Chapter 1

Machine Language	Assembly Language	High Level Language
- Natural - Defined by hardware design - 0 and 1 - Machine dependent	- English-like abbreviations languages - Assemblers: Translator programs to convert to machine language at computer speeds - > Clearer & easier to understand - Requires many statements - MOV AL, 88h	- To speed programming - Compiler: Convert into machine language - Java

- Java

- Object-Oriented Programming (OOP): made up of objects to perform actions
- Used to develop Java applet → executed in Web browser/ applet viewer
- Edit → Compile → Load → Verify & Execute
  i. Edit
     - With Integrated Development Environment (IDE)
     - File end with .java extension
     - File name same as Class name
  ii. Compile
  - Translate into byte codes
  - Java Platform (JDK 15) provides compiler
  - Command: ***javac* filename.*java***
  - Syntax: grammar rules (arrangement of words & punctuations)

  iii. Load
  - Class loader take ***.class*** file & translate to memory
  - Java applications loaded into memory & executed using Java Interpreter/Virtual Machine/ run-time system

  iv. Verify
  - Byte code verifier ensures byte codes do not violate security requirements

  v. Execute
  - Interpret program 1 byte code at a time
  - Command (java file): ***java* filename**
  - Java Achieve (JAR) file created for a java application that consists multiple java files
  - Command (jar file): ***java -jar* filename.*jar***

- Problem Solving

- Before writing program

i. Have a clear understanding of problem

ii. Have a carefully planned approach to solve

  - Any computing problem can be solved by executing a series of actions in a **specific order**.
  - **Algorithm:** A procedure for solving a problem in terms of the actions to be executed and the order in which actions are to be executed

- General

i. Understand problem. Solve manually with few examples.

ii. Devise an algorithm to solve

iii. Write program using programming syntax & compile program

iv. Correct syntax error after compilation. Save program & compile again.

v. Execute program if error free

vi. Test results against expected output.
If results not correct, modify algorithm, rewrite & recompile program.

- Terms

  i. **Syntax** – A set of rules, principles, and processes that govern the **structure of statement** in a programming language.

  ii. **Semantic** – Describe the **meaning** of the things written while following the syntax rules of the language. Semantic describes the things happen when a program is executed.

  iii. **Debugging** – A process of **eliminating mistakes** in the program. A mistake in a program is called a bug.

- Common bugs/errors

i. **Syntax Error:** A grammatical mistake in the program. A mistake in the arrangement of words and punctuations.

ii. **Logic Error:** A mistake in the underlying algorithm or semantic error.

iii. **Run-time Error:** An error that happen when the program is executed

- Input Process Output

- Pseudocode: informal high-level description of operating principle of computer program or algorithm

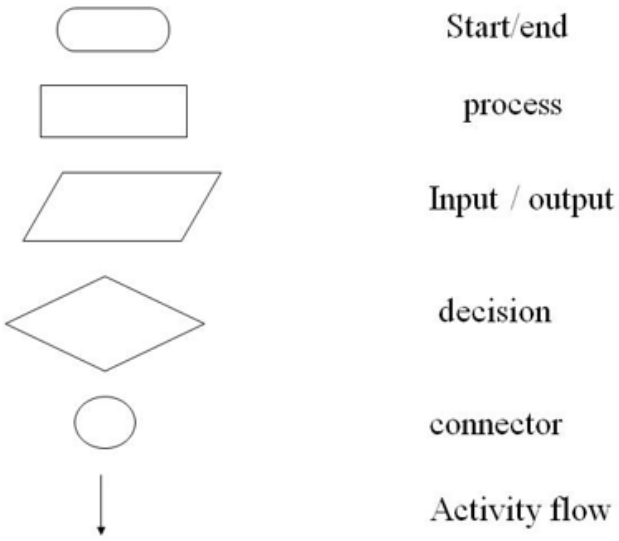
- A numbered list of instructions to perform some task

i. 1 line 1 statement 1 action. Write only 1 statement per line. Each statement express just 1 action.

ii. Indent: statement fall inside selection/loop

- Flow Chart: Diagram represents algorithm/process

- Shows steps as various kinds boxes & order by connecting with arrows
- Used in analyzing, designing, documenting or managing a process/ program
- NOTATION



- Sample Java Program
- Consist at least 1 class definition {}
- Contains method main, always static
- ***Void***: method perform task & will not return any info when task completed
- Program >> Classes>> Methods >> Statements
- Case sensitive
- Each statements must end with ;

CHAPTER 2

- Variable: a storage location in memory that has a type, name & contents
- Declared: type & name (identifier: letters + digits, underscore; X spaces/reserved words)
- Primitive types

Java Primitive Data Types

Type	Values	Default	Size	Range
byte	signed integers	0	8 bits	-128 to 127
short	signed integers	0	16 bits	-32768 to 32767
int	signed integers	0	32 bits	-2147483648 to 2147483647
long	signed integers	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	+/-1.4E-45 to +/-3.4028235E+38, +/-infinity, +/-0, NAN
double	IEEE 754 floating point	0.0	64 bits	+/-4.9E-324 to +/-1.79769313486231570E+308, +/-infinity, +/-0, NaN
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
boolean	true, false	false	1 bit used in 32 bit integer	NA

- Constant: ***final*** **type name = value;**

- Operator: Special symbols perform specific operations on >=1 operands (part of a computer instruction which specifies what data is to be manipulated/operated on)
- Assignment (=): Change value of variable
- Arithmetic
  i. + (addition)
  ii. – (subtraction)
  iii. * (multiplication)
  iv. / (division)
  v. % (modulo or remainder)
- Parentheses (): controls order
- Postfix: use current value of num, then increment/decrement by 1 for next statement
  i. number++;
  ii. number--;
- Unary: Increment/decrement num by 1 then use the value
  i. ++number;
  ii. –number;
- Other: +=, -=, *= /=, %=
  number += a : number = number+a

Operator Precedence

Operators	Precedence
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

- Type Casting
- (type) varName

- String
  String strName = ***"***value***"***;
- ***+*** concatenate

- Console Input

Import java.util.Scanner;
Scanner keyboard = ***new Scanner(System.in);***
varName = keyboard***.nextInt();***

- nextInt(), nextLong(), nextDouble(), next(): 1word, nextLine(): entire line

- Console Output

System.out.println : Output cursor to beginning of next line
System.out.print : X position output cursor at beginning of next line
System.out.printf : specific format ("%6.2f", varName >> Display 6 spaces with 2 d.p.)

i. \n : newline character
ii. \t : horizontal tab
iii. \\ : display backslash
iv. \" : display double quote

- Comment
- To help other programmers understand program; Not executed
  //Single Line Comment

  /*
  Multiple line
  comments
  ****/***

  /**
  This method display a line of text on the screen (documentation describe functionalities of each Java class & methods)
  ****/***

- To run Javadoc on a package:
  Javadoc -d documentationDirectory **PackageName**

- Random Number
  Import java.util.Random;
  Random name = ***new Random();***
  Num = name***.nextInt()***;
- .nextInt(100) : random value from 0 to 99

CHAPTER 3: FLOW OF CONTROL (SELECTION)

I. Sequence Flow: Statement executed one after other in order
II. Selection Flow: Chooses among alternative courses of action
III. Repetition Flow: Specifies an action is to be repeated while some condition remains true

- Relational Operator: Tests relationship between 2 values

==	Equal
!=	Not Equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal

- Logical Operator: Merging multiple constraints/ condition → Create complex Boolean expression

Operator	Description	Examples
&&	AND (true && true is true, others false)	a==b && c==d
||	OR (false && false is false, others true)	c!=d || a<b
!	NOT (!false is true. !true is false)	!(x>y)

- (Multiway) If-else

```
if ( condition1 ){
   statement1;
}
else if{
   statement2;
}
else{
   statement3;
}
```

- Switch

```
switch( variable ){
case value1:
   statement1;
   break;
case value2:
    statement2;
   break;
default:
   statement3;
}
```

- Ternary Operator

condition1***?*** statement1 ***:*** statement2

- True execute 1, false execute 2

- String Comparison
- Equality
  i. s1***.equals***(s2)

ii. s1***.equalsIgnoreCase***(s2)

- Alphabetical/Lexicographic order

i. s1.***compareTo***(s2)

ii. s1.***compareToIgnoreCase***(s2)

s1 before s2 → -ve || s2 before s1 → +ve || s1==s2 →0

CHAPTER 3: FLOW OF CONTROL (REPETITION)

I. Count-controlled loop: Executed statements for fixed number of times

II. Sentinel-controlled loop: Executed statements repeatedly until sentinel encountered

- While: Executes repeatedly, condition controls how often loop executed

while(condition ***){***
  statements;
}

- Do-while: Executes body at least once & perform condition check after

do {
  statements;
} while(condition ***);***

- For: Count-controlled loops; Step through some int var in equal increments/decrements

for (initialization; condition; update){
  statements;
}

- ***break;*** : ends nearest enclosing loop statement
- ***continue;*** : skip remaining statement (ends current loop body iteration of nearest enclosing loop statement) & proceeds with next iteration

- label: label a loop statement, used by the break & continue statement
  stop: { //label statement
    …
      break stop; //break the stop label statement
  }

CHAPTER 5: ARRAY

- Sequence of same type values
- Data structure to process a collection of same type data
- A group of contiguous memory locations that all have the same name & type
- Ordered list of values, each with numeric index (subscript)
- Array size N indexed from 0 to N-1
- A reference data type
- ***new*** operator to construct

Declaration	type***[]*** name ***= new*** type***[length];*** - Default: 0(numeric), false(Boolean), null(references/ string)
Initialization	type***[]*** name = ***{***value1, value2***};***
Length	name***.length***
To access	name***[indexNo.]***
To display element	***for(***int i=0; i<name.length; i++***){*** System.out.println(name***[i]***); ***}*** ***for(***int value ***:*** name***){*** System.out.println(***value***); ***}*** //can't modify value
Multidimensional (Initialization)	type ***[][]*** name = ***new*** type ***[***no.OfRows***][***no.OfColumns***]*** type ***[][]*** name = ***{ { , } , { , } }***

Ragged Arrays (diff rows diff columns)	type ***[][]*** name = ***new*** type [3][];
	t[0] = ***new*** type[3];
	t[1] = ***new*** type[10];
	t[2] = ***new*** type[5];

Bubble Sort

- Uses nested loop to make several passes, each pass compares successive pairs
- If pair increasing order, leaves value as they are; If pair decreasing order, swaps values in array

```
//control number of passes

for( int pass = 1; pass<b.length; pass++){
   //control number of comparison
   for( int i = 0; i<b.length-1; i++){
       if (b[i] = b[i+1]){
          int hold = b[i];
          b[i] = b[i+1];
         b[i+1] = hold;
}
```

- Linear Search: Small/unsorted arrays

```
for (int cnt = 0; cnt<name.length; cnt++){
    if( name[cnt] == searchKey ){
        return cnt;
return -1; //key not found

}
```

Binary Search

CHAPTER 6: METHODS

- Modules = Methods = Classes : Allow programmer to modularize program
- Local variable: a variable declared within a method
- Call-by-value method invocation: When invoke a method, a copy of the value of each actual parameter is passed to the method; Any changes to the copy inside method have no effect on actual parameter
- Reference type(Object & array): Any changes to instance variable will have effect on actual parameter; constant ***null*** indicate no real value
- Comment types
  i. Precondition: states what is assumed to be true when method is called
  ii. Postcondition: describes effect of method call, describes value returned by method

- Define

accessSpecifier returnType methodName (parameterType parameterName, …) {}

- accessSpecifier: public; void: return nothing

- Static Method: do not require an calling object
  public static returnType methodName(parameterType parameter, …)
- To invoke in same class: ***methodName***
- To invoke in different class: ***className.methodName***

LAB

- Generate random integer, range is 10 to 50 (include 50)

```
int x = random.nextInt(41)+10;
```

- random.nextInt(n) : 0 <= x < n

- random.nextInt(n)+1 : 1 <= x <=6

```
roll = (int) (Math.random() * 6 + 1);
```

- Convert seconds to hr min sec

```
hr = inputSec/3600;

min = (inputSec % 3600) / 60;

sec = inputSec % 60;
```

- To sum all digits

```
 while(num>0){
                digit = num%10; //this gets the last digit
                sum = sum + digit;
                num = num/10; //this removes the last digit
                }
```

- Don't use import java.util.* because the file will become very large (be specific)

- Determine point outside/inside circle

```
double h = x*x;
 double v = y*y;
 double l = Math.sqrt(h + v);
 if(l > radius){ //outside
```

<LAB 4>

- **Factors** of an integer

```
for (int j=1; j<=integer; j++){
       if (integer % j == 0) {
               System.out.print(j + ", "); }}
```

- 1 + (1+2) + (1+2+3) + … + (1+2+3+…+n)

```
for (int x = 1; x<=num; x++){
```

```
            int result2 = 0;
            for(int y=1; y<=x; y++){
            result2 += y;
            }
            result1 += result2;
       }
```

Or

```
for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                sum += j;
            }
        }
```

- Leap Year

```
(year % 400 == 0) || (year % 100 != 0) && (year % 4 == 0)
```

- Dice game

```
boolean isP1Turn = true;
        while(Math.max(p1Score,p2Score)<=100){
            roll = (int)(Math.random()*(6)+1);

//roll 6 get 6 points and can roll again
            if(roll == 6){
                bonus = (int)(Math.random()*(6)+1);}
//To store marks for each players
            if(isP1Turn){ p1Score += roll + bonus; } else {p2Score += roll + bonus;}
//To take turns
            isP1Turn = !isP1Turn; }
```

- Math.random() function returns **a floating-point, pseudo-random number that's greater than or equal to 0 and less than 1**,

- Calculate number of digits

```
int counter = (int) Math.log10(randomNum); //result= counter+1
```

- System.out.printf

%.2f : 2 decimal place

%23s or %23.2f or %23.2s or %23f: 23 spaces before print

- Print n prime numbers

```
while (ni < n) {

    boolean isPrime = true;

//Check Prime
    for (int j = 2; j < thisNum; j++) {

        if (thisNum % j == 0) {
            isPrime = false;
            break;
        }
    }

     if(isPrime) { //print
          ni++;
      }

       thisNum++; //Initial = 2
```

- To arrange array in reverse order

```
int[] revArr = new int[array.length];

        for(int j = 0; j < array.length; j++){
            revArr[array.length - 1 - j] = array[j];
        }
```

```
static void reverse(int[] array) {

    int n = array.length;

    System.out.println("Array in descending order");
#
    Arrays.sort(array);

    for (int j = 0; j < n / 2; j++) {
        int temp = array[j];
        array[j] = array[(n - 1) - j];
        array[(n - 1) - j] = temp;
    }
#
    System.out.println(Arrays.toString(array));
}
```

- Linear Search

```
static void linearSearch(int[] array, int n) {

    int loopCount = 0;
```

```
#
    for (int i: array) {
        if (i == n) { break; }
        loopCount++;
    }
#
    if (loopCount < array.length) {
        System.out.printf("%d found\nLinear Search - %d
loop(s)\n", n, loopCount);
    }else{
        System.out.println("Not found");
    }
}
```

- Binary Search

```
static void binarySearch(int[] array, int n) {
    int low = 0, high = array.length-1, mid = 0, loop=0;
    while (low < high){
        mid = (low + high)/2;
        if( n == array[mid]){System.out.println(mid);}
        else if( n > array[mid]){low = mid+1; } //n is on right
side
        else {high = mid -1;} //n is on left side
    loop++;
    }
    if(mid == n){
    System.out.println("Binary Search - "+loop+" loop(s)");}
    else{System.out.println("Not found");}
```

LAB 6

- Print 20 Triangular Number

```
for (int j = 1; j <= 20; j++) {
    curTriangleNumber += j;

    System.out.println(curTriangleNumber);
}
```

- Reverse string

```
static void reverseStr(String str){
        char[] arr = str.toCharArray();
        for(int i = arr.length-1; i>=0; i--){
            System.out.print(arr[i]);
        }
    }
```

- Euclidean Algorithm (GCD: Greatest Common Divisor)

```
// How GCD(200,625) autochange to GCD(625, 200)? Refer IDE to ask
public static int GCD(int a, int b){
        if (b == 0) {return a;}
        else {return GCD(b, a%b);}
    };
Or
public static void GCD(int a, int b){
        //if a>b, a=x , if a<b, b=x

        int x, y;
        if (a>b){
               x = a; y = b;
        } else {
               x = b; y = a;
        }
        // x is always greater than y

        int r1 = y, r2 = x%y;
        while (r2 != 0){
            int hold = r2;
            r2 = r1%r2;
            r1 = hold;
        }
        System.out.println(r1);
    }
```

- Palindromic Prime & Emirp

- Palindromic Prime: Palindromice + Prime

```
public static boolean isPalinPrime(int arg) {
        return isPalin(arg) && isPrime(arg);
        }
```

- Emirp: Prime(ori) + Prime(reverse) + !PalinPrime

```
public static boolean isEmirp(int arg) {
        int reverse = 0;
        int argCopy = arg;
```

```
        while(argCopy > 0){
        reverse = reverse * 10 + argCopy % 10;
        argCopy = argCopy / 10;
        }
        if (Integer.toString(arg).length() !=
Integer.toString(reverse).length()){
        return false;
        }
        //System.out.printf("%d %d\n", arg, reverse);
        return isPrime(arg) && isPrime(reverse)
&& !isPalinPrime(arg);
        }
```

- Palindromic: aba → reflect → aba

```
public static boolean isPalin(int arg) {
        int argCopy = arg;
        int reverse = 0;

        while(arg > 0){
        reverse = reverse * 10 + arg % 10;
        arg /= 10;
        }

        return (argCopy == reverse);
        }
```

- Prime

```
public static boolean isPrime(int arg) {
        for (int i = 2 ; i <= Math.sqrt(arg) ; i++){
        if (arg % i == 0)
        return false;
        }
        return true;
        }
```

- Take char as input

```
char c = input.next().charAt(0);
```

- To print shape with repeated character
- Multiprint to print spaces ‘ ’

```
static void multiPrint(int n, char c){

        for (int i=1;i<n;i++)

            System.out.print(c);

    }
```

- Multiprint with spaces to print char for shape

```
static void multiPrintWithSpaces(int n, char c){
```

```
        for (int i=1;i<n;i++)
            System.out.print(c + " ");
    }
```

- Triangle

```
static void triangle(int n, char c){
        for(int i=1;i<=n;i++){      // loop to iterate for the given number of
rows
            multiPrint(n-i+1, ' ');
            multiPrintWithSpaces(i, c);
            System.out.println();
        }
    }
```

- Diamond

```
static void diamond(int n, char c){
//Triangle: for upper
// + another for loop: for lower
for (int i=1; i<n;i++){
            multiPrint(i + 1, ' ');
            multiPrintWithSpaces(n-i, c);
            System.out.println();
        }
```

- Generate non-duplicate random number – ArrayList

```
ArrayList<Integer> randomNum = new ArrayList<Integer>(10);

for (int i = 0; i < 10; i++) {
    int integer = (int) (Math.random() * 21);

    while (randomNum.contains(integer)) {
        integer = (int) (Math.random() * 21);
    }

    System.out.print(integer);
    System.out.print(i == 9 ? "" : ",");
    randomNum.add(integer);
}
```

- Rotate array 90 degrees

1 5 7

3 6 9

5 3 8

5 3 1

3 6 5

8 9 7

```
int[][] matrix = {{1,5,7},{3,6,9},{5,3,8}};
int[][] matrix2 = new int [matrix.length][matrix[0].length];

for(int i = 0, j = matrix.length-1; i < matrix.length; i++, j--) {
        matrix2[0][j] = matrix[i][0];
        matrix2[1][j] = matrix[i][1];
        matrix2[2][j] = matrix[i][2];
    }
```

- To print multidimensional array

```
for(int i = 0; i <= matrix.length-1; i++){
    for(int j = 0; j <= matrix[0].length-1; j++){
        System.out.print(matrix[i][j] + " ");
        if(j>0 && j%2 == 0){System.out.print("\n");}
    }
}
```

- To factorial

```
public int factorial(int i) {
    if (i == 0)
        return 1;
    return i * factorial(i - 1);
}
```

- Pascal Triangle
- Equilateral

```
// Print Pascal's Triangle in Java
import java.io.*;

class GFG {
    public int factorial(int i)
    {
        if (i == 0)
            return 1;
        return i * factorial(i - 1);
    }
    public static void main(String[] args)
    {
        int n = 4, i, j;
        GFG g = new GFG();
        for (i = 0; i <= n; i++) {
            for (j = 0; j <= n - i; j++) {

                // for left spacing
                System.out.print(" ");
            }
            for (j = 0; j <= i; j++) {

                // nCr formula
                System.out.print(
                        " "
                                + g.factorial(i)
                                / (g.factorial(i - j)
                                * g.factorial(j)));
```

```
                }

                // for newline
                System.out.println();
            }
        }
    }
```

- Square (0 when no number)

```
public int factorial(int i) {
    if (i == 0)
        return 1;
    return i * factorial(i - 1);
}

public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of rows of Pascal Triangle
to generate: ");
    int n = keyboard.nextInt();

    int i, j, k;
    Main g = new Main();
    for (i = 0; i < n; i++) {

        for (j = 0; j <= i; j++) {
            // nCr formula
            System.out.print(
g.factorial(i) / (g.factorial(i - j) * g.factorial(j)));
        }
        for (k = n - 1 - i ; k > 0; k--) {
            System.out.print(0);
        }

        // for newline
        System.out.println();
    }
}
```

➢ MUST REVISE

- L4Q4: Display Calendar
- L5Q5: Reverse array, Linear & Binary Search
- L5Q2: Generate non-duplicate integer (ArrayList)
- L5Q4: Rotate array 90 degrees
- L5Q6: Pascal Triangle
- L6Q2: Print n times c char to get triangle & diamond
- L6Q6: Palindromic Prime & Emirp

- Check grammar for programming: https://kodezi.com/

- How to check if array contains a value alr then generate another without arrayList?

```java
import java.util.Arrays;

public class Main {
    public static void main(String[] args) {
        int[] intArr = new int [10];
        for (int i = 0; i < 10; i++) {
            int num = (int) (Math.random() * 21);
            if (check(intArr, num)){
                intArr[i] = num;
            }
            }

        System.out.print(Arrays.toString(intArr));
    }

    static boolean check(int[] arr, int key) {
        for (int j = 0; j < arr.length; j++) {
            for (int k = arr.length - 1; k >= 0; k--) {
                if (arr[j] == arr[k]) {
                    arr[j] = (int) (Math.random() * 21);
                    check(arr, key);
                }
                return true;
            }
        }
        return false;
    }
}
```

Tutorial

- QTA
- T3Q3

Write the output for the following statements when x=9 and y=10.

a)
```
if (x < 10)
if (y > 10)
System.out.println("*****");
else
System.out.println("#####");
System.out.println("$$$$$");
```

```
#####
$$$$$
```

b)
```
if (x < 10) {
if (y < 10)
System.out.println("*****");
else{
System.out.println("#####");
System.out.println("$$$$$");
}}
```

```
#####
$$$$$
```

c)
```
if (x < 10) {
if (y < 10)
System.out.println("*****");
System.out.println("#####");
}
else {
System.out.println("$$$$$");
}
```

```
#####
```

d)
```
if (x > 10) {
if (y > 10) {
System.out.println("*****");
System.out.println("#####"); }
else
System.out.println("$$$$$");
}
```

```
No Output.
```

Write the statements that generate 1 random integer within 0 – 255. Convert the number to binary and store the bit into an 8 bit array. Then, display the binary number.

```
char[] bits = new char[8];

int randNum = (int) (Math.random() * 255);
String binary = Integer.toBinaryString(randNum);

binary = String.format("%8s", binary).replaceAll(" ", "0");

for (int i = 0; i < 8; i++) {
    bits[i] = binary.charAt(i);
    System.out.print(bits[i]);
}
```