Chapter - 1 : Boolean Algebra

REVISION NOTES

Propositional Logic

- A proposition is a declarative sentence that is either true or false but not both. Statements are also called statements.

  For example,

  1. *The Sun sets in the East direction.*

  (It is a proposition with false value)

- **Logical Connective (Operators) :** For combining two or more statements we use special symbols, called logical connectives. There are several logical connective as follows :
  - **Negation (NOT) :** If S is a statement or proposition, then NOT (S) or (~S) is negation of proposition S.

    For example, S : Ram eats the mango.

    Not (S) : Ram doesn't eat the mango.
  - **Conjunction (AND) :** If R and S are two propositions, then R and S are connected by 'AND' or '.' symbol.

    For Example,

    R : He is a player.

    S : He plays football very well.

    R ∧ S : He is a player and he plays football very well.
  - **Disjunction (OR) :** It is represented by 'OR' or '∨' symbol. Let R and S two proposition. Then R V S is read as R or S.

    For Example,

    R : She is a player.

    S : She is a dancer.

    R V **S** : She is a player or dancer.
  - **Conditional (⇒) :** A → B is called a conditional statement. It is also read as 'A implies B'. A ⇒B means A'+B

    For Example,

    A : You will do hard work.

    B : You will get success.

    A → B : If you will do hard work, then you will get success.
  - **Biconditional (⇔) :** A ↔ B is known as biconditional statement or proposition. ' ↔ ' or '⇔' shows equivalence velation between two propositions A ↔ B represents AB+A'B'.

    For example :

    A : She is smart.

    B : She is beautiful.

    A → B : She is smart as well as beautiful.
- **Truth Table :** A table used to determine all possible truth values regarding a proportional value. A single proposition contains two possible values as 'True' for correct and 'False' for incorrect value. True value (denoted by 1) or false value (denoted by O) are called 'truth values'.
- We can determine the truth values of the various connections as follows :

p	q	$\sim p$	$\sim q$	$p \wedge q$	$p \vee q$	$p \Rightarrow q$	$p \Leftrightarrow q$
0	0	1	1	0	0	1	1
0	1	1	0	0	1	1	0
1	0	0	1	0	1	0	0
1	1	0	0	1	1	1	1

Table : Truth Table for connectives

➢ **Other important Terminology**

- **Well Formed Formula (WFF) :** It refers a simple proposition and compound proposition. A WFF can be determined with the help of following properties :

  - If P is a propositional variable, then it is a WFF.

  - If P is wff, then . p is a wff.

  - If P and a are wff, then $(p \wedge q)$, $(p \vee q)$, $p \Rightarrow q$, $p \Leftrightarrow q$ are wff.

- **Tautology :** A compound proposition that is always true.
- **Contradiction :** Negation of tautology is termed as contradiction.
- **Contingency :** A compound proposition that is neither tautology nor contradiction.

  For Example :

  $p \vee \neg p$, is a tautology, $p \wedge \neg p$, is a contradiction and $p \vee p$, or $p \wedge p$, is a contingency.

  Argument : Sequence of statements (premises). An argument is valid if it is a tautology, otherwise it is fallacy.

  Syllogism : A Logical process of drawing conclusions from given premises.

  Converse : The converse of $p \rightarrow q$ *is* $q \rightarrow p$.

  Inverse : The inverse of $p \rightarrow q$ *is* $\neg p \rightarrow \neg q$,

  Contra positive : The contra positive of $p \rightarrow q$ *is* $\neg q \rightarrow \neg p$.

➢ **Some basic Equivalence Laws of Propositions**

- **Commutative Law :**

  - $p \vee q \Leftrightarrow q \vee p$ and $p \wedge q \Leftrightarrow q \wedge p$

- **Distributive Law :** $p \wedge (q \vee r) \Leftrightarrow (p \wedge q) \vee (p \wedge r)$

  and $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$

- **De Morgan's Law :** $\neg (p \wedge q) \Longleftrightarrow \neg p \vee \neg q$

  and $(p \vee q) \Longleftrightarrow \neg p \wedge \neg q$

- **Involution Law :** $\neg (\neg p) \Leftrightarrow p$
- **Absorption Law :** $p \vee (p \wedge q) \Leftrightarrow p$ *and* $p \wedge (p \vee q) \Leftrightarrow p$
- $p \Rightarrow q = \neg p \vee q$
- $p \wedge T = p$ *and* $p \vee F = p$; $p \vee T = T$ *and* $p \wedge F = F$.
- $p \Leftrightarrow p = (p \rightarrow q) \wedge (q \rightarrow p)$

Theorems and Postulates of Boolean Algebra

➢ Boolean Algebra is an algebra given by George Boolean which used for performing logical operations on 0 and 1 as binary variables.

➢ **Laws of Boolean Algebra**

S. No.	Theorem /Laws	Expression	Dual
1.	Identity law	A + 0 = A	A · 1 = A
2.	Complementary law	A + A´ = 1	A · A´ = 0
3.	Commutative law	A + B = B + A	A · B = B · A
4.	Associative law	A + (B + C) = (A + B) + C	A · (B · C) = (A · B) · C
5.	Distributive law	A + (B · C) = (A + B) · (A + C)	A · (B + C) = (A · B) + (A · C)
6.	Idempotent law	A + A = A	A · A = A
7.	Absorption law	A + (A · B) = A	A · (A + B) = A
8.	DeMorgan's law	(A + B)´ = A´ · B´	(A · B)´ = A´ + B´
9.	Third distributive law	A + (A´ · B) = A + B	

➢ **Principle of Duality :** It states that a boolean relation can be obtained by changing OR by AND and Vice-versa.

- For example, (A) dual of X + Y · Z is X · (Y + Z).
- (B) dual of O is always 1 and Vice-versa.

Forms of Representing Boolean Expressions

- A boolean expression results either true or false *i.e.,* a boolean value. It is also called as comparing expression, conditional expression, and relational expression. In Boolean expression two important terms are—Minterms and Maxterms.
- A boolean expression is an expression that results either true or false boolean value. In boolean expression, a single variable or its complement is called literal.
- **Minterms** two or more literals joined by AND operator are known as minterms. For example, xy, $x'y$, $x'y'$ and xy' are minterms for two literals and x and y.
- **Maxterm** two or more literals jointed by or operator are known as maxterms. For examples, $x + y$, $x' + y'$, $x' + y$ and $x + y'$ are maxterms for two variables x and y.

x	y	z	Maxterm ()		Minterm	
			Term	Symbol	Term	Symbol
0	0	0	$x+y+z$	m_0	$x'y'z'$	M_0
0	0	1	$x+y+z'$	m_1	$x'y'z$	M_1
0	1	1	$x+y'+z$	m_2	$x'yz'$	M_2
0	1	1	$x+y'+z'$	m_3	$x'yz$	M_3
1	0	0	$x'+y+z$	m_4	$xy'z'$	M_4
1	0	1	$x'+y+z'$	m_5	$xy'z$	M_5
1	1	0	$x'+y'+z$	m_6	xyz'	M_6
1	1	1	$x'+y'+z'$	m_7	xyz	M_7

- **Canonical Expression :** A boolean expression composed entirely of either minterms or maxterms.
- **Sum-of-products (SOP) form:** A boolean expression represented purely as sum of minterms is said to be in SOP form.
- **Product of sum (POS) form:** A boolean expression represented purely as product of maxterms is said to be in canonical POS form.
- **Canonical SOP and POS Forms:** When each term of a logic expression contains all variables, It's said to be in the canonical form.

Conversion from SOP Expression to Canonical SOP Expression:

The following procedure is used to convert SOP expression into canonical SOP expression.

1. Check the missing variable in each term.
2. Perform AND operation with (*e.g.* $X + \bar{X}$) terms, which have missing variables.
3. Expand all terms and remove the duplicate terms such that $X + X = X$ or $X + X + X = X$.
4. The produced SOP expression is the canonical SOP expression.

Conversion from POS Expression to Canonical POS Expression:

The following procedure is used to convert POS expression into canonical POS expression.

1. Check the missing variables in each term.
2. Perform Or operation with (*e.g.* $X \cdot \bar{X}$) missing variables.
3. Expand all terms and remove the duplicate terms such that $X \cdot X = X$ or $X \cdot X \cdot X = X$.
4. The produced POS expression is the canonical POS expression.

Conversion from SOP Expression to Canonical POS Expression:

The following procedure is used to convert the SOP expression into canonical POS expression—

1. Check the given SOP expression is canonical SOP expression or not, if not then convert it into canonical SOP expression. If it is already in canonical SOP expression then continue to step 2.
2. Use the duality theorem to convert SOP into canonical POS such that:
   (A) Change every OR sign (+) to AND sign (·).
   (B) Change every AND sign (·) to OR sign (+).
   (C) Change normal variable (A) into complement variable ($\bar{A}$) and vice versa.

3. The produced expression is the canonical POS expression.

Conversion from POS Expression into Canonical SOP Expression :

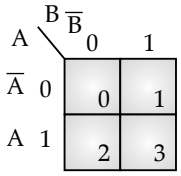
The following procedure is used to convert POS expression into canonical SOP expression:

1. Check the given POS expression is canonical POS expression or not, if not then convert it into canonical POS expression. If it is already in canonical POS expression then continue step 2.
2. Use the duality theorem to convert POS expression into canonical SOP expression such that:
   (A) Change every OR sign (+) to AND sign (•).
   (B) Change every AND sign (•) to OR sign (+).
   (C) Change normal variable (A) into complement variable ($\overline{A}$) and vice versa.
3. The produced expression is the canonical SOP expression.

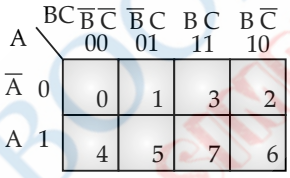
Karnaugh Map (K-map)

- **Karnaugh Map (K-map) Method :** It is a most popular method for simplification of boolean expressions of two variables, 3-variables, 4-variables and soon.

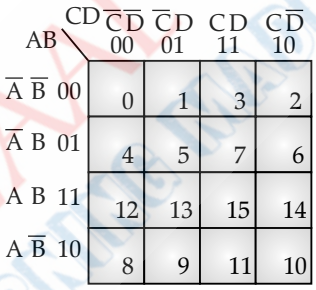
Representation of K-maps :



2-Variables K-map



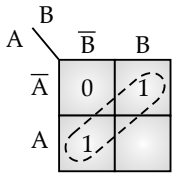
3-Variables K-map



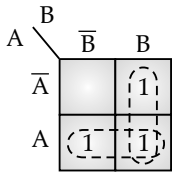
4-Variables K-map

- **Possible combinations $_2n$ cells for n-variables :**
  - **2-variables K-map (4 cells) :** $\overline{A}\,\overline{B}$, $\overline{A}\,B$, $A\,\overline{B}$, AB.
  - **3-variables K-map (8 cells) :** $\overline{A}\,\overline{B}\,\overline{C}$, $\overline{A}\,\overline{B}\,C$, $\overline{A}\,B\,\overline{C}$, $\overline{A}\,B\,C$, $A\,\overline{B}\,\overline{C}$, $\overline{B}\,C$, $A\,B\,\overline{C}$, ABC
  - **4-variables K-map (16 cells) :** take combination as shown in the K-map
- **K-map Simplification Rules :** K-map uses the following rules for the simplification of expressions by grouping expressions by together in the form of Octal, Quad and pair form.

  Some rules are as follows :
  - Groups may not include any cell containing a zero. It means we cannot make a pair of 0 and 1.
  - Groups may be vertical or horizontal, but not diagonal.

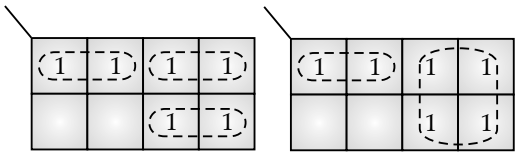


Incorrect Pattern



Correct Pattern

  - Each groups should be as large as possible *i.e.,* Octet
  - (8 adjacent 1's), Quad (4 adjacent 1's) and pair (2 adjacent 1's)

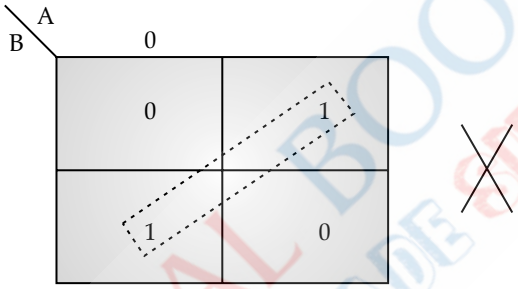


- Groups may be overlap as shown above in the correct pattern.
- In a K-map, first find out all possible octet group, then Qued-group and at the end make pairs. Then you will found the optimal solution or simplest form of given expression.

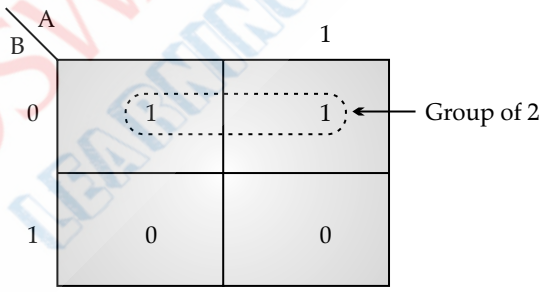
➢ **Karnaugh map or K-map** is a graphical display of the fundamental products in a truth table. It is a rectangle made up of certain number of squares, each square representing a maxterms or minterms.

➢ **Rules of Simplification :** The Karnuagh map uses the following rules for the simplification of expressions by grouping together adjacent cells containing ones.

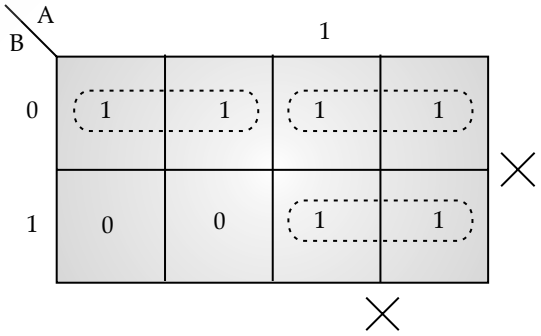
- Groups may not include any cell containing a zero.
- Groups may be vertical or horizontal, but not diagonal.



- Groups must contain 1, 2, 4, 8, or in general 2n cells.
- That is if n = 1, a group will contain two 1's since $2^1 = 2$.



- Each group should be as large as possible.



KNOW THE TERMS

- Logic means some rules which can be applied over an expression to find its optimal solution.
- Mathematical logic is used in designing the circuits in the computer system.
- Propositional logic is a declarative statement with two possible values either 'True (01)' or 'False (0)' but not both.

- AND ($\wedge$), OR ($\vee$), Not (—), conditional ($\rightarrow$) and biconditional ($\leftrightarrow$) are logical connectives. With the help of these connective, compound statements are created.
- If there are n-variables in the expressions, then it contains 2^n possible combination of their values.
- **Tautology :** A compound statement having all its possible values become true.
- **Contradiction :** A compound statement that is always false.
- **Contingency :** A compound statement that is neither true nor false.
- **Argument :** An argument is valid iff it is a tautology.
- **Syllogism :** Logical process of drawing conclusion from given propositions.
- **Basic Postulates :**
  - $A + 0 = 0 + A = A$ $\quad 0 + 1 = 1 ; a + 0 = 0$
  - $A.1 = 1 . A = A$ $\quad 0 . 1 = 0 ; 1 + 1 = 1$
  - $A + A = A; A.A = A$
  - $A + \overline{A} = 1; A.\overline{A} = 0$
  - $\overline{(\overline{A})} = A$
- **De Morgan's Theorem :** The complement of the sum of variables is equal to the product of their complement of their variables.
  - $\overline{(A+B)} = \overline{A} . \overline{B}$ or : $\overline{(A . B)} = \overline{A} + \overline{B}$
- **Sum-of Products (SOP) :** Sum of minterms is known as sum of products (SOP) form. Example, $xy' + x'y$ is a SOP form.
- **Product-of-Sum (POS) :** Product of maxterms is known as product of sums (POS) from. Example, $(x + y)\cdot(x' + y)$ is a POS form.
- Mini term is a product of all literals and maxterm is a sum of literals.
- **Canonical from :** If an expression is represented in its maxterms or minterms.
- **Cardinal form :** If an expression is represented in its decimal equivalent of its terms.

❑❑

Chapter - 2 : Computer Hardware

REVISION NOTES

Logic Gates

A logical gate a small circuit which is used to implement boolean function. It is also used to design logic circuits in computer hardware.

Every logic gate produces signals in the form of 0 and 1. If any gate acts as a open circuit. Output for the given inputs does not produced.

■ **Types of Logic Gates :**

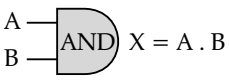
As we know that the logic gates represents electronic circuits in boolean algebra. There are two types of logic gates mainly :

(i) Fundamental or Elementary Logic Gates

(ii) Universal Gates

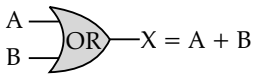
1. **Fundamental Logic Gates :** AND, Or and NOT gates are called fundamental logic gates.

   (i) **AND gate :** This logic gates contains at least two inputs and one output. It will give output when all inputs are enabled in the form of 1.



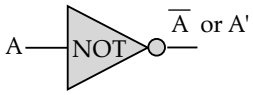
A	B	X = A.B
0	0	0
0	1	0
1	0	0
1	1	1

(ii) OR gate : This logic gate contain at least two inputs and single output. It enables or gives output iff any of one of the inputs in enabled.



A	B	X = A + B
0	0	0
0	1	1
1	0	1
1	1	1

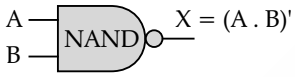
(iii) NOT Gate : It has only single input and single output. It acts as inverter.



A	$\overline{A}$
0	1
1	0

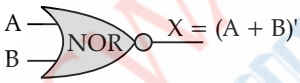
2. Universal Gates : NAND and NOR gates are known as universal gates. With the help of these gates, fundamental gates can be created.

(i) NAND Gate : It is a universal logic gate. It is the combination of NOT and AND gate. All possible outputs in NAND Gate is opposite of the outputs of AND gate.



A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(ii) NOR Gate : It is also a universal logic gate. It is combination of NOT gate and OR gate. All possible outputs in NOR gate is Opposite of the outputs of OR gate.

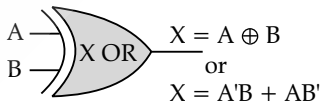


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

■ **Others Specific Logic Gates :**

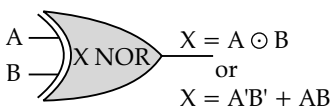
1. Exclusive OR (X-OR) Gate : It is a digital logic gate that gives a true (1) output when the number of true inputs is odd.

X-OR gates are used to implement binary addition in computer. The algebraic notation used to represent the XOR operation is $X = A \oplus B$



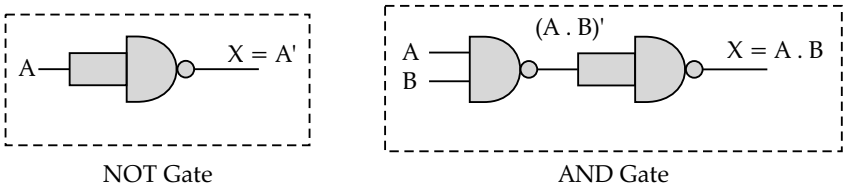
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

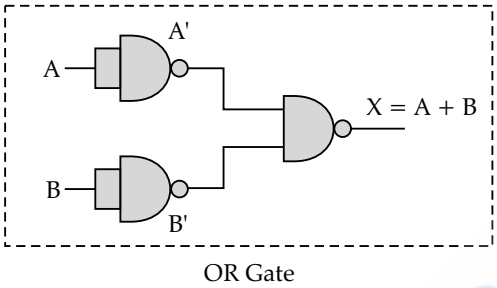
2. Exclusive-NOR Gate : It is combination of Not gate and XOR gate. It is the logical complement of the XOR gate. The algebraic notation used to represent the XNOR operation is $S = A \odot B$



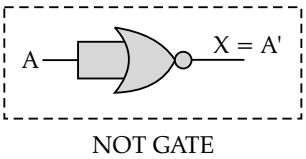
A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

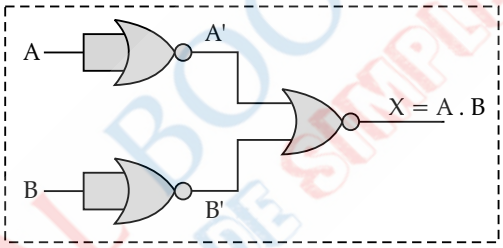
- **Realisation of Logic Gates using NAND Gate :**

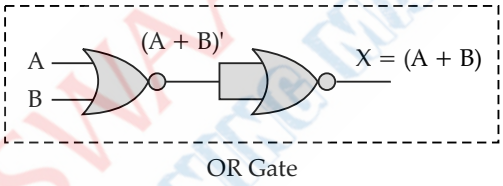




- **Realisation of Logic Gates using NOR Gate :**







Combination Circuits

Combinational logic circuit is a circuit in which we combine the different gates in the circuit. For Example, Half Adder, full Adder, encoder, decoder, multiplexes and demultiplexer.

Characteristics of combinational circuits :

- A combinational circuit can have an *n* number of inputs and *m* number of outputs.

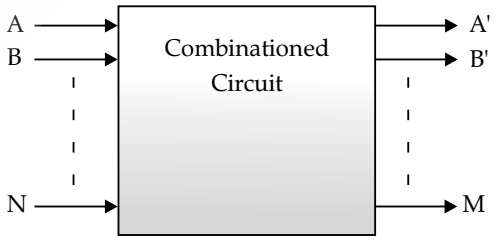
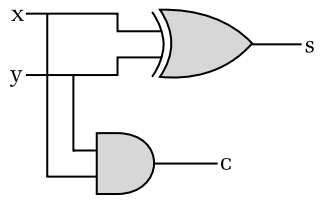
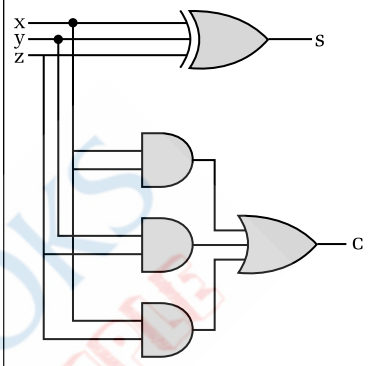
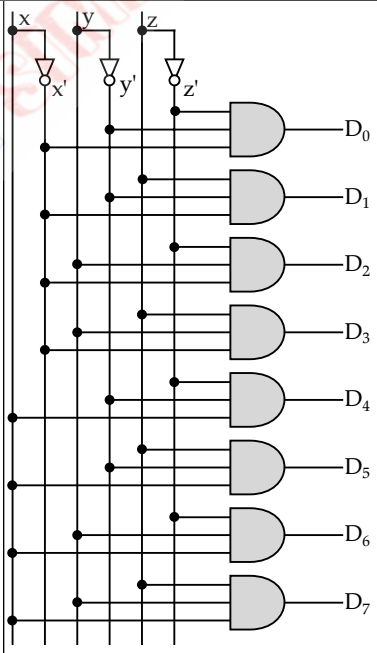
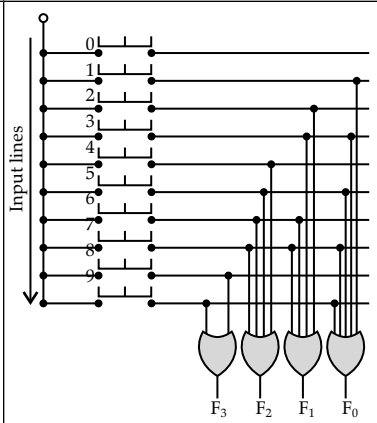


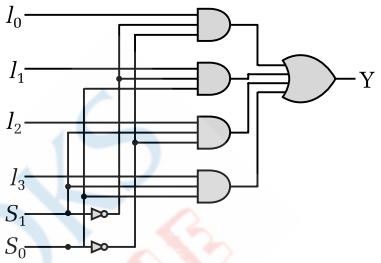
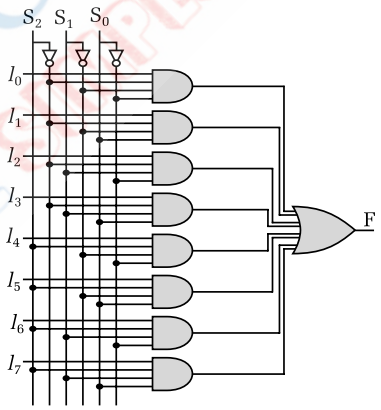
Fig. Block Diagram

- The output of combinational circuit at any instant of time depends only on the levels present at input points.
- They do not use any memory. The previous state of input doesn't have any effect on the present sate of the circuit.

A combinational circuit consists of related logic gates whose output at any time can be determined directly from the present combination of inputs irrespective of previous inputs.

(eg.) half adder, full adder, decoder, encoder, multiplexer, etc.

<table>
<tr><th>Circuit</th><th>Description</th><th>Expression</th><th>Truth Table</th><th>Logic diagram</th></tr>
<tr><td>Half Adder</td><td>Performs the addition of two bits</td><td>$S = x´y + xy´ = x \oplus y$
$C = xy$</td><td>
<table>
<tr><th>x</th><th>y</th><th>S</th><th>C</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>1</td></tr>
</table>
</td><td> 
</td></tr>
<tr><td>Full Adder</td><td>Performs the addition of three bits.</td><td>$S = x \oplus y \oplus z$
$C = xy + yz + xz$</td><td>
<table>
<tr><th>x</th><th>y</th><th>z</th><th>S</th><th>C</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
</table>
</td><td> 
</td></tr>
<tr><td>Decoder</td><td>Converts binary information from n input lines to a maximum of 2^n unique output lines.</td><td>2×4 Decoder
$m \le 2^n$
$n = 2$</td><td>
<table>
<tr><th>x</th><th>y</th><th>D_0</th><th>D_1</th><th>D_2</th><th>D_3</th></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
</table>
</td><td> 
</td></tr>
<tr><td>Encoder Decimal to Binary</td><td>2^n input lines
n output lines
$n = 4$</td><td>
<table>
<tr><th>Decimal number</th><th>F_3</th><th>F_2</th><th>F_1</th><th>F_0</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>2</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>3</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>4</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>5</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>6</td><td>0</td><td>1</td><td>1</td><td>0</td></tr>
<tr><td>7</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>8</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>9</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
</table>
$F_0 = \Sigma(1,3,5,7,9)$
$F_1 = \Sigma(2, 3, 6, 7)$
$F_2 = \Sigma(4, 5, 6, 7)$
$F_3 = \Sigma(8, 9)$
</td><td> 
</td></tr>
</table>

<table>
<tr><td>Multiplexer</td><td>Selects binary information from one of many inputs lines and directs it to a single output line.</td><td></td><td></td></tr>
<tr><td>4 × 1 MUX</td><td>4 input lines
2 selection lines</td><td>
<table>
<tr><th>S_1</th><th>S_0</th><th>Y</th></tr>
<tr><td>0</td><td>0</td><td>I_0</td></tr>
<tr><td>0</td><td>1</td><td>I_1</td></tr>
<tr><td>1</td><td>0</td><td>I_2</td></tr>
<tr><td>1</td><td>1</td><td>I_3</td></tr>
</table>
</td><td> 
</td></tr>
<tr><td>8 × 1 MUX</td><td>8 input lines (2^3)
3-selection lines
$I_0 = S_2'S_1'S_0'I_0'$</td><td>
<table>
<tr><th>S_2</th><th>S_1</th><th>S_0</th><th>Y</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>I_0</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>I_1</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>I_2</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>I_3</td></tr>
<tr><td>1</td><td>0</td><td>0</td><td>I_4</td></tr>
<tr><td>1</td><td>0</td><td>1</td><td>I_5</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>I_6</td></tr>
<tr><td>1</td><td>1</td><td>1</td><td>I_7</td></tr>
</table>
</td><td> 
</td></tr>
</table>

KNOW THE TERMS

- Elementary Logic gates are NOT, AND, OR gates
- Universal gates are NAND gates and NOR gate because with the help of there gates, fundamental/elementary gates can be created.
- **And gate :** Two inputs (at least) and one output is required for AND gate. If both inputs are true (1), then output will be true, otherwise false (O).
- **OR Gate :** At least two inputs and single output is required for OR gate. It is represented by '*t*' symbol in logical expressions when any input out of the given inputs in true (1), then output will be true (1), otherwise false (0).
- **NOT Gate :** It acts as inverter. If input is true (1), then output is false (0) and vice-versa. It is shown by '—' or ','.
- **NAND Gate :** Complement of AND gate values is termed as NAND gate. It is universal gate.
- **NOR Gate :** Complement of OR gate is termed as NOR gate.
- **X-OR Gate :** These are used to implement binary addition in computer. It is shown by '⊕' symbol.
- **X-NOR Gate :** It is the complement of XOR gate. It is represented by '⊙' symbol.
- **Half Adder :** It performs the addition of two bits. There are two inputs A and B and the two outputs are Sum (S) and Carry (C).
- **Full Adder :** A combinational circuit that performs the addition of three bits as two significant bits and one previous carry.
- **Decoder :** It is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- **Encoder :** It produces a reverse operation of decoder. Basically. It is a process of converting signals from one type to another type.

- **Multiplexer :** It selects binary information from one of many input lines and directs it to a single output line. Since. It selects the data, it is also termed as a data selects.

❑❑

Chapter - 3 : Programming in Java

REVISION NOTES

Fundamentals of Java

- Java is a high programming language originally developed by Sun Micro systems and released in 1995. Java runs on several platforms as windows, Unix and Mac OS.

  When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each others method.

  Object Oriented Programming (OOP) is a programming paradigm which deals with the concepts of object to create programme and software applications. The features of OOP are closely with the features of real-world like object, class, inheritance, abstraction, encapsulation and polymorphism.

  - **Objects :** Objects have states and behaviours. For example; a student has states or characteristics : name, age, group as well as behaviour such as student studies, playing, acting, etc.
  - **Class :** A class can be defined as a template or blueprint that describes the behaviour or state that the object of its type supports. An object is an instance of a class. A class is a way to bind the data and its associated functions together.
  - **Methods :** Method basically is behaviour. A class contains many behaviour. It is in methods where the logics are written, data is manipulated and all actions are executed.
  - **Inheritance :** In Java, classes can be derived from classes. If you require to create a new class and here is already a class that has some of the code you need, then it is possible to derive your new class from the already existing code. The feature is termed as inheritance. The existing class is called super class and the derived class is called the subclass.
  - **Data Abstraction :** An act of representing main features without having the background details or internal details. It deals with the outside view of an object as interface. For example, We all are performing several operations on ATM machine like withdrawals, depositing, mini-statement, etc. but we can't know internal details of ATM.
  - **Data Encapsulation :** is a process of wrapping of data and methods in a single unit is called encapsulation. It is achieved by class concept in Java. The insulation of the data from direct access by the program is called 'data hiding'.
  - **Polymorphism :** A process of representing one form in various forms is called 'Polymorphism'. For example, a single persons acts like customer in the shopping mall, student in a school, a son at home, a passenger in a bus/train. It means that one person in different-different behaviours.

- **Java Identifiers :**

  All Java components require names. Names used for classes, variables and methods are called identifiers. There are following points to remember for identifiers :

  - All identifier should begin with a letter (A to Z or a to z), $ (dollar sign), (__) underscore).
  - A key word cannot be used as an identifier.
  - Identifiers are case-sensitive.
  - Legal identifiers : age, $ salary, _value, _1_value.
  - Illegal identifiers : 123abc, _salary

- **Java Keywords :** The following lists show the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.

abstract	private	double	synchronized	new	float
byte	short	final	while	public	implements
class	switch	go to	catch	strictly	interface
do	throws	instance of	continue	this	package

extends	volatile	native	else	try	return
for	assert	protected	finally	char	super
import	case	static	if	default	throw
long	const	transient	int	enum	void

- **Constructors :** Every class has a constructor. The java compiler builds a default constructor for that class. Each item a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class have more than one constructor.

  Example :

```
Public class student
{
      Public student ( )
    {
      .....
    }
      Public student (string name)
    {
       //This constructor has one parameter
    }
}
```

- **Data Types :**

  There are two data types available in Java :

  - Primitive Data Types
  - Reference (Non-primitive) Data Types

- **Primitive Data Types :**

  Primitive data Types are defined by the language and named by a keyword. There are eight primitive data types supported by Java.

Data Type	Size	Range	Default Value
Byte	1 byte	– 128 to + 127	0
short	2 bytes	32768 to + 32767	0
int	4 bytes	-2^{31} to + (2^{31} – 1)	0
Long	8 bytes	-2^{63} to + (2^{63} – 1)	0L
Float	4 bytes	1.4e – 0.45 to 3.4e + 0.38	0.0F
double	8 bytes	4.9 e - 324 to 1.8e + 308	0.0d
boolean	boolean	only uses 1 bit	False
Char	2 bytes	0 to 65535	can store any character

- **Non Primitive Data Types :**

  It is a variable that can contain the reference or an address of dynamically created object. The non-primitive data types are arrays, classes and interfaces.

- **Java Literals :** A Literal is a source code representation of a fixed value. They are represented directly in the code without any computation. Literals can be assigned to any primitive type variable.

  Example : byte a = 23;
  Char P = 'S';

- **Variables :** A Variable provides us with named storage that our programs can manipulate. It is also termed as identifier that holds values, used in Java program. Example, Sum, Total marks, average etc. It may consists of alphabets, digit, specific symbols like (_), $.etc.

  data type variable [= value] [, variable [= value] .....];

  Example : int *a*, *b*, *c* //declares three integers a, b and c.
  int *a* = 10, *b* = 10; // Example of initialization.

```
byte A = 25;            // Initializes a byte type variable A.
double Pi = 3.14159; // declares and assigns a value of PI.
```

There are three types of variables used in Java

(i) Local Variables : These are declared in methods, constructors or blocks. These are implemented at stack level internally. There is no default value for local variable, that's why when local variable is declared, on initial value should be assigned to it before the first use.

Syntax : Variable Name

Example : Age //defined inside student Age () method

(ii) Instance variables : These are declared in a class, but outside a method, constructor or any block. Every object created from a class definition would have its own copy of variable. These are non-static variables.

Syntax : Object Reference. Variable name

(iii) Class Variables : These are static variables that declared with the static keyword in a class, but outside a method, constructor. It can be accessed by calling with class name.

Syntax : Class Name. Variable Name

- **Wrapper Classes in Java :**

For each fundamental data type there exist a pre-defined class, such predefined class is called 'wrapper class'. The purpose of wrapper class is to convert numeric string data into numerical or fundamental data.

Example : String s = "10.6f";

$$\underbrace{\text{Float } x}_{\text{data type}} = \underbrace{\text{Float}}_{\text{wrapper class}} .\text{parse } \underbrace{\text{Float (S);}}_{\text{method name}}$$

System.out.println(x); //10.6

There are two uses with wrapper classes :

» To convert simple data types into objects.

» To convert strings into data type.

Fundamental Data type	Wrapper Class Name	Conversion method from numeric string to numeric value
byte	Byte	Public Static byte Parse Byte (String)
Short	Short	Public Static short Parse Short (String)
Int	Integer	Public Static integer Parse Int (String)
long	Long	Public Static Long Parse Long (String)
double	Double	Public Static double Parse Double (String)
Float	Float	Public Static Float Parse Float (String)
Char	Character	
boolean	Boolean	Public static boolean Parse Boolean (String)

- **Type Casting :**

Assigning a value of one type to a variable of another type is called 'type casting'.

These are two type of type casting as follows :

(i) Implicit Type casting : Lower size data type is assigned to higher size data type. Java compiler upgrades lower type to higher type and it is termed as implicit type casting.

(ii) Explicit Type casting : Higher size date type cannot be assigned to a data type of lower size implicitly.

Methods, Recursion and Simple Programming

Control Structure	Syntax

If-else statement
 	
The if else statement tests the result of a condition, and perform appropriate action based on result.
	if(condition 1){
	action 1;
	}else if(condition 2){
	action 2;
	}else{
	action 3;
	}
	where condition is boolean expression, it returns true or false value;

Switch-case
 	
It can be used as an alternative for if-else condition. If the programmer has to make number of decision, and all the decisions depend on the value of variable, then switch case is used instead of if else condition.
	Switch(expression){
	case 1 :
	action 1 statement ;
	break;
	case 2 :
	action 2 statement;
	break;
	.
	.
	.
	case N :
	action N statement;
	break; default;
	default statement;
	}
	Where expression: is variable containing the value to be evaluated. It must be of type byte, short, int, char.
	default : is an optional keyword used to specify the statements to be executed only when all the case statements evaluate to false.

The **while loop** executes till condition is specified. It executes the steps mentioned in the loop only if the condition is true. This is useful where programmer doesn't know in advance that how many times the loop will be executed.	**Syntax** of while loop :
	While (condition){
	action statements :
	.
	.
	.
	}
	Where condition : is any boolean expression that returns a true or false value. The loop continues upto condition returns true value.

Do-while loop The do-while loops execute certain statements till the specified condition is true. This loop ensures that the loop body is executed atleast once.	do { action statements; }while(condition) do { action statements; }while(condition);

For loop
All loops have some common feature; a counter variable that is initialized before the loop begins. The for loop provides the feature that, initialized the variable before loop begins, a condition for counter variable and counter upto which loop lasts.
	for(initialization statements; condition; increment/decrement statement){
	action statements;
	.
	.
	}
	where initialization statements : sets or initialize the value for counter variable.
	condition : A boolean expression that returns true or false value. The loop terminates if false value is returned.
	Increment/decrement statements : Modifies the counter variable. This statements are always executed after the action statements, and before the subsequent condition check.

Break statement The break statements are used to, Terminate a statement sequence in a switch statement Exit a loop	break;
Continue statement continue is used to skip the remaining statements of the current iteration and start the next iteration.	continue;

Table : Function and their description

Function	Description
sin (x)	Returns the sine of the angle x in radians
cos (x)	Returns the cosine of the angle x in radians
tan (x)	Returns the tangent of the angle x in radians
asin (y)	Returns the angle whose sine is y
acos (y)	Returns the angle whose cosine is y
atan (y)	Returns the angle whose tangent is y
atan 2 (x, y)	Returns the angle whose tangent is x/y
pow (x, y)	Returns x raised to y (xy)
exp (x)	Returns e raised to x (e^x)
log (x)	Returns the natural logarithm of x
ceil (x)	Returns the square root of x
sqrt (x)	Returns the square root of x
ceil (x)	Returns smallest whole number greater than or equal to x
floor (x)	Returns the largest whole number less than or equal to x
rint (x)	Returns the truncated value of x
round (x)	Returns the integer closest to the argument
abs (a)	Returns the absolute value of a
max (a, b)	Returns the maximum of a and b
min (a, b)	Returns the minimum of a and b

Arrays

Arrays are the type of data structure which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data and also a collection of variables of the same type.

- **One-Dimensional Arrays**

A one-dimensional array is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is

type var_name[ ]=new type[size];

Here, type declares the base type of the array. The base type determines the data type of each element that comprises the array. Thus, the base type for the array determines what type of data the array will hold. For example, the following declares an array named month_days with the type "array of int": int month_days[ ];

- **Multidimensional Arrays**

In Java, multidimensional arrays are actually arrays of arrays. There are a couple of subtle differences. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two-dimensional array variable called **twoD**.

int twoD[][] = new int[4][5];

This allocates a 4 x 5 array and assigns it to **twoD**. Internally this matrix is implemented as an array of arrays of **int**. The following program numbers each element in the array from left to right, top to bottom, and then displays these values:

```
// Demonstrate a two-dimensional array.
class TwoDArray {
public static void main(String args[]) {
int twoD[][]= new int[4][5];
int i, j, k = 0;
for(i=0; i<4; i++)
for(j=0; j<5; j++) {
twoD[i][j] = k;
k++;
}
for(i=0; i<4; i++) {
for(j=0; j<5; j++)
System.out.print(twoD[i][j] + " ");
System.out.println();
}}}
```

This program generates the following output:

```
0 1 2 3 4
5 6 7 8 9
10 11 12 13 14
15 16 17 18 19
```

Strings

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

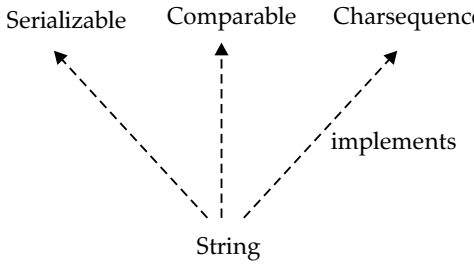
```
char[] ch={'j','a','v','a','t','p','o','i','n','t'};
String s=new String(ch);
```

is same as:

```
String s="javatpoint";
```

Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, Comparable and *CharSequence* interfaces.



- **How to create String object?**

  There are two ways to create String object:

  By string literal

  By new keyword

  (1) String Literal

  Java String literal is created by using double quotes. For Example:

  String s="welcome";

  Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

  String s1="Welcome";

  String s2="Welcome";//will not create new instance

  (2) By new keyword

  String s=new String("Welcome");//creates two objects and one reference variable

  In such case, JVM will create a new string object in normal (non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap (non pool).

  Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

No.	Method	Description
1.	char charAt(int index)	returns char value for the particular index
2.	int length()	returns string length
3.	static String format (String format, object... args)	returns formatted string
4.	static String format(Locale l, String format, Object... args)	returns formatted string with given locale
5.	String substring(int beginIndex)	returns substring for given begin index
6.	String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index
7.	boolean contains(CharSequence s)	returns true or false after matching the sequence of char value
8.	static String join(CharSequence delimiter, Char Sequence... elements)	returns a joined string
9.	Static String join (CharSequence delimiter, iterable < ? extends CharSequence > elements)	returns a joined string
10.	boolean equals(Object another)	checks the equality of string with object
11.	booleanis Empty()	checks if string is empty
12.	String concat(String str)	concatenates specified string
13.	String replace(char old, char new)	replaces all occurrences of specified char value
14.	String replace(CharSequence old, CharSequence new)	replaces all occurrences of specified CharSequence
15.	static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.
16.	String[] split(String regex)	returns splitted string matching regex
17.	String[] split(String regex, int limit)	returns splitted string matching regex and limit
18.	String intern()	returns interned string
19.	int indexOf(int ch)	returns specified char value index
20.	int indexOf(int ch, int fromIndex)	returns specified char value index starting with given index

21.	int indexOf(String substring)	returns specified substring index
22.	int indexOf(String substring, int fromIndex)	returns specified substring index starting with given index
23.	String toLowerCase()	returns string in lowercase.
24.	String toLowerCase(Locale l)	returns string in lowercase using specified locale.
25.	String toUpperCase()	returns string in uppercase.
26.	String toUpperCase(Locale l)	returns string in uppercase using specified locale.
27.	String trim()	removes beginning and ending spaces of this string.
28.	static String valueOf(int value)	converts given type into string. It is overloaded.

KNOW THE TERMS

- **Objects :** Objects contains several states and behaviours.
- **Class :** A template or blueprint that describes the behaviour as state that the object of its types supports.
- **Data Abstraction :** It is the process that involves identifying the essential features without including the internal details.
- **Data Encapsulation :** A mechanism to bind the code and data together for manipulation is called data manipulation.
- **Inheritance :** It is the process by which one object acquires the properties of another objects. It can provide the idea of reusability, drive a new class from the existing code. It means that a process in which a class is derived from the base class is called inheritance.
- **Polymorphism :** An ability to take one operation and acts with different behaviour according to situation. is known as polymorphism.
  - ñ The scope of a variable is the part of the program over which the variable name can be referenced.
  - ñ Assigning a value of one type to a variable to another type is called type casting.
- **Default constructor :** It is automatically created by complier in the absence of explicit constructor.
- **Parameterized constructor :** These constructors are needed to pass parameters on creation of objects.
- **Constructors with default arguments :** It is required to define constructors with default arguments.
- **Autoboxing :** It is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper class
- **Unboxing :** a reverse process of autoboxing is known as unboxing.
- **Recursion** is when a function calls itself. That is, in the course of the function definition there is a call to that very same function.
- **Method Overloading :** When a class has two or more methods by the same name but different parameters
- An array is a group of like-typed variables that are referred to by a common name. Arrays of any type can be created and may have one or more dimensions. A specific element in an array is accessed by its index. Arrays offer a convenient means of grouping related information.
- A one-dimensional array is, essentially, a list of like-typed variables. To create an array, you first must create an array variable of the desired type. The general form of a one-dimensional array declaration is type *var-name*[ ];
- *Multidimensional arrays* are actually arrays of arrays. To declare a multidimensional array variable, specify each additional index using another set of square brackets. For example, the following declares a two dimensional array variable called *twoD*.

  int twoD[ ][ ] = new int[4][5];
- String defines a sequence of characters. The String type is used to declare string variables. You can also declare arrays of strings. A quoted string constant can be assigned to a String variable. A variable of type String can be assigned to another variable of type String. You can use an object of type String as an argument to println( ).

- Java provides three classes as
  Character class can hold only single character.
  String class can hold strings that cannot be changed.
  String Buffer class can hold strings that can be changed or modified.
- Substring( ), string concatenation as concat( ) and string length as length( ) etc. are the operations defined on string.

❑❑