz example, the first pseudocode statement can be refined as:

Initialize total to zero

Initialize counter to zero

We initialize only the variables that needs it. Some variables will receive values so they dont need values at the start.

For the second statement of our first refinement, we use sentinel controlled repetition because we dont know how many grades will be entered.

Prompt the user to enter the first grade

Input the first grade(possibly the sentinel)

While the user has not yet entered the sentinel

Add this grade into the running total

Add one to the grade counter

Prompt the user to enter the next grade

Input the next grade(possibly the sentinel)

In pseudocode we do not use braces around the statements that form the body of the while structure. Instead we indent.

The last statement in our first refinement can be refined again as:

If the counter is not equal to zero

Set the average to the total divided by the counter

Print the average

“Else”

“Print “No grades were entered”

Blank lines seperate the algorithms into their phases and set off control statements; the indentation emphasizes the bodies of the control statements.

Terminate the top-down stepwise refinement process when you have specified the pseudocode algorithm in sufficent detail for you to convert the pseudocode to Java. Normally implementing the Java program is then straightforward.

**-Modular programming:** When you develop software, you shouldnt write a single block. Divide your software into blocks(for example using functions in same file), and different files.

The use of small self contained units make the underlying logic eaiser to devise and to understand for both the developer and the user. Development is facilitated because each module can be perfected in isolation. In fact, for large projects, different programmers can work on individual parts. Modular design also increases the ease with which a program can be debugged and tested because errors can be more easily isolated. Finally, program maintanance and modification are facilitated.

**-Speed of a program:** Asidefrom speed of the computer, what also determines the speed of the program you run is the amount of executions your program does. For example a worst case scenario is the situation when you have to do the maximum possible executions to finish your program. Best case scenario is lowest possible amount of executions that program needs to finish. And there are different definitions for average speed of the program. So tiny differences could become huge if you run your algorithm many times.