SECTION A

CHAPTER-1
BOOLEAN ALGEBRA

**Note- These Concepts are only for your reference. Questions from these parts will not be asked in the Final Examination.*

Topic-1

Propositional Logic

Concepts covered: Propositional Logic; Negation; Conjunction; Disjunction; Conditional; bi-conditional; truth table; equivalence laws of propositions.

Revision Notes

- A proposition is a declarative sentence that is either 'True' or 'False' but not both. Propositions 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 ($\bar{S}$) is negation of proposition S.

    For example,

    S: Ram eats the mango.

    Not ($\bar{S}$): Ram doesn't eat the mango.

  - **Conjunction (AND):** If R and S are two propositions, then R and S are connected by 'AND' or '$\wedge$' symbol.

    For Example,

    R: He is a player.

    S: He plays football very well.

    $R \wedge S$: He is a player AND he plays football very well.

  - **Disjunction (OR):** It is represented by 'OR' or '$\vee$' symbol. Let R and S be two propositions. Then $R \vee S$ is read as R or S.

    For Example,

    R: She is a player.

    S: She is a dancer.

    $R \vee S$: She is a player OR dancer.

  - **Conditional ($\Rightarrow$):** $A \rightarrow B$ is called a conditional statement. It is also read as 'A implies B'.

    For Example,

    A: You will do hard work.

    B: You will get success.

    $A \rightarrow B$: If you will do hard work, then you will get success.

  - **Biconditional ($\Leftrightarrow$):** $A \leftrightarrow B$ is known as biconditional statement or proposition. ' $\leftrightarrow$ ' or '$\Leftrightarrow$' shows equivalence relation between two propositions.

    For example:

    A: She is smart.

    B: She is beautiful.

    $A \rightarrow B$: 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 0) are called 'truth values'.

    We can determine the truth values of the various connections as follows:

p	q	~ p	~ p	p ∧ q	p ∨ q	p ⇒ q	p ⇔ 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 q are WFF, then (p ∧ q), (p ∨ q), p ⇒ q, p ⇔ 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:

    $p \vee \bar{p}$, is a tautology, $p \wedge \bar{p}$, is a contradiction and p ∨ p, or p ∧ p, is a contingency.
  - **Argument:** Sequence of statements (premises). An argument is valid if it is a tautology, otherwise it is fallacy, (non-valid).
  - **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 $\bar{p} \rightarrow \bar{q}$,
  - **Contra positive:** The contra positive of $p \rightarrow q$ is $\bar{q} \rightarrow \bar{p}$.
- **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:**

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

Key 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 (∧), OR (∨), Not (~), conditional (→) and biconditional (↔) 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 if it is a tautology.
- **Syllogism:** Logical process of drawing conclusion from given propositions.

Topic-2 Theorems and Postulates of Boolean Algebra

Concepts covered: Boolean Algebra; Laws of Boolean algebra.

Revision Notes

- Boolean Algebra is an algebra given by George Boolean which uses 0 and 1 as binary variables, and three basic operations OR, AND and NOT.
- Laws of Boolean Algebra

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

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

  For example,

  (i) Dual of $X + Y \cdot Z$ is $X \cdot (Y + Z)$.

  (ii) Dual of O is always 1 and Vice-versa.

Key Terms

- **Basic Postulates:**

  $A + 0 = 0 + A = A0 + 1 = 1$; $a + 0 = 0$

  $A.1 = 1 \wedge A = A$ $\quad 0 \wedge 1 = 0$; $1 + 1 = 1$

  $A + A = A$; $A.A = A$

  $A + A– = 1$; $A.\overline{A} = 0$

  $(\overline{A}) = A$
- **De Morgan's Theorem:** The complement of the sum of variables is equal to the product of the complement of the variables.
- $\overline{(A + B)} = \overline{A} \,.\, \overline{B}$ or $\overline{(A \,.\, B)} = \overline{A} + \overline{B}$

Topic-3 Forms of Representing Boolean Expressions

Concepts covered: Minterms; Maxterms; SOP form; POS form; Canonical forms

Revision Notes

- 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 Max terms.
- A Boolean expression is an expression that results in either true or false. 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	0	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 is 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. with $X + \overline{X}$ if x is missing) 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. with $X \cdot \overline{X}$ if x is missing) missing variables.
  3. Expand all terms and remove the duplicate terms such that X·X = X or X·X·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:
     - **(i)** Change every OR sign (+) to AND sign (·).
     - **(ii)** Change every AND sign (·) to OR sign (+).
     - **(iii)** Change normal variable (A) into complement variable ($\overline{A}$) 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:
     - **(i)** Change every OR sign (+) to AND sign (·).
     - **(ii)** Change every AND sign (·) to OR sign (+).
     - **(iii)** Change normal variable (A) into complement variable ($\overline{A}$) and vice versa.
  3. The produced expression is the canonical SOP expression.

Key Terms

- **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)·(x' + y) is a POS form.
- Miniterm 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.

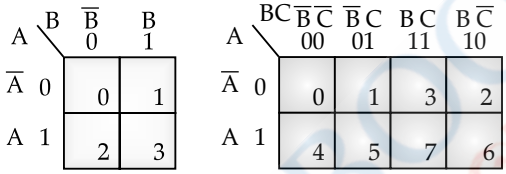
Topic-4 Karnaugh Map (K-map)

Concepts covered: Karnaugh map; Reducing a boolean expression into its canonical form using K-map

Revision Notes

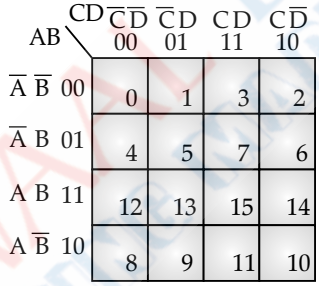
- **Karnaugh Map (K-map) Method:** It is a most popular method for simplification of Boolean expressions of two variables, 3-variables, 4-variables and so on. It is a graphical display of the fundamental products in a truth table. It is a rectangle made up of certain number of squares, each square representing a maxterm or minterm.

  Representation of K-maps:



2-VariablesK-map

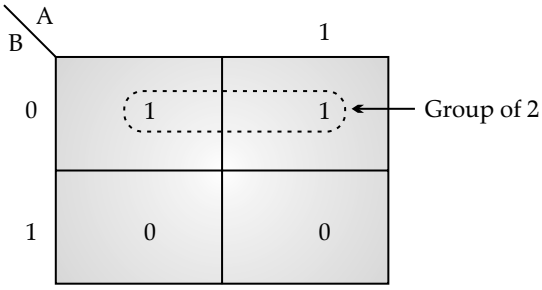
3-Variables K-map



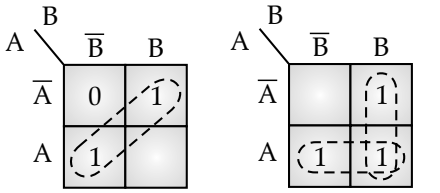
4-Variables K-map

- **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$\overline{C}$, $\overline{A}$BC, A$\overline{B}\overline{C}$, $\overline{B}$C, AB$\overline{C}$, 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 together in the form of Octet, Quad and Pair.

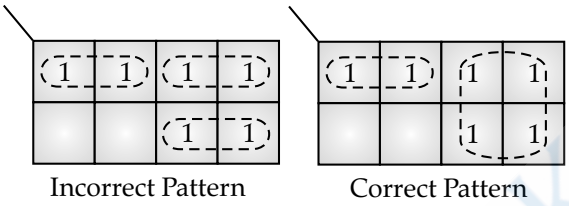
  Some rules are as follows:
  - Groups must contain 1, 2, 4, 8, or in general 2^n cells.
  - That is if n = 1, a group will contain two 1's since $2^1 = 2$.



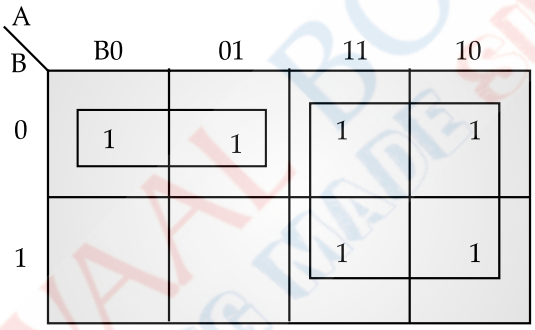
  - Groups may not include any cell containing a zero. It means we cannot make a pair of 0 and 1.
  - Groups may be vertical or horizontal, but not diagonal.



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



❑❑

CHAPTER-2

COMPUTER HARDWARE

Topic-1 Logic Gates

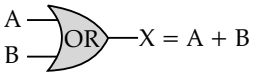
Concepts covered: AND, OR, Not logic gates; Universal logic gates; XOR, XNOR gates; Realisation of logic gates using universal gates.

Revision Notes

- A logic gate is an electronic circuit which operates on one or more signals to produce an output signal.
- 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 are not produced.
  - **Types of Logic Gates:**
    The logic gates represent electronic circuits in Boolean algebra. There are two types of logic gates namely:
    1. Fundamental or Elementary Logic Gates
    2. Universal Gates
    1. Fundamental Logic Gates: AND, OR and NOT gates are called fundamental logic gates.
       (i) AND Gate: This logic gate contains at least two inputs and one output. It will give output when all inputs are enabled in the form of 1.

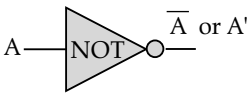
A	B	X = A.B
0	0	0
0	1	0
1	0	0
1	1	1

(ii) OR Gate: This logic gate contain at least two inputs and single output. It enables or gives output if any one of the inputs are enabled in the form of 1.



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

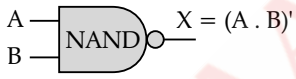
(iii) NOT Gate: It has only single input and single output. It acts as inverter.



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

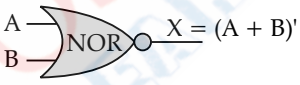
2. **Universal Gates:** NAND and NOR gates are known as universal gates. With the help of these gates all fundamental gates can be created.

(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.

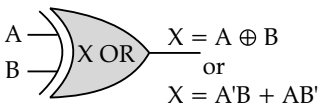


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

- **Others Specific Logic Gates:**

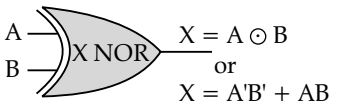
1. **Exclusive OR (X-OR) Gate:** It is a digital logic gate that gives a True (1) output when the number of True inputs is odd.

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



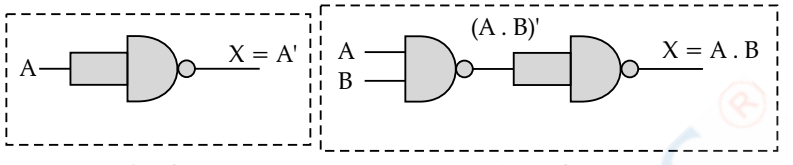
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

2. **Exclusive-NOR Gate:** It is combination of Not gate and XOR gate. It is the logical complement of the XOR gate. It gives a True (1) output when all of its inputs are True (1 or High) or when all of its inputs are false (0 or Low). The algebraic notation used to represent the XNOR operation is $S = A \odot B$

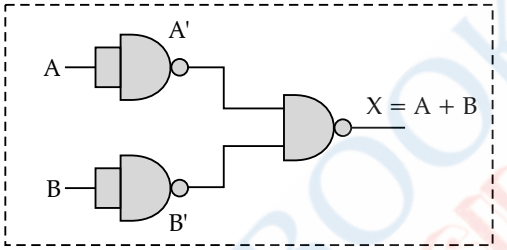


A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

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

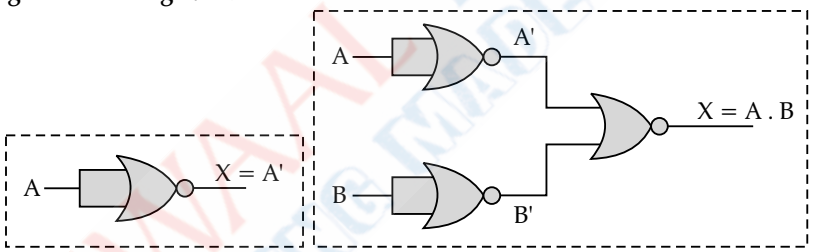


NOT Gate AND Gate

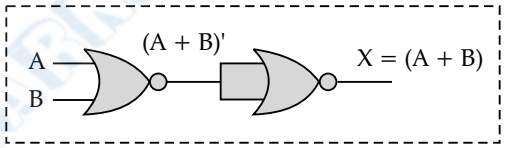


OR Gate

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



NOT GATE AND Gate



OR Gate

Key Terms

- Elementary Logic Gates are NOT, AND, OR Gates
- Universal Gates are NAND gate and NOR gate because with the help of these 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 (0).
- **OR Gate:** At least two inputs and single output is required for OR gate. It is represented by '+' 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 also called 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.

Topic-2 Designing Simple Logic Circuits

Concepts covered: Designing circuits for given Boolean expression

Revision Notes

- The K-map is a systematic way of simplifying Boolean expressions. With the help of the K-map method, we can find the simplest POS and SOP expression, which is known as the minimum expression.
  - The K-map provides a cookbook for simplification.
  - The K-map method is used for expressions containing 2, 3, 4, and 5 variables.
  - There are the following steps used to solve the expressions using K-map:
    1. First, we find the K-map as per the number of variables as 2, 3, 4, and 5 variables.
    2. Find the maxterm and minterm in the given expression.
    3. Fill cells of K-map for SOP with 1 respective to the minterms.
    4. Fill cells of the block for POS with 0 respective to the maxterm.
    5. Next, we create rectangular groups that contain total terms in the power of two like 2, 4, 8, … and try to cover as many elements as we can in one group.
    6. With the help of these groups, we find the product terms and sum them up for the SOP form.

Topic-3 Combinational Circuits

Concepts covered: Combinational circuits; Adder (Half and Full); Encoder and decoder; Multiplexer and Demultiplexer

Revision Notes

- 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.

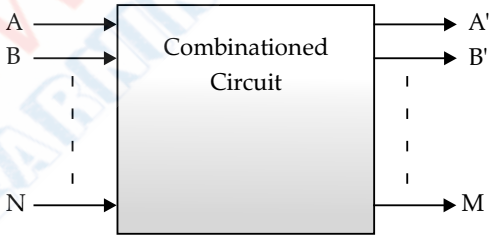
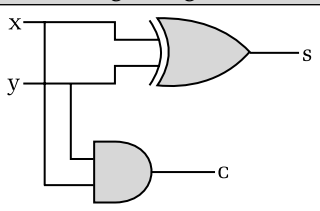
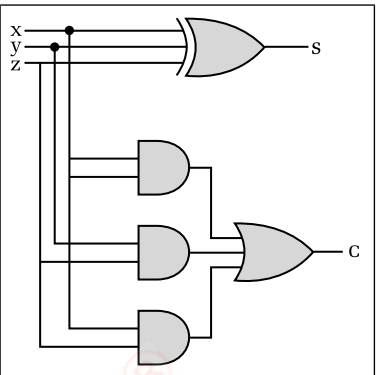
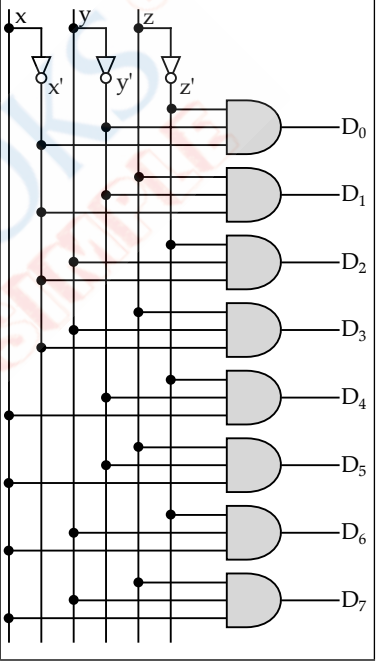


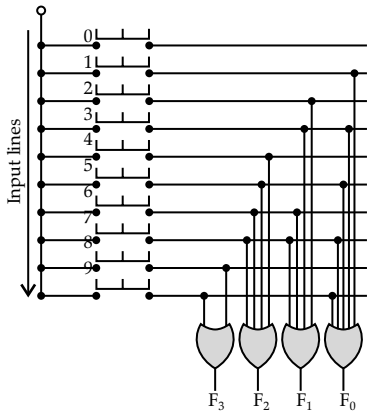
Fig. Block Diagram

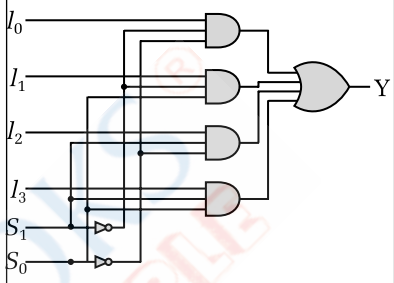
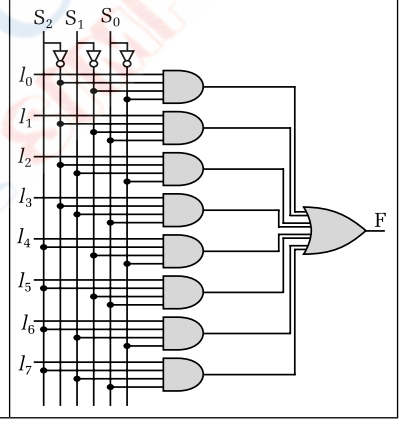
- The output of combinational circuit at any instance of time depends only on the levels present at input points.
- They do not use any memory. The previous state of input doesn't have any effect on the present state of the circuit.

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.

Circuit	Description	Expression	Truth Table	Logic diagram
Half Adder	Performs the addition of two bits	$S = x'y + xy' = x \oplus y$ $C = xy$	x y S C 0 0 0 0 0 1 1 0 1 0 1 1 1 1 0 1	

<table>
<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
$m \leq 2^n$
$n = 2$</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>
</table>

<table>
<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>

Key Terms

- **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.

❑❑

SECTION-B

CHAPTER-3

PROGRAMMING IN JAVA

Note- These Concepts are only for your reference. Questions from these parts will not be asked in the Final Examination.

Topic-1

Fundamentals of Java

Concepts covered: Introduction to Java; OOPs and its concepts; Tokens in Java; Data types in java; Scope of identifiers; Wrapper classes; Type casting; Conditional Constructs; Iterative constructs

Revision Notes

- Java is a high level programming language originally developed by Sun Micro systems and released in 1995. Java runs on several platforms as windows, Unix and Mac OS.
- 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 linked with the features of real-world.
  - **Objects:** Objects have characteristics and behaviours. For example; a student has states or characteristics: name, age, group as well as behaviour such as student studies, plays, acts, 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:** Process of wrapping of data and methods in a single unit is called encapsulation. It is achieved by using class concept in Java. The isolation of the data from direct access by the program is called 'data hiding'.
  - **Polymorphism:** A process of representing one entity in various forms is called 'Polymorphism'. For example, a single person 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 components of a Java program 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:** A constructor is the member function of a class that is used to initialize the data members when an object of that class is created. The java compiler builds a default constructor for each class. Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they have the same name as the class. A class can have more than one constructor.

  Example:

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

- **Data Types:**

  Data types specify the different sizes and values that can be stored in the variable. 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	1 bit	True/False	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, class 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 consist of alphabet, digit, special 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, an initial value should be assigned to it before the first use. These have scope only within the block in which they are declared.

  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 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

- **Public** members are the members that can be directly accessed by any function.
- **Private** members are the class members that are hidden from the outside world. They can be used only by member functions (and friends) of the class in which it is declared.
- **Protected** members are the members that can be used only by member functions, friend functions and sub class functions.
- A **constructor** is a member function of a class that is automatically called when an object is created of that class. It has the same name as that of the class's name and its primary job is to initialize the object to a legal value.
- **Default** constructor is a constructor that accepts no parameter. If a class has no explicit constructor defined, the compiler will supply a default constructor.
- **Parameterized** constructors can accept parameters.
- **Copy** constructor takes one argument, also called one argument constructor. The main use of copy constructor is to initialize the objects while in creation, also used to copy an object. The copy constructor allows the programmer to create a new object from an existing one by initialization.
- **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 a value of primitive type in an object.

  Example:
  ```
  String s = "10.6f";
  float x = Float.parseFloat (S);
  ```
  Object of wrapper class Float — wrapper class — method name

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

  There are two uses of wrapper classes:

  - To convert simple data types into objects.
  - To convert strings into numbers and then to other primitive type.

Fundamental Data type	Wrapper Class Name	Conversion method from numeric string to numeric value
byte	Byte	static byte parseByte (String)
short	Short	Static short parseShort (String)
int	Integer	Static integer parseInt (String)

long	Long	Static Long parseLong (String)
double	Double	Static double parseDouble (String)
float	Float	Static Float parseFloat (String)
char	Character	charValue()
Boolean	Boolean	static Boolean parseBoolean (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.

Control Structure	Syntax

If-else statement
 	
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.

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){
	action statements;
	.
	.
	}

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;

Key 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.
  - The scope of a variable is the part of the program over which the variable name can be referenced.
  - 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.

Topic-2 Methods and Recursion

Concepts covered: Methods; Need for methods; Call by value; Call by reference; Recursion; Recursive functions

Revision Notes

- A function or method in Java contains the executable code.
- Need for methods
  - Complex problem can be broken into smaller and more easily understandable tasks.
  - Low level details about as to how a method is accomplishing a task are hidden from the users.
  - Reusability of the code increases.

  A function must be defined before it is used anywhere in the program.
- Syntax of a method:

```
Public static int method name(int a, int b)
{
   //body
}
```

public Static -> modifier
int -> return type
methodName -> name of the method
a, b -> formal parameters
int a, int b -> list of parameters

- A method in Java has to be inside a class.
- The first line of the function definition that tells the program about the type of the value returned by the function and the number and type of arguments.
- A function is called by providing the function name followed by the parameters enclosed in parentheses.
- A function can be invoked in two ways:
  - **Call by value:** value of actual parameters are copied into the formal parameters.
  - **Call by reference:** A reference of the actual parameters are passed to the formal parameters.
- A function terminates either when a return statement is encountered or the last statement of the function is executed.
- main() method is the logical entry point of the program.
- A static method of a class can't access a non-static method or variable of its own class.
- **Recursive function:** A function definition that calls itself.
- A recursive function must have two cases
  - The recursive case
  - The base case
- When many methods have the same name with different functionality it is called overloading.
- **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

Key Terms

- **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

Topic-3 Arrays

Concepts covered: Arrays; Array implementation; Single and multi dimensional arrays; calculating size and address of arrays.

Revision Notes

- Arrays are the type of data structure which store 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 must first create an array variable of the desired type.

  The general form of a one-dimensional array declaration is:

  type var_name[ ];

  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 subtitle 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
```

Key Terms

- 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.
- An array of objects is created like an array of primitive type data items.

Topic-4 Strings

Concepts covered: Strings (declaration and initialization; String class methods and functions; String buffer

Revision Notes

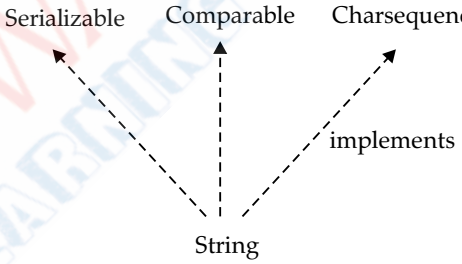
- In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

```
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.	Boolean is 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.

Key Terms

- String Buffer class is part of java.long package.
- 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.

Topic-5 File Handling

Concepts covered: Types of files; Opening and reading and writing files; File handling methods

Revision Notes

- File handling is an essential part of any programming language. Using files, a program can store data on a storage device.
- To perform various actions on a file, such as read, write, and so on.

Why File Handling is Required?

- File Handling is an integral part of any programming language as file handling enables us to store the output of any particular program in a file and allows us to perform certain operations on it.
- In simple words, file handling means reading and writing data to a file.

Streams in Java

- In Java, a sequence of data is known as a stream.
- This concept is used to perform I/O operations on a file.
- There are two types of streams :
  1. Input Stream:
  2. Output Stream:

- The input stream is used to read data from numerous input devices like the keyboard, network, etc. InputStream is an abstract class, and because of this, it is not useful by itself. However, its subclasses are used to read data.

  Creating an InputStream

  // Creating an InputStream
  InputStream obj = new FileInputStream();
  Here, an input stream is created using FileInputStream.
- The output stream is used to write data to numerous output devices like the monitor, file, etc. OutputStream is an abstract superclass that represents an output stream. OutputStream is an abstract class and because of this, it is not useful by itself. However, its subclasses are used to write data.

  Creating an OutputStream

  // Creating an OutputStream
  OutputStream obj = new FileOutputStream();
  Here, an output stream is created using FileOutputStream.

- Methods for file handling:

S.No.	Method	Description
1.	public void close() throws IO Exceptions{}	This method closes the file output stream. Releases any system resources associated with the file. Throws an IO Exception.
2.	public void finalize() throws IO Exception{}	This method cleans up the connection to the file. Ensures that the close method of this file output stream is called when there are no more references to this stream. Throws an IO Exception.
3.	public void write(int w) throws IO Exception{}	This method writes the specified byte to the output stream.
4.	public void write(byte[] w)	Writes w.length bytes from the mentioned byte array to the OutputStream

Key Terms

- File handling in Java means to read from and write to file in Java. It provides the basic I/O package for reading and writing streams.java.io package allows to do all input and output tasks in Java.
- Java.io package contains the inbuilt Java classes and predefined methods using which you can read from and write data to file.
- Byte stream classes are as follows: InputStream, OutputStream, FileInputStream, FileOutputStream, ByteArrayInputStream, ByteArrayOutputStream, PrintStream, RandomAccessFile etc.
- Methods defined by InputStream: read(), read(byte[] r), available(), close(), etc.
- Methods defined by OutputStream: write(int w), write(byte[] w), finalize(), close(), etc.

Topic-6 Exception Handling & JVM

Concepts covered: Exceptions and their types, functions for exception handling, JVM, JDK

Revision Notes

- An exception is an abnormal condition that arises in a code sequence at run time. In other words, an exception is a run-time error. In computer languages that do not support exception handling, errors must be checked and handled manually—typically through the use of error codes, and so on. This approach is as cumbersome as it is troublesome. Java's exception handling avoids these problems and, in the process, brings run-time error management into the object oriented world.
- **Exception Handling:** A Java exception is an object that describes an exceptional (that is, error) condition that has occurred in a piece of code. When an exceptional condition arises, an object representing that exception is created and thrown in the method that caused the error. That method may choose to handle the exception itself, or pass it on. Either way, at some point, the exception is caught and processed. Exceptions can be generated by the Java run-time system, or they can be manually generated by your code. Exceptions thrown by Java relate to fundamental errors that violate the rules of the Java language or the constraints of the Java execution environment. Manually generated exceptions are typically used to report some error condition to the caller of a method.
- Java exception handling is managed via five keywords: try, catch, throw, throws, and finally.
  - Program statements that you want to monitor for exceptions are contained within a try block.
  - If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch) and handle it in some rational manner.
  - System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. A throw is followed by an instance variable.
  - Any exception that is thrown out of a method must be specified as such by a throws clause. throws is followed by exception class names
  - Any code that absolutely must be executed after a try block completes is put in a finally block.
- Two types of exceptions are:
  (i) Checked exception
  (ii) Unchecked exception
- General form of an exception-handling block:

```
try     {
        // block of code to monitor for errors
}
catch   (ExceptionType1 exOb) {
    // exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
    // exception handler for ExceptionType2
}
    // ...
finally {
    // block of code to be executed after try block ends
}
```

- **Java Virtual Machine (JVM)**
  - A JVM is an abstract computation machine that enables a computer to run a Java program. There are three notions of the JVM as specification, implementation and instance.
  - The specification is a document that describes what is required of a JVM implementation. A JVM implementation is a computer program that meets the requirements of the JVM specification. An instance is an implementation running in a process that executes a computer program compiled into Java bytecode.
  - Java development Kit (JDK) is a superset of a Java Runtime Error (JRE) and contains tools for Java programmers, e.g. a javac compiler.
  - Java-in-time Compiler (JIT) is the part of the JVM that is used to speed up the execution time. JIT interpret parts of the bytecode that have similar functionality at the same time, and hence reduce the amount of time needed for full interpretation.

Key Terms

- Way of handling anomalies situations in a program-run is known as exception-handling
- Exception handling has three keywords: try, throw and catch
- Program statements that you want to monitor for exceptions are contained within a try block.
- If an exception occurs within the try block, it is thrown. Your code can catch this exception (using catch) and handle it in some rational manner.
- System-generated exceptions are automatically thrown by the Java run-time system. To manually throw an exception, use the keyword throw. A throw is followed by an instance variable.
- A JVM is an abstract computation machine that enables a computer to run a Java program. There are three notions of the JVM as specification, implementation and instance.

❑❑

SECTION-C

CHAPTER-4
INHERITANCE, INTERFACES, POLYMORPHISM, DATA STRUCTURES

**Note- These Concepts are only for your reference. Questions from these parts will not be asked in the Final Examination.*

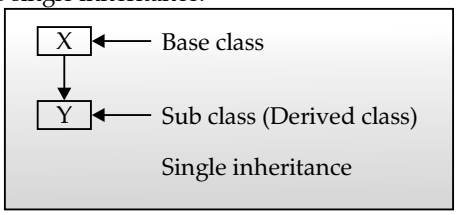
Topic-1 Inheritance

Concepts covered: Inheritance; Super class; Sub class; Visibility Mode; Method overriding; Super keyword

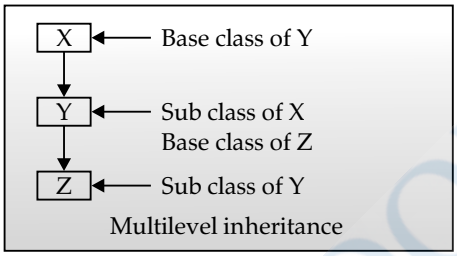
Revision Notes

- Inheritance is a mechanism in which one object acquires all the properties and behaviours of a parent object.
- Inheritance is an important part of OOPs (Object Oriented Programming System).
- **Derivation:** It is the relationship between a base class and derived class. It is also known as inheritance hierarchy.
- **Inheritance graph:** A derivation from multiple base classes is known as inheritance graph.
- **Subclass:** It is the derived class and its declaration must include name of its base class.
- **Base class:** It is the base class or the class from which the sub class is derived.
- **Class object:** It is a class defined in java.lang package and every class that we define is a subclass of this class.
- Only single inheritance, hierarchial and multi level inheritance can be implemented in Java.
- Every class in Java has only one immediate or direct superclass.
- Inheritance relation is transitive. i.e., every class inherits states and behaviour from all superclasses which are higher in class hierarchy.

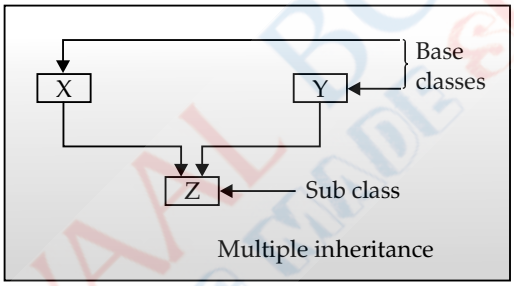
- **Forms of inheritance**
  - **Single inheritance:** When a subclass inherits from only one base class, and there are only 2 classes in this relationship, then it is known as single inheritance.



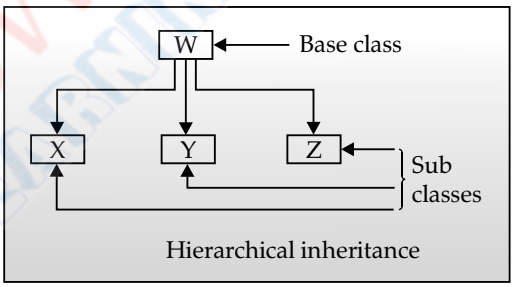
  - **Multilevel inheritance:** When a subclass inherits from a class that itself inherits from another class, it is known as multilevel inheritance.



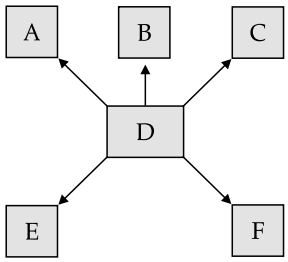
  - **Multiple inheritance:** When a subclass inherits from multiple base classes it is known as multiple inheritance. Java does not support multiple inheritance.



  - **Hierarchical inheritance:** When many subclasses inherit from a single base class it is known as hierarchical inheritance.



- **Hybrid Inheritance:** When a subclass inherits from multiple base classes and all of its base classes inherit from a single base class it is hybrid inheritance.



- Variables in superclass having the same names as variables of sub-class are hidden.
- Methods in superclass with same name as that of member methods of subclass are overridden.
- To refer to hidden data members and overridden methods of superclass 'super' keyword is used.
- **Overriding the inherited method:** If two method of the base and the derived class have the same signature then the subclass method hides or overshadows the base class method. This is called overriding the inherited member.
- Constructor of super class are not inherited by sub classes.

- Default constructor of the super class is automatically invoked before any statements within the constructors are executed.
- Polymorphism is Java is a concept by which we can perform a single action in different way.
- Polymorphism means many forms.
- There are two types of polymorphism in Java:
  - Compile Time polymorphism
  - RunTime Polymorphism
- Two ways to implement polymorphism in Java are:
  (a) Method Overloading
  (b) Method Overriding
- Method overloading is one of the ways through which java supports polymorphism. Method overloading can be done by changing number of arguments or by changing the data type of arguments. If two or more method have same name and parameter list but differs in return type they are not overloaded method
- **Method Overriding:** If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
- It is used to provide the specific implementation of a method which is already provided by its superclass, and it is used for runtime polymorphism.

Key Terms

- **Inheritance:** One object acquires the properties of another object
- **Polymorphism:** Ability to take more than one form

  Constructor chaining - phenomena of calling one constructor from another constructor of same class.
- **Inline functions:** The compiler replaces the function statement with the function code itself (expansion) and then compiles the entire code.
- **Concrete superclass:** It is the class whose objects can be declared and created.
- **Abstract class:** It is the one that represents a concept only. It's objects can't be created.
- **Keyword abstract** is used to create an abstract class.
- **Abstract methods:** The methods have no method statements and are declared using abstract keyword.
- **Extends** is the keyword used to inherit the properties of a class
- The **implements** keyword is used with classes to inherit the properties of an interface.
- **This** keyword is used to refer to current object.
- **Super** keyword is used to refer to immediate parent class of a class.
- **Overriding:** To override the functionality of an existing method.

Revision Notes

- **Interface:** An interface in java is a blueprint of a class. It has static constants and abstract methods.
- **Extending Interface:** An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface.
- **Marker Interface:** An empty interface is known as tag or marker interface.
- **Nested Interface:** An interface which is declared inside another interface or class is called nested interface.
- **Use of Java Interface:** There are mainly three reasons to use interface:
  - It is used to achieve abstraction.
  - By interface, we can support the functionality of multiple inheritance.
  - It can be used to achieve loose coupling.

Key Terms

- Interface fields are public, static and final by default, and the methods are public and abstract.
  - A class implements an interface, but one interface extends another interface.

Topic-2 Interfaces

Concepts covered: Interface; Implementing Interface; Abstract class

Revision Notes

- **Interface:** An interface in java is a blueprint of a class. It has static constants and abstract methods.
- **Extending Interface:** An interface can extend another interface in the same way that a class can extend another class. The extends keyword is used to extend an interface.
- **Marker Interface:** An empty interface is known as tag or marker interface.
- **Nested Interface:** An interface which is declared inside another interface or class is called nested interface.
- **Use of Java Interface:** There are mainly three reasons to use interface:
  - It is used to achieve abstraction.
  - By interface, we can support the functionality of multiple inheritance.
  - It can be used to achieve loose coupling.

❑❑

CHAPTER-5

DATA STRUCTURES

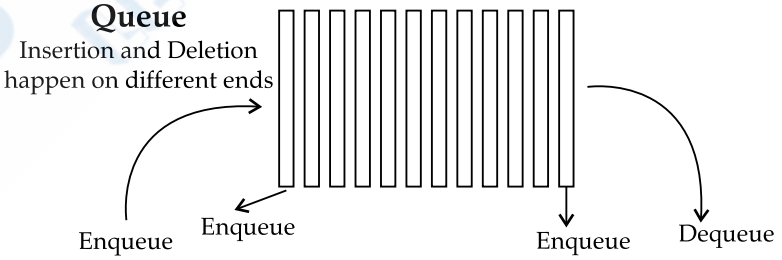
Topic-1 Queues and Stacks

Concepts covered: Queue; implementing Queues using arrays; Insertion and deletion in a queue; deque; Stacks; Implementing Stacks as arrays; Push and Pop in a stack; Conversion of an algebraic expression from infix to prefix and postfix notation

Revision Notes

QUEUE & DEQUEUE

- A Queue is a linear structure which follows a particular order in which the operations are performed.
- Queue obeys the FIFO (First In, First Out) manner. It indicates that elements are entered in the queue at the end and eliminated from the front.

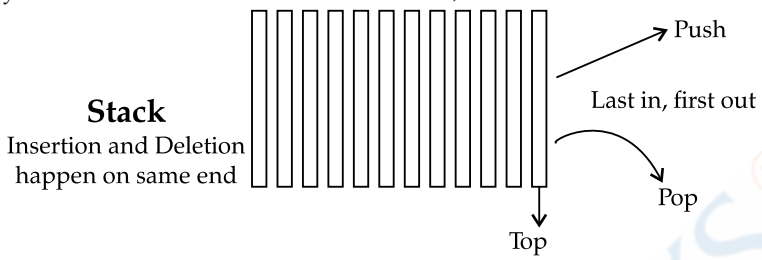


- There are two different classes which are used to implement the Queue interface. These classes are:
  - LinkedList
  - PriorityQueue
- The Deque or (double-ended queue) is also a type of queue that carries the inclusion and deletion of elements from both ends.
- The deque is also considered thread-safe.
- **Enqueue:** When an element is inserted in the queue, it is called enqueue.
- **Dequeue:** When an element is removed from the queue, this process is called dequeue.

STACK

- The stack is a linear data structure that is used to store the collection of objects.

- It is based on Last-In-First-Out (LIFO).
- The stack data structure has the two most important operations that are push and pop.
- **push():** When we insert an element in a stack then the operation is known as a push.
- If the stack is full then the overflow condition occurs.
- **pop():** When we delete an element from the stack, the operation is known as a pop.
- If the stack is empty means that no element exists in the stack, this state is known as an underflow state.



- Java collection framework provides many interfaces and classes to store the collection of objects.

Infix, Postfix, and Prefix

- **Infix:** The typical mathematical form of expression that we encounter generally is known as infix notation. In infix form, an operator is written in between two operands.

  For example, An expression in the form of A * (B + C) / D is in infix form.
- **Prefix:** In prefix expression, an operator is written before its operands. This notation is also known as "Polish notation".

  For example, The above expression can be written in the prefix form as / * A + B C D.
- **Postfix:** In postfix expression, an operator is written after its operands. This notation is also known as "Reverse Polish notation".

  For example, The above expression can be written in the postfix form as A B C + * D /.

Key Terms

- **Enqueue (qstore):** adding an element to the queue at its rear
- **Dequeue (qstore):** deleting an element at the front of a queue.
- **Empty:** status of queue being Empty.
- **Full:** status of a queue being full.
- **Front:** Refers to the first element of the queue.
- **Rear:** Refers to the last element of the queue.

Topic-2

Trees and Link list

Concepts covered: Trees; Terminology related to a tree; Binary tree; Linked list and implementing various data structures using linked list.

Revision Notes

- A binary tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element.
- The "root" pointer points to the topmost node in the tree.
- The left and right pointers recursively point to smaller "subtrees" on either side.
- A null pointer represents a binary tree with no elements -- the empty tree.
- A Binary tree is either empty or it consists of a node called the root, pointing to two nodes, left and right, each being a binary tree in itself.
- A "binary search tree" (BST) or "ordered binary tree" is a type of binary tree where the nodes are arranged in order: for each node, all elements in its left subtree are less-or-equal to the node (<=), and all the elements in its right subtree are greater than the node (>).
- A linked-list is a linear collection of data elements called nodes, pointing to the next following nodes by means of pointers.
- In linked lists number of elements need not be predetermined.

- Linked lists are of two types:
  - Single linked list.
  - doubly linked list.
- Single linked list contain nodes with single pointer pointing to the next node.
- START if a single linked list is a special reference which stores the address of the first mode.
- Next or link reference of the last node of the list is always NULL.

Key Terms

- **Parent and Child:** Every node (except the root) has a unique predecessor called its parent successor of a node is known as its child.
- **Siblings:** Two nodes of some parent.
- The line drawn from a node to its successor is called an edge.
- **Path:** Sequence of consecutive edges is called a path.
- **leaf:** Terminal node
- **branch:** A path ending in a leaf.
- **level number:** Root has level number 0 and each child has level number one more than us parent.
- **Weight/depth:** It is the largest level number of a tree.

❐❐

OSWAAL BOOKS® LEARNING MADE SIMPLE