CTEVT

Object Oriented Programming in Java

EG2203T

Fourth Semester

Prepared by Ganesh Prasad Kapadi

Course Contents:

Theory

Unit 1. Object-Oriented Programming **[3 Hrs.]**

1.1. Procedure Oriented Programming
1.2. Object-Oriented Programming
1.3. Procedure Oriented versus Object Oriented Programming
1.4. OOP principles
1.5. Advantages and Disadvantages of OOP

Unit 2. Introduction to Java **[2 Hrs.]**

2.1. Java as a Programming Platform
2.2. History of Java
2.3. Java Buzzwords
2.4. Java Virtual Machine

Unit 3. Fundamental Programming Structures **[10 Hrs.]**

3.1. Whitespace, Identifiers, Literals, Comments, Separators and Keywords
3.2. Data Types and Conversion
3.3. Variables
3.4. Constants
3.5. Operators
3.6. Strings
3.7. Control Structures
3.8. Loop
3.9. Methods
3.10. Arrays

Unit 4. Classes and Objects **[10 Hrs.]**

4.1. Defining Class
4.2. Adding Variables
4.3. Adding Methods
4.4. Static Variables, Methods, Blocks and Class
4.5. Access Control
4.6. Method Parameters
4.7. Creating Objects
4.8. Accessing class members
4.9. Setters and Getters
4.10. Constructors
4.11. Overloading Methods
4.12. Call by value, Call by reference
4.13. this keyword
4.14. final modifier
4.15. Nested Classes
4.16. Wrapper Classes in Java
4.17. Garbage Collection

Unit 5. Inheritance **[8 Hrs.]**

5.1. Introduction
5.2. Types of Inheritance
5.3. Method Overriding
5.4. Using Super keyword
5.5. Execution of Constructors in Multilevel Inheritance
5.6. Abstract Classes and Methods

Unit 6. Interface and package **[8 Hrs.]**

6.1. Defining Interfaces
6.2. Extending Interfaces
6.3. Implementing Interfaces
6.4. Accessing Interface Variables
6.5. Introduction to java Packages
6.6. Creating a Package and naming convention
6.7. Using Packages

Unit 7. Exception Handling **[6 Hrs.]**

7.1. Exceptions and its types
7.2. Exception handling fundamentals (try, catch, throw, throws and finally)
7.3. Using try and catch
7.4. Using throw and throws

Unit 8. Multithreading **[6 Hrs.]**

8.1. Introduction of Thread
8.2. Creating a Thread
8.3. Thread Priorities
8.4. Life cycle of a Thread (Thread states)

Unit 9. I/O **[7 Hrs.]**

9.1. Java.io package
9.2. Byte Stream and Character Stream classes
9.3. Using FileInputStream and FileOutputStream classes
9.4. Using FileReader and FileWriter Classes

Final written exam evaluation scheme

Unit	Title	Hours	Marks Distribution*
1	Object-Oriented Programming	3	4
2	Introduction to Java	2	3
3	Fundamental Programming Structures	10	13
4	Classes and Objects	10	13
5	Inheritance	8	11
6	Interface and package	8	11
7	Exception Handling	6	8
8	Multithreading	6	8
9	I/O and Java Applets	7	9
	Total	**60**	**80**

* There may be minor deviation in marks distribution.

VIT 1 Object - oriented programming

1.1 **Procedure Oriented Programming:-** In pop a large program is brokdown into smaller manageble part is called procedure oriented programme or Function rather then data

- A computer programming language that execute a set of command inorder is called procedural language. It is written as a list (set) of instruction telling the computer step by step. What to do, for eg:- open a file, read a number, display something

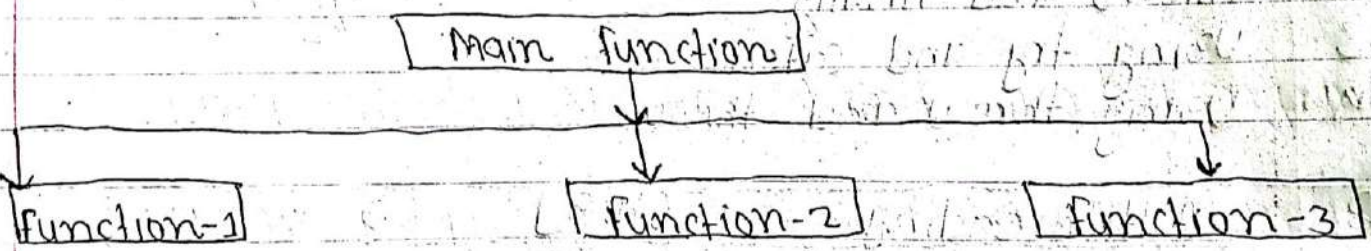


fig:- program division in pop

- **Characteristics :-**

The characteristics of pop are listed as follows:

- A large program is broken down into small manageable procedures or functions
- pop focuses on procedure or function rather then data.
- For sharing or common data among different functions the data is made global.
- The program design of procederal oriented programming flows top-down methodology.

1.2 Object oriented programming :-

Programming model that organizes software design around data or object rather function and logic.

→ The object oriented approach is :-

- The resent concept among programming paradigms.
- Fundamental idea it to combine data and functions those operaters on these data into single unit called object.
- The data of an object can be accessed only by the function associated with these object.

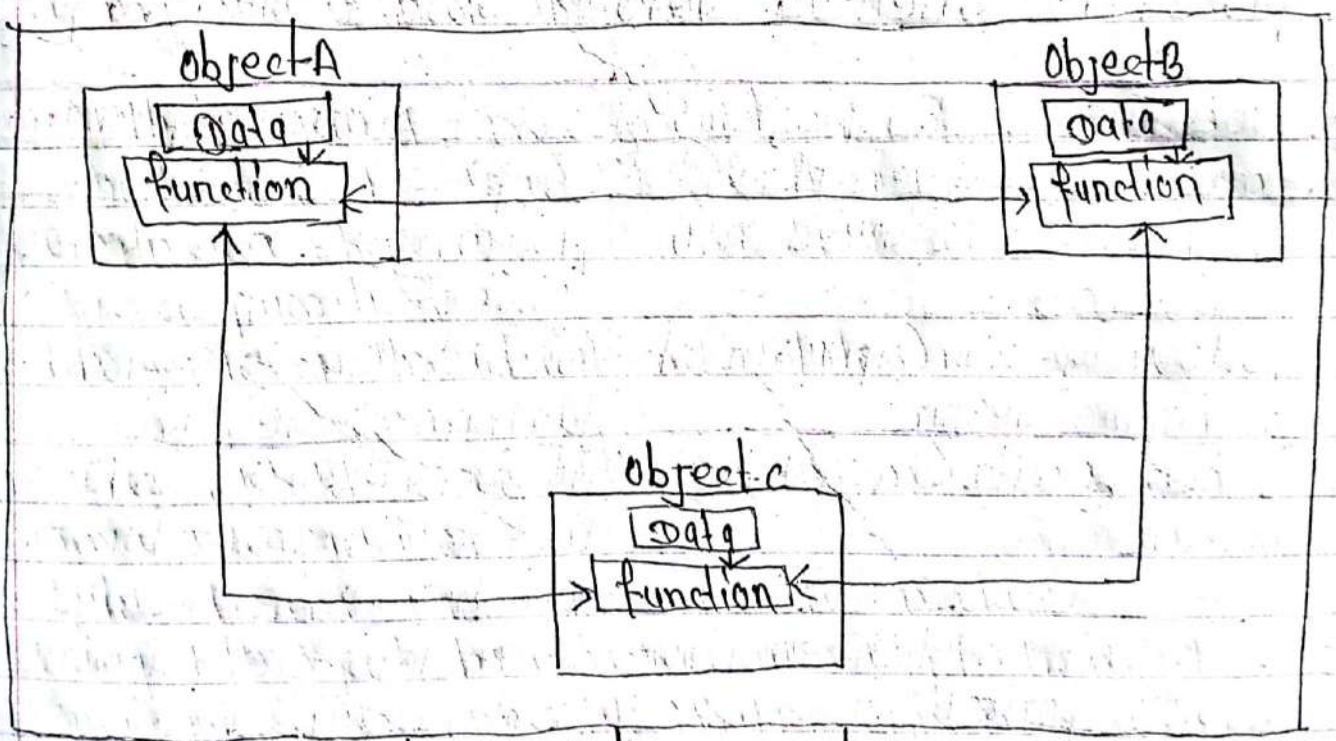


fig:- Object orienled approach.

Characteristics of oop.

Some noticeable characteristics of oop follows.

- Emphasis is on data rather than function (procedure).
- Programme are divided into object.
- Function and data are tied together in a single unit.
- Data can be hidden to prevent from accidental alternation from other function or object.
- Data can access is done through the visiable function so that communication between object is possible
- Data structure are modeled as objects
- follows buttom-up approach of program design methodology.

1.3 procedural oriented versus Object oriented programming

	Procedure oriented programming	object oriented programming
1	Emphasis is given of function	Emphasis is given on data
2	programme are dived into functions	programme are divided into object
3	follow top-down approach of programme design	follow buttom-up approach of programme design
4	Generally data can not be hidden	Generally data can be hidden so that non-member function can not access them
5	It doesn't model the real world problem perfectly.	It models the real world problem very well.
6	Data more from function to function	Data and function are tied together only related function can access them
7	maintaining and enhencing code is still difficult	maintaining and enhencing code is easy
8	code reusability is still difficult	code reusability is easy
9	for eg:- C, COBOL, FORTRAN	for eg:- JAVA, C++, gmall Talk

1.4 principle / features of object oriented programming.

a) object :- Object are entities is in object oriented system through which we perceive (arround) the world around us. The entities are represented as object in the program.

→ It can be also defined as any entity that has state (data) and behaviour is known as an object. for eg:- A chair, pen, table, keyboard, bike etc.
→ object can be physical or logical
→ An object can be defined as instances of a class.
→ It contains address and take up same memory space.
→ for example :- If we create an object of the class student and object name std.

object name :- std	Syntax
Data	Data
name	Data 1
Rollon	Data 2
marks	Data n
function	function
Total()	function-1()
Average()	function-2()
Display()	function-n()

b) Class :- A class collecting of simple type of object is called class it is a logical entity.

→ The class is the user defined data type used to declare the object.

— Class does not consume any memory space.

Syntax	Class student

class
class-name
{
Access specifier
Data members / variables
Member function
{
}
};
	Data member:
	name
	reg-number
	marks
	function member
	Total-marks()
	percentage()
	Division()

c> **Abstraction:-** Abstraction involves hiding the complex internal working of object and providing us simplified high level interface for interacting them. It focuses on what an object does rather then how it does it.

▷> **Encapsulation & Data hiding:-** The ~~mehan~~ mechanism of combining data & function together into single unit is called Encapsulation. Because of encapsulation data and its manupulating function can be kept together. By making use of encapsulation we can easily achieve abstraction.

→ The ensulation of data from direct access by the program is called data hide.

(E) Inheritance :- Inheritance is the process by which one class can inherit the attributes and methods of another class. It allows for the certain of new classes based on existing one; enabling code reuse and the modeling of "is a" relationship.

→ When one object acquires the property and behaviours of a parent object then it is known as inheritance.

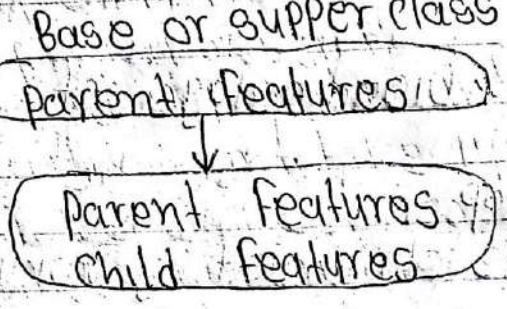


fig:- Child class inherited from parent Class.

F Polymorphism :- polomorphism is a technique which allows an object to behave in different situations.

→ polymorphism means "many form", in oop it allows objects to take on multiple forms or types. This can be achieved through function overloading (same function or method name different parameter) and function overriding.

→ eg:- A shape can be drawn with different from in different situations ie triangle, rectangle, lies.

g. **Code reusability:-** object can be reused in different part's program or mdifferent program altogether. Once a class is defined you can create multiple object from it, each with its own unique set of attribute values.

h. **modeling real world concept:-** Objects provide a structured way to represent and organize data and behaviour.

1.5 Advantage & Disadvantage of OOP:-

* Advantage of oop:-

- Reduent code is eliminated by various technique like Inheritance
- Through data hiding programmer can built secure programme.
- Existing classes can serve as library class for further ~~here~~ enhancement
- Division a programme into object makes software development easy.
- less software complexity
- Upgrading and maintenance is easily managable
- mode is real world system perfectly
- code reusability

* Disadvantage of oop:-

- compiler and run time overhead is high
- Software developer should analyzed the problem in object oriented way.
- Benifits only in long run while managing large software projects.

UNIT 2 INTRODUCTION TO JAVA

- Introduction to Java:- Java is popular programming language created in 1995 it is owned by oracle. More than 3 billion device run Java

- Application of Java:
  - Mobile Application (specially android application)
  - Desktop Application
  - Web Application
  - Web Server and Application server
  - Games
  - Database connection and manymore

- Advantages of Java:
  - Java works on different platforms (windows, MAC, LINUX etc)
  - It is easy to learn and simple to use
  - It is open source and free
  - It is secure, fast and powerful
  - It has huge community support.
  - Java is an object oriented language which gives a clear structure to programme.

2.1 Java as a programming platform

Java is widely used and versatile programming platform known for its "write once, run anywhere" (WORA) principle. It was developed by James Gosling at sun-microsystem (now owned by oracle corporation) and relized in 1995

Java is widely used programming platform known for its ability to run on various devices and operating

System. It allows developers to write code once and have it run on different machines. Java is secure, reliable, and versatile making it suitable for a ~~white~~ wide range of application from web development to mobile application. It is also known for robustness, meaning it handless error well, and it has large community and extensive library of pre-built code making development more efficient, overall Java is powerfull for creating software that can work across different devices and platforms.

2.2 History of Java: following are given significant points that describes the history of Java

- 1990's :- Java conceived by sun-microsystem's Java aimed to create a versitale programming language originally "oak" It was designed for consumer electronics.
- 1995's :- Java 1.0 released known for its "write once, run anywhere" features.
- 2000's :- Java became a standard for enterprise level application
- 2010's :- oracle acquired sun-microsystem's taking over Java's development
- 2014's :- Java it introduced lamdas and String API (Application programming interface)

- 2017's :- Java 9 brought module system for scalability
- 2018's :- Java 11 marked a longterm support release
- present's Java continuous evolving with regular update diverserange of application.

2.3 Java Buzzwords: following are the Java Buzzwords or features.

- Simple:- Java is very easy to learn and it's syntax is simple, clean and easy to understand to c++ it make familer to developer.
- Object oriented:- Java is a object oriented programming language because everything in a Java is an object i.e. Java supports object oriented features such as classes Object inheritance, polymorphism composition.
- Distributed:- Java includes features for network programming enable development of application that can communicate over a network.
- Robust:- Java was built to be robust with features like automatic memory management (garbage collection) to prevent common programming errors.
- Secure:- Java is used to creates virus free, temper free software system therefore it is secure.

- **Architecture neutral :-** Java programs can run any device and operating system with a JVM (Java virtual machine) making it platform independent.
- **portable :-** Java's ability to on different devices and operating system without modification its portability.
- **High performance :-** Java aimed to provide high performance execution through techniques like Just-in-time (JIT) compilation and adaptive optimization.

→ Java program can also have multiple thread of execution that increase the execution speed.

- **Multithread :-** Java supports concurrent programming allowing multiple-thread to execute simultaneously usly.
- **Dynamic :-** Java programming allows classes to be loaded dynamically at run time, providing flexibility and extensibility in application.
- **Interpreted-compiled :-** Java initially used an interpreter for execution but modern Java implementation use a combination of interpretation and Just-in-time compilation for improved performance.
- **Garbage collected :-** Java introduced automatic memory

management which helps prevent memory leaks and simplifies memory allocations.

2.4 **Java virtual machine :-** Java virtual machine as an abstract machine It is a specification that provide run time enviroment in which Java bytecode can be executed.

→ The JVM performed following operation

- loads code
- verifies code.
- Executes code
- provides run time enviroment

JVM Architecture:

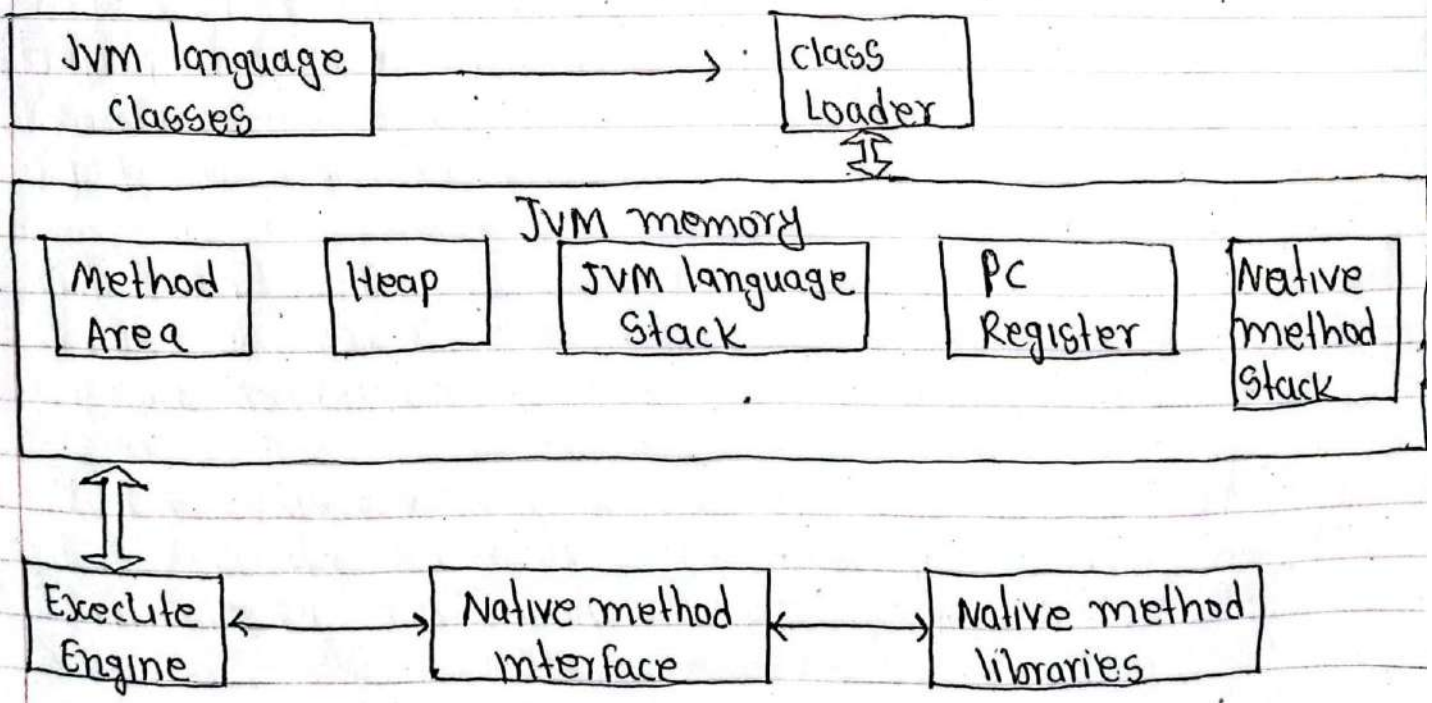


fig:- JVM Architecture

13

- Class loader:- Responsible for Java classes into memory. There are different types of class loader such as boot-Strack class loader, extension class loader and Application class loader. Each responsible for loading classes from specific sources

UNIT 3

FUNDAMENTAL PROGRAMMING STRUCTURE

3.1 Whitespace, Identifiers, Literals, comments, separators & keyword:

- Whitespace :- White space refers to characters that are used for formating and are not typically not visible in the output.

→ Whitespace is used to enhance the readability of code for human & ignored by the Java compiler.

Some common use of White-space in Java:

* spaces :- Spaces are used to separate words, operators elements in a statement or expression. for eg: int x=10;
* Tabs :- Tabs are used to indent code blocks to improve readability and maintain a constant structure for eg of condition

```
{
    // Indented code block
}
```

* Lines break :- Line breaks separate different line of code and are used to structure the program

for example

```
System.out.printer("Hello");
```

* Comments :- Comment which start with // for single line comment or /* */ for multi-line comments are also considered whitespace.
* Empty lines :- Empty lines containing only white space characte are used to separate block of code or improve readability

for eg:

```
int a=10;
int y=20;
```

- Identifier :- All Java Variable must be identified with unique name these unique names are called identifier.

→ Identifier is name given to variable method, class or other program elements.

The general rules for naming variable are.

* Names can contain letters, digits, underscores and dollar signs
* Names must begin with a letter.
* Names should start with a lowercase letter and it can not contain white space,
* Names can also begin with $ sign
* Reserved keywords can not be used as names
* Variables are case sensitive (my and my var are different variables)

for eg:

```
int n; float x;
```

- Literals :- literals refers to a fixed value that directy written in source code. These values can be of various types such as integers, floating point number, characters, strings and boolean values

Types of literals

* Integer literals : for example int num = 42;
  - 42 is an integer leteral.
* Floating point literals: for example : double pi = 3.14;
  - 3.14 is a floating point literals.
* Character literals :- for example char grade = 'A';
  - A is a character literal.
* String literals :- for example String message = "Hello world";
  - "Hello world" is a string literals

* Boolean literals :- Example boolean is true = true;
  • true is boolean literals
* Null literals :- Example Object obj = Null;
  • Null represent the absence of a value

* Integer literals (in different bases) :- for example
  Binary (prefix ~~zero 0~~ 0b : int binary 0b1010;
  equivalent to decimal 10)
* Floating point literals (with f for f suffix for float):
  example : float num = 3.14f
* Scientific Notation :-
  eg: double sciNum = 2.5e3
  (equivalent to 25000)

Remember literals are constant values and cannot be changed during programs executions.

• Comments :- Comments can be used to explain Java code and to make it more readable It can also be used to prevent execution when testing alternative code.
* Single line comments :- Single line comments start with two forward slashes (//). Any text between // and the end of line is ignored by Java
  for eg: // This is comment.

* multiline comment :- ~~multline~~ multiline comment start with /* and ends with */ any text between /* and */ will be ignored by Java
  for eg: /* This is comment */

- Separators :- programming "separators" refers to special character or symbols used to delimits different elements within the code.

→ They help the compiler or interpretor understand the structure of program.

→ Here are some common separators in programming

(a) Semicolon (;) (b) comma (,) :- Separate item used to terminate in least statement.

b) parenthesis (()) :→ used to enclose parenthesis in function or method calls.
d) Braces ({}) :- used to define block of code
e) Square braces ([]) :- used to denote arrays in many programming language
f) Angle Brackets (<>) :- The are used for genetic types.
g) period (.) :- used to access members & object of class.
h) colon (:) :- used to various contexts such as for each loops or conditional statement.
i) question mark ("or") :- used to denote or.
j) Vertical bar (|) : Acts as an OR operator
k) Slash (/) :- used for division or comments.
l) Backslash (\) : used to in escape sequence.
m) Ampersand (&) :- Acts as an AND operator
n) Double colon (::) : used for method references (eg Java)

- Keywords :- Java has a set of keywords that are reserved words that can not be used as variables, methods, classes or any other identifier.

→ They are 51 reserveds words in Java

① Abstract	12 do	24 interface	35 short	46 try
2 Assert	13 double	25 long	36 static	47 void
3 Boolean	14 else	26 native	37 strictfp	48 Volatile
4 Break	15 enum	27 new	38 super	49 while
5 Byte	16 final	28 null	39 switch	50 true
6 case	17 finally	29 package	40 goto	51 false
7 catch	18 float	30 const	41 synchronized	
8 Char	19 for	31 private	42 this	
9 class	21 implements	32 protected	43 throw	
10 continue	22 if	33 public	44 throws	
11 diffult	23 int	34 return	45 transient	

→ Simple Java program that points "Hello world" and brief explanation.

Answer

```
public class Helloworld
{
public static void main (string[] args)
{
system.out.println ("Hello world");
}
}
```

Explanation:

public class Helloworld :- This declares a class name Hello world.

public static void main (String[] args) :- This declares the main method. It is starting point of execution for Javaprogram. It takes an arrays of String (args) as parameter but is not used in this example

3.2 Data type and conversion

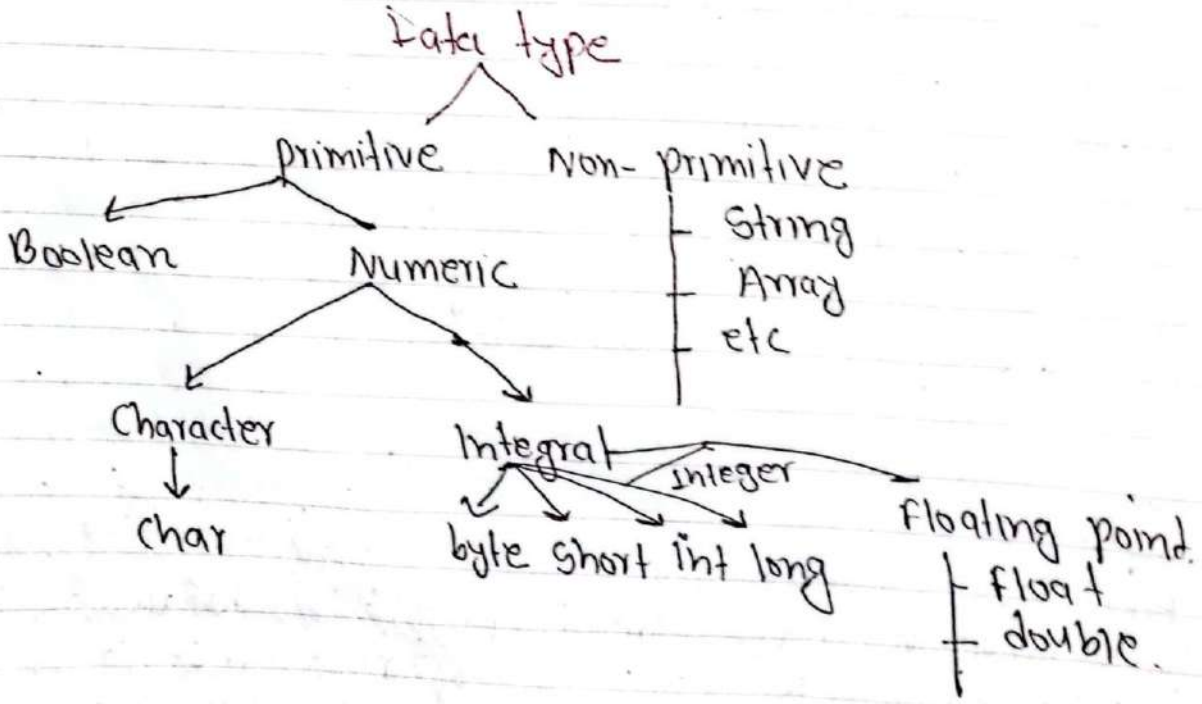
→ Data types specifys the different sizes and values that can be stored in the variable.

→ There are two types of data types in Java

* primitive data ~~Java~~ types :- The primitive data types include boolean, char, byte, short, int, long, float and double.

→ They are the most basic data types directly supported by Java. They represent simple values and are not objects.

* Non-primitive data :- These data types are used to create objects and work with more complex data. Reference types include classes, arrays etc.



Data type	Default value	Default size
Boolean	false	1 bit
char	'\u0000'	2 byte
Byte	0	1 byte
Short	0	2 byte
Int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
Double	0.0d	8 byte

⇒ Data type conversion (Type-casting)

→ Datatype conversion refers to the process of converting a value from one data type to another.

→ There are two types of data type conversion.

a) Implicit conversion (Automatic or widening) :- This happens automatically when you assign a value of a smaller data type to larger data type

→ There is no risk of data loss during implicit conversion for eg:

```
int intvalue = 42;
double doublevalue = intvalue;
```

→ byte → short → char → int → long → float → double

b) Explicit conversion (Narrowing or manual) :- These requires manual intervention and is needed when you want to convert a value from a larger data type to a smaller one.

→ There is a risk of data loss during explicit conversion, so it should be done carefully.

```
double doublevalue = 3.14
Int intvalue = (int) doublevalue;
```

c i String to other types:- converting a String to other data type is common you can use method like parse Int, parse Double etc to achieve this
for eg:

```
String number = "123";
Int number = Integer parseIn(numberstr);
convert string to int
```

(ii) Other types to String:- You can convert other types to a String using String value of () or concatenation with an empty string.

3.3 Variables:- A variable is a named storage location that holds data, and its value can be changed during the program execution.

→ Variable must be declared before they are used. This involves specifying the variable datatype and giving it a name.

→ Java is a statically typed language meaning each variable must be declared with a specific data type (eg int, double, String)

- Naming Rule
  * variable are case sensitive (myvar and Myvar are different variable)
  * Names can contain letters digits underscore (_) and dollar sign ($).
  * Names must begin with a lowercase letter.
  * Name can also begins with dollar sign
  * It cannot contain whitespace.
  * Reserved keywords cannot be used as name..

3.4 Constant (final variable):- Constant are non-modifiable (immutable) variables declared with keyword final you can only assign values to final variables once. Their value cannot be changed during program execution.

for example:

```
final double pi= 3.14159; // Declare and initialize constant
final int screen_x_max= 800; // compilation error: cannot assign
                                value to final variable
final int screen_y_min;
          screen_y_min =0;
```

3.5 Operators

→ operators in java is a symbol which is used to perform operation for eg: +, −, *, / etc

→ There are many types of operators in Java which are given below

a) Java unary operator

→ unary operators requires only one operands unary operator are used to perform various operations i.e incrementing, decrementing a value by one, negating an expression, inverting the value of boolean.

eg:

```
class unaryoperator
{
public static void main (string[] args);
 {
 int x= 10;
System.out.println(x++); // 10(11)
System.out.println(++x); // 12
```

```
System.out.println(x--)  // 12 (11)
System.out.println(--x)  // 10
  }
}
```

output 10, 12, 12, 10

eg2

```
 Class unaryoperator
{
Public Static voidmain (String [] args)
{
 int a= 10;
 int b= 10;
System.out.println (a++ + ++a); // 10+12 = 22
System.out.println (b++ + b++); // 10+11 = 21
}
 }
```

output 22
        21

eg:-3

```
 Class unaryoperator
{
public Static voidmain (String args[])
{
   int a=10;
    int b= -10;
  boolean c = true;
  boolean d = false
System.out.println (~a); //-11
System.out.println (~b); // 9
```

```
System.out.println(!c);  /* false/opposite
System.out.println(!d)   // True
 }
}
```

output -11
9
false
True

b Java arithmetic operators :- Java arithmetic operators are used to perform addition, subtraction, multiplication and division. They acts as basic mathematical operations.
for example

```
Class Arithmetic operators
   {
public static void main (String [] args)
   {
   int a=10;
   int b=5;
System.out.println(a+b); //15
System.out.println(a-b); //5
System.out.println(a*b); //50
System.out.println(a/b); //2
System.out.println(a%b; //0
 }
}
```

Output 15
5
50
2
0

c Shift operators:

i) left shift operator:

The Java left shift operator << is used to shift all of the bits in a value to the left-side of the specified number of times

eg:

```
class Leftshift {
public static void main (string [] args)
{
System.out.ln (10<<2); // 10 * 2^2 = 40
System.out.ln (20<<3); // 20 x 2^3 = 160
System.out.ln (15<<4); // 15 x 2^4 = 240
}
}
```

output

ii) Java Right Shift operator:

The Java right shift operator >> is used to move left operand value to right by the number of bits specified by the right operand.

eg:

```
class Rightshift
{
public static void main (string [] args)
{
System.out.println (10>>2); 10 / 2^2 = 2
System.out.println (20>>2); 20 / 2^2 = 5
System.out.println (20>>3); 20 / 2^3 = 2
}
}
```

D Logical operator :-

a) Java AND operator (&&) :- The logical && operator doesnot check second condition only if first condition is true.
→ If the left operand is false, the right operand is not evaluated.

b) Logical OR (||) :- It evaluates to true if at least one operand is true.
→ If the left operand is true, the right operand is not evaluated.

c) Logical NOT (!) :- Inverts the truth value the operand (true becomes false and vice versa)

eg.

```
public class logicaloperator
{
public static voidmain (string [] args)
 {
     // logical AND
boolean is sunny = true;
boolean is warm = true;
if (is sunny && is warm)
{
system.out.println(" It's a good day for picnic");
}
else
  {
  System.out.println ("may be another day of picnic");
  }
// Logical OR (||)
boolean is Rainy = false;
Boolean is umberella = true;
```

```
if (is Rainy has umbrella)
{
System.out.println ("you are prepared for the whether");
}
else
{
System.out.println ("consider taking an umbrella");
}
        // Logical Not
boolean is Evening = false;
If ( ! is Evening)
{
System.out.println(" It's not evening yet.");
}
else
{
system.out.println (" It's ~~not~~ already evening.");
}
}
}
```

E Bitwise operator :-

* Bitwise AND (&) :- It performs a bitwise AND operation between corresponding bits of operands

* Bitwise OR (|) :- It perform a bitwise OR operation between corresponding bits of operands

* Bitwise XOR (^) :- It perform a bitwise XOR (exclusive OR) between corresponding bits of operands -

* Bitwise NOT (~) :- Inverts the bits of the operand.

F Java Ternary operator :- Java Ternary operator is used as one linear replacement for if then else statement and used a lot in Java programming. It is the only conditional operator which takes three operands.

eg:
```
Class Ternary operator
{
public static void main (String [] args);
{
int a=2;
int b=5;
int min = (a<5)? a:b;
System.out.println(min);
}
}
```
output :-2

G: Java assignment operator :- Java assignment operator is one of the most common operator. It is used to assign a value on its right to the operand on its left

for eg:
```
Class operatorexample
{
public Static void main(String [] args);
{
int a=10;
int b= 20;
a+=4; // a=a+4 (a=10+4)
b-=4; // b=b-4 (b=20-4)
System.out.println(a);
System.out.println(b);
}
}
```

eg:-2

```
Class operator example
{
public static void main (string [] args);
{
int a=10;
a+=3; // 10+3
System.out.println(a);
a-=4  //13-4
System.out.println(a);
a*=2; // 9*2=18
System.out.println(a);
a/=2;  // 18/2 = 9
System.out.println(a);
}
 }
```

3.6 Control Structure :- Java provides several control statement to manage the flow of a program the main include.

a) Decision making Statement:

i Java if-else statement:

→ The Java if statement is used to test the condition It checks boolean condition (true or false). There are various types of if statement in Java

- If statement
- If-else statement
- If-else ladder
- nested if statement

a) Java if statement:- The Java if statement tests the condition It executes the if block if condition is true.

Syntax:

```
if (condition)
{
    // code to be executed
}
```

example

```
public class example
{
public static void main (String [] args)
{
int age=20; // defining age variable
if (age>18)
{
System.out.println ("Age is greater than 18");
}
}
}
```

o/p :- Age is greater than 18

b) Java if-else statement :- The Java if-else statement also tests condition It executes if block if condition is true.

Syntax:

```
if (condition)
{
    // code of condition is true.
}
else
{
    // code if condition is false
}
```

example program to check odd or even number

```
public class ifelse example
{
public static void main (String [] args);
{
  int number = 13;
if (number % 2 == 0)
{
 System. out. println ("Even number");
}
else
{
 system. out. println ("odd number");
}
}
}
```

c Java if-else-if ladder statement:- The if-else-if ladder executes one condition from multiple statements.

Syntax

```
if (condition 1)
 {
    // code to be execute if condition 1 is true
}
else if (condition 2)
{
    // code to be execute if condition 2 is true
}
else if (condition 3)
{   // code to be execute if condition 3 is true
}
```

```
else
{
  // code to be executed if all the condition are false
}
}
```

example program to Grading System for fail, D grade, Cgrade, Bgrade, Agrade and A+ grade

```
public class If else if example
{
public static void main (String[] args)
{
  int marks = 65;
if (marks < 50)
 {
system.out.println ("fail");
}
else if (marks >=50 && marks < 60)
{
system.out.println ("D grade");
}
else if (marks >= 60 && marks < 70)
{
System.out.println (" C grade");
}
else if (marks >= 70 && marks < 80)
{
   system.out.println (" B grade");
}
else if (marks >= 80 && marks < 90)
}
```

```
System.out.println("A grade");
}
else if (marks >= 90 && marks < 100)
{
System.out.println("A+ grade");
}
else
{
  System.out.println("Invalid");
}
}
}
```

d> Java Nested if Statement

→ The nested if statement represent if block within another if block here the inner if block condition executes only when outer if block condition is true.

Syntax

```
if (condition)
{
  // code be execute
if (condition
{
  // code to execute
}
}
```

example

```
public class JavaNestedexample {
public static void main (String [] args)
{
```

```
int age=20;
int weight=18;
if (age>=18)
{
  if (weight>50)
  {
System.out.println ("you are eligible to donate blood");
}
}
}
  }
```

Write a program to check positive, Negative or zero

```
import Java.util.Scanner;
public class NumberAnalyzer
{
public static voidmain (String[] args)
{
  // create a scanner object to read input
Scanner Scanner = new scanner (System.in)
System.out.println ("Enter a number:");
int number = Scanner nextint();
 if (number > 0)
 {
System.out.println ("The number is positive");
}
else if (number <0 );
{
System.out.println ("The number is Negative");
}
```

```
else
{
System.out.println("The number is zero");
}
Scanner.close();
}
}
```

NOTE

```
To Read Float
Float floatvalue = Scanner.nextfloat();
To Read double
double userinput = Scanner.nextdouble();
to Read string
String input1 = Scanner.nextLine();
to read array
int size = Scanner.nextInt();
int[] numbers = new int[size];
```

(II) Switch Statement:

The Java switch statement executes one statement from multiple condition. It is like a if-else ladder statement.

Syntax

```
Switch (expression)
{
case value1:
  // code to be executed:
```

```
break; //optional
case value 2:
 // code to be executed
break;
----
default
// code to be executed if all cases are not match
}
```

Example

```
public class switchexample
{
public static void main (String[] args)
{
int number =20;  // switch expression
Switch (number)
 {
case 1; System.out.println ("10");
break;
case 2; System.out.println ("20");
break;
case 3: system.out.println ("30");
break;
default : System.out.println ("Not in 1,2,&3");
}
}
}
```

(B)

3.7 Loop :- In Java, we have three types of loops that execute similarly However there are difference in their Syntax and condition checking time.

a) Java for loop :- In Java for loop is similar to C & C++. It enables use to initialize the loop variable, check the Condition and increment / decrement in a single line of code.

Syntax:

```
For (initialization; condition; increment /decrement )
  {
    // Block of Statement
  }
```

Example:

```
public class calculation
{
public static void main (String[] args)
{
 int sum=0;
for (int i=0; i<=10 ; i++)
{
 sum = sum+i;
}
System.out.println ("The sum of first 10 natural
 number is , "+sum);
 }
 }
```

ii Java do-while loop :- The do-while loop checks the condition of the end of the loop after executing the loop statement. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

→ It is also known as exist controlled loop since the condition is not checked in advance.

Syntax:

```
do
 {
  // statements
 }
 while (condition);
```

For example

```
public class calculation
 {
public static void main (String [] args)
 {
  int i= 0;
System.out.println ("printing the list of first 10 even number\n");

do
{
 System.out.println (i);
  i= i+2;
}
  while (i<=10);
}
  }
```

(iii) Java while loop :- The while loop is also used to iterate over the number of statements multiple time. However if we don't known the number of iterations in advance, it is recommended to use a while loop.

Syntax:-

```
while (condition)
{
  // looping statements
}
```

→ It is also known as entry control loop.

example

```
public class calculation
{
public static void main (String[] args)
 {
 int i=0
System.out.println (" printing the list of first even
                      numbers /n" );
while (i<=10)
 {
System.out.println (i);
i= i+2;
}
}
```

c Jump statement :- They are used to transfer control of the program to the specific statements.

- Break statement
- continue statement

i) Break statement :- The break statement is used to break the current flow the program and transfer the control the next statement outside a loop or switch statement.

example

```
public class Breakexample
{
public static void main (String [] args)
{
for (int i=0; i<=10; i++)
{
System.out.println(i);
if (i==6)
 {
 break;
}
}
}
```

O/P:-

```
0
1
2
3
4
5
6
```

11) Java continue Statement:- Unlike break statement, the continue statement doesnot break the loop where as it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

for example:

```
public class continueexample {
public static void main (string [] args)
{
  for (int i=0; i<=2; i++)
  {
for (int j=i; j<=5; j++)
{
 if (j==4)
{
continue;
}
System.out.println(j);
}
}
}
```

o/p

0
1
2
3
5
1
2
3
5
1
2
3
5

(a) Write a programme ~~two~~ add any two number, input ~~any two num~~ from keyword

```
import Java.Still.Scanner;
public class AddNum
{
public static Voidmain (String[~~args~~] args)
{
 int sum=0;
Scanner scan=new Scanner(system.in);
System.out.println("Enter any two number");
int n1 = scan.nextInt();
int n2 = scan.nextInt();
Sum=n1+n2;
System.out.println(sum);
}
}
```

b) Write a programme to calculate simple Interest

→

```
import Java.util.Scanner;
public class SimpleInterest {
public static void main (String[] args)
  {
float p, r, t, SI;
Scanner s= new Scanner (System.in);
System.out.println("Enter the principal :");
p= s.nextFloat();
System.out.println("Enter the Rate of interest...");
r= s.nextfloat();
```

```
System.out.println("Enter the time period ---:");
t= s.nextFloat();
SI= (p*t*r)/100;
System.out.println("simple interest is:" +SI);
}}
```

3.9 Methods:

→ A method is a block of code which only runs when it is called. you can pass Data known as parameters, into a method.

→ methods are use to perform certain actions and they are also known as function.

→ A method mustbe declare within a class. It is defined with the name of the method followed by parenthesis ().

→ Syntax : Void method-name();
Access specifier return-type function-name();

Call a method:

→ To call a method in Java, write the methods name followed by two parenthesis() and a semicoln.

eg:-
```
public class Main{
{
```

```
static void mymethod()
{
System.out.println("I Just got executed");
}
public static void main(String[] args)
{
mymethod();
}
}
```

o/p I Just got execute

Method parameter

→ Ans: page 59:

Method argument:

→ These are the actual values that are passed to a method when it is called method argument

3.10 Arrays:

→ An array is a collection of similar type of element which has contiguous memory location

→ Java array is an object which contains element of a similar data type.

→ We can stored only a fixed set of elements in a Java array

→ Syntax dataType[] arr; or
dataType arr[];

→ Instantiation of an Array in Java
arrayRefVar = new datatype[size];

eg:-
```
Class Test
{
public Static void main(String[] args)
  {
  Int a[] = new int [5]; // declaration and instatition
   a[0] = 10;   // initialization
   a[1] = 20;
   a[2] = 70;
for (int i=0 ; i<a.length; i++)
 System.out.println(a[i]);
}
}
```

O/P
```
10
20
70
```

UNIT 4

CLASSES AND OBJECTS

4.1 Concept of JAva Classes and Objects

→ Classes and objects are the two most essential that Java concepts that every programmer must learn.

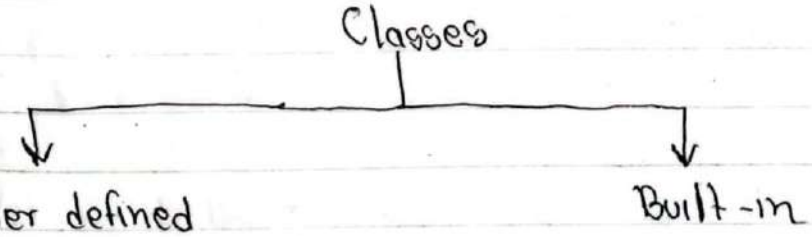
→ Classes and object are closely related and work together. An object has behaviour and states and is an instance of class. for instance, a cat is an object it's color and size are states and its meowing and clawing for miture are behaviours.

→ A class models the object, a blueprint or template that describes the state and behaviour supported by objects of that type.

→ class can be defined as a container that stores data members and method together. These data members and method are common all the objects present in particular package.

Types of Classes

In Java, we classify classes in two types



Built in classes :- Built in classes are the pre defined classes that come along with the Java development kit (JDK). These built in libraries classes libraries some example of built-in class include.

Java.lang.system
Java.util.Date
Java.lang.thread

Lots of classes and method are already predefined by the time you start writing your own code.
Libraries :- collection of package
package :- Contain several classes
class :- Contain several method
method :- A set of instruction

(ii) User defined classes :- User defined classes are rather self-explanatory. The name says it all. They are the classes that the user defines and manipulates in the real time programming. User defined classes are broken down in to three types.

a) Concrete class :- Concrete class in just another standard class that the user defines and stores the methods and data members in

Syntax

```
Class con
{
    // class body;
}
```

b) abstract class :- Abstract classes are similer to concrete class, except that you need to define them using the "abstract" keyword

Syntax

```
abstract class Abstclas
  {
  //method();
  abstract void demo();
  }
```

c Interfaces :- Interfaces are similar to classes the difference is that while class describes an objects attributes and behaviours. Interfaces contain the behaviours a class implements

Syntax

```
public interface demo {
public void Signature1();
public void Signature2();
}
public class demo2 implements demo {
public void Signature1()
  {
// Implementation
}
  public void Signature2()
    // Implementations;
  }
}
```

— Rules for creating class

* The following rules are mandatory when you are working with Java classes :

<i> The keyword "class" must be used to declare a class.

<ii> Every class name should start with an uppercase characters and if you intend to include multiple words in a class name make sure you use the camel case.

<iii> A Java project can contain any number of default classes but shouldnot hold more than one public class.

(iv) You should not use special character when naming classes.

(v) You can implement multiple interfaces by writing their names infront of the class separated by commas.

(vi) You should expect a Java class to inherit only one parent class.

Object:

An object in Java is the most fundamental unit of the oop paradigm. It includes real world entities and manipulates them by invoking methods. An object in Java programming consists of the following:

a) Identity:- This is the unique name given by the user that allows it to interact with other objects in the project.
eg:- name of the student.

b) Behaviour:- The behaviour of an object is the method that you declare inside it. This method interacts with other objects present in the project. eg:- studying, playing, writing,

c) State:- The parameter present in an object represent. Its state based on the properties reflected by the parameters of other object in the project.
eg:- section, Rollnumber, percentage.

(4.7) * Create an object / class: An object is created from a class. Create an object of myobj and print the value of x;

Syntax:

```
class-name object_name = new.class-name ();
```

4.8 # Accessing class member:

```
public class main
  {
  int x=5;
public static void main(String[] args)
  {
main my-obJ = new main();
System.out.println(my-obJ.x);
}
}
```

Multiple objects:

We can create multiple objects of one class for eg:- from above program: main myobJ1 = new main();
main myobJ2 = new main();

- Using multiple classes you can also create an object of a class and access it in another class. This is often used for better organization of classes.
Remember that the name of the Java file should match the class name. In this example we have created two files in the same directory

/ Folder:
main.Java
Second.Java

main.Java

```
public class main {
  int x=5;
}
```

Second.java

```
Class Second
{
public static void main(String[] args)
{
main my obj = new main();
System.out.println (my obj.x);
}
}
```

When both files have been compiled Run the Second.java file and the output will be : 5

- Key differance between Java classes and object

Class	object
-A class is a blueprint for creating object.	An object is a copy of a class.
-A class is a logical entity	- An object is a physical entity
- the keyword used is "class"	- The keyword used is new.
- A class is designed or declared only once	- you can create any number of object using one single class
- The computer doesnot allocate memory when you declare a class	The computer allocate any number of objects using one single class

4.2 Adding variable:- An adding variables refers to the process of declaring initializing and performing operations with variable in your Java programs.

→ To add variables in Java you declare them with a specific data type

for eg:

```
public class Adding var
{
public static void main (String[] args)
{
 int num1= 5;
 int num2 = 10;
 Int sum= num1 +num2;
 System.out.println ("sum:" + sum);
 }
}
```

O/P: sum: 15

1.3 Adding methods: We can add more method with different functionality based on our programs requirements. methods enhance code organization, reusability and readability.

```
public class calculation
{
public static voidmain (String[] args)
{
Int S= sum (5,10)
```

```
double d= multiply (2.5, 3.0);

System. out. println ("sum :" +s);
System. out. println ("product:" +d);
}
public static int sum (int num1, int num2)
 {
  return num1+num2;
 }
public static double multiply ( double num1, double num2)
 {
   return num1 * num2;
 }
```

4.4 Statics variable, methods, Block and class

Statics variable :- Statics variable is a class level variable that is associated with the class rather than with instances of the class. It is shared among all instances of the class and can be accessed using the class name or an instances of the class.

```
public class Example
{
Static  int n = 42;
public static void main (String [] args)
{
System. out. println (" static variable:" + n);
}
}
```

o/p static variable : 42

Static methods:

→ static methods in Java are methods that belong to the class rather than instance of class. They are associated with the class itself not with any specific object created from the class. for example :-

```
public class main {
static void mystaticmethod ()
  {
System.out.println ("static methods can be called without
  creating objects");
}
// for public method:
public void mypublic method ()
 {
system.out.println (" public methods must be called by
 creating object");
// main method
public static voidmain (string [] args)
 {
my staticmethod ();
main m = new main();
  m.my publicmethod ();
}
}
```

Static block :-

→ A static block in Java is a block of code code enclosed within curly braces. They are executed only one once when the class is loaded.

→ They are written using the static {---}

Syntax within the class.

for eg:

```
public class Example {
  static int n;
  static
 {
System. out. println ("static block executed");
    n=42;
 }
public static void main(String[] args)
 {
System. out. println (" static variable :" +n);
 }
}
```

Static class :- In Java, the term static class usually refers to a static nested class, which is a class declared inside another class and marked as static.

for eg

```
public class outerclass {
 static class Innerclass
    {
```

```
void nested method ()
{
System.out.println("method inside static inner class");
}
}
public static void main (String[] args)
  {
Innerclass Ic = new Innerclass();
    Ic.nestedmethod();
 }
}
```

4.5 Access Specifier (Access control) :- Access control in Java refers to the mechanism that determine how different parts of your Java program can interact with each other

→ Here are the main access modifiers in Java

i) public :- Accessible from anywhare

→ No access restriction

for eg :- public class myclass { }

ii) private :- Accessible only within the same class

for eg :- private int myvariable { }

iii) protected :- Accessible within the same class, sub class & package.

for eg :- protected void mymethod () { }

iv Default :- package - private, No modifiers

for eg :- int n;

→ Access modifiers can be applied to class interface, methods, and variable.

24.6 Method parameters :- method parameters are variable listed in the method signature and acts as placeholders for the values that will be provided when the method is called method parameters.

→ We can pass single or multiple parameter through Java function.

→ for example:-

```
Public class Javamethodparameter
 {
Static void mymethod (String fname , int age)
{
System.out.println (fname + "is" + age);
public static voidmain(String [] args )
  {
    mymethod (" yogesh", 5);
    mymethod (" Ramesh", 3);
 }
}
```

O/P Yogesh is 5
Ramesh is 3

4.7 & 4.8 are page no. 51 & 52

4.9 Setters and Getters :- Getters and setters are methods used to access and modify the private fields (member variable) of a class.

(i) Getter method (get variable name)

→ A getter method retrieves the value of a private field
→ It's named with the prefix "get" followed by the name of the field with the first letter capatalized.
→ It has no parameters and returns the value of the associated field.

(ii) Setters method :-

→ A setter method updates the value of a private field.
→ It is named with the prefix "set" followed by the name of the field with the first letter capitalized.
→ It takes a parameter representing the new value to be set for the associated field.

for example

```
public class person
{
private String name;
public string getName()   // Getter 'Name'
{
 return name;
}
public void setName (String newName) // setter 'Name
  {
    name = NewName;
  }
}
```

```
public static void main(String[] args)
{
person p = new person();
p.setName (" paras khadka);
String retrievedname = p.getName();
System.out.println ("Name: " + retrievename);
}
}
```

o/p Name: paras khadka

4.10 constructors:- A constructor in Java is a special method that is used to initialize objects. The constructors is called when an object of a class is created.

→ It can be used to set initial values for object attributes.

→ constructors parameter:- constructors can also take a parameters which is used to initialize attributes.

For example: without parameter vs with parameter

[Next page]

constructor without parameter	construct with parameter

public class constExm
{
int x;
public constExam()
{
x=5;
}
public static void main(string [] args)
constExam S= new(constExam()) ;
System.out.println (S.x);
}
}
	public class const.Examp
	{
	int x
	public const_EXMP (int y) {
	x = y;
	}
	Public static void main (String[] args)
	const-EXMP S=new const.Exmp(5);
	System. out.println (S.x);
	}
	}

4.11 Overloading method:

method overloading is a features in Java that allows a class to have multiple methods having the same name if their parameters lists are different. It is related to Compile time (or static) polymorphism.

for example
overloaded Example to add integers

```
public class Addition_integer
{
public int add (int a, int b)
{
 System.out.println(" adding two integers");
 return a+b;
}
```

```
public int add (int a, int b, int c)
{
System.out.println ("Adding three integers");
return a+b+c;
}
public static void main (String[] args)
{
Addition_integer s=new Addition_integer ();
int Sum_Int = s.add (3,5);
int Sum = s.add (3,4,5);
System.out.println ("Sum of two integer" +Sum_Int);
System.out.println ("Sum of three integers" +Sum);
}
}
```

- condition for method overloading:
* The methods must be in the same name.
* The methods must be in the same class
* The methods must have different parameter lists. This can involve a different number of parameter, different types of parameters or both.

4.12 Call by Value, Call by reference

Call by value :- call by value is a parameter passing mechanism in programming language.
→ In this mechanism the actual value of the argument is passed to the method or function.

This means that changes made to the parameter inside the method donot affect the original value of the argument.

eg:

```
public class callbyvalue
 {
public static int add (int a, int b)
 {
return a+b;
 }
public static void main (string[] args)
 {
 int c= add (3,7); // calling the method
System. out. println (" Result of addition" +c);
}
 }
```

output ·· Result of addition 10

call by reference :- Call by reference is a term used to ~~address~~ describes a method of passing arguments to a function or method where the actual memory address (reference) to the variable is passed to the function.

for eg:-

```
public class Reference
 {
static int add (int x, int y)
 {
return x+y;
 }
```

```
public static void main (String [] args)
  {
  int a=15;
  int b= 83;
System. out. println ("The sum of two number:"
                        +add (a, b));
    }
}
```

O/P: The sum of two number: 98

⇒ In this main() method, we call the add() method by passing parameters that refer to the address of the formal parameter. Here 'a' and 'b' are the reference parameter for variables "x" and "y"

4.13 This keyword :-

→ This keyword is a reference variable referring to the current objects.

→ This is used in the constructors to distinguish between the instance variable value and the constructors parameter value.

→ It is primarily used to differentiate instance variable from local variables when they have the same name.

for eg:-

```
public class AddExample
{
    private int result;
public void add(int a, int b)
{
  this.result = a+b; /* This is used to distongous instance
                        variable from parameter */
}
public int getResult()
 {
 return this.result;
 }
public static void main(String[] args)
 {
AddExample adder = new AddExample();
adder.add(5,7);
System.out.println("final result :" +adder.getResult());
}
   }
```

4.14 final Modifier:

→ The result keyword in Java is used to restrict the user the Java final keyword can be used in many context.
final can be,

* Variable
* method
* class

→ The final keyword can be applied with the variable. A final variable that have no value it is called blank final variable or uninitialized final variable.

* Final variable:- If you make any variable as final, you cannot change the value of final variable (It will be constant)

* Java final method: If you make any method as final, you cannot override it.

for eg:-

```
Class Bike
{
final void run()
{
System.out.println("running");
}
}
class Honda extends Bike
{
void run()
{
System.out.println("Running safely with 40 kmph");
}
public static void main (String arg[])
{
Honda honda = new Honda();
honda.run();
}
}
```

o/p:- compile time error.

- Using Java Variable

```
class Bike
{
final int speedlimit = 90;   // final variable
void run()
{
speedlimit = 400;
}
public static void main (String [] args)
{
Bike b = new Bike();
b.run()
}
}
```

O/P :- compile time error

* Java final class : If you make any class as final, you cannot extend.

for example:

```
final class Bike { }
class Honda1 extends Bike
{
void run()
{
System.out.println("Running safely with 40 kmph");
public static void main (String args[])
{
Honda1 honda = new Honda1();
honda.run();
}
}
```

O/P :- compile time error

4.15 Nested class:- In Java a class can be defined within another class and such classes are known as nested class. These classes help you to logically group classes that are only used in one place.

→ This increases the use of encapsulation and create a more readable and maintainable code

For eg:

```
class outerclass
{
 int num;
private class innerclass
 {
public void print()
 {
System.out.println ("this is an our inner class");
 }
  }
void display_inner()
 {
innerclass inner= new innerclass();
 inner.print();
 }
  }
public class my-class
{
public static void main (String[] args)
{
outerclass outer = new outerclass();
outer.display_inner();
}}
```

O/P:- this is our innerclass

u.16 Wrapper class in Java: Wrapper classes are used to convert primitive datatype into object.

→ The Java programming language provide a set of wrapper classes for each of the primitive data types. These wrapper classes are part of the Java.lang package

primitive Data types	corresponding wrapper classes
byte	Byte
Short	Short
Int	Integer
long	Long
Float	Float
double	Double
char	character
boolean	Boolean

for eg:-

```
Public class wrapperclass
{
public static Void main (String [] args)
  {
Integer intobj = 42;
Double  doubleobj = 3.14;
Character charobj = 'A';
Boolean  boolobj = true;
Int intValue = intobj;
double doubleValue = doubleobj;
Char  charValue = charobj;
boolean boolvalue = boolobj;

System. out. println ("Integer:" + IntValue);
```

```
System.out.println("double : " + doubleValue);
System.out.println("character : " + charValue);
System.out.println("Boolean : " + boolValue);
}
}
```

4.17 Garbage collection :- Garbage collection in Java is an automatic memory management process where the Java Virtual memory (JVM) identifies and reclams memory that is no longer reachable or in use by program.

→ This process helps prevent memory leaks.

for example

```
public class Garbage_collection
{
public static void main (String [] args)
{
  Object obj = new object (); // creating an object
obj = null; // making the object eligible for garbage collection
System.gc(); // suggesting garbage collection
}
}
```

UNIT 5

INHERITANCE

5.1 Introduction

→ ~~When~~ It is a mechanism in Java by which one class is allowed to inherit the features of the another class.

→ In, Java extends keyword is used to perform inheritance

Syntax

```
Class A
{
=
=
}
Class B extends Class A
{
=
=
}
```

5.2 ~~Types of inheritance~~

5.2 Types of inheritance

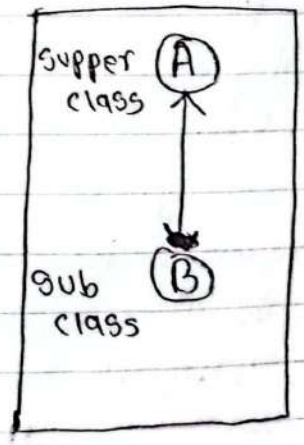
* Simple inheritance:- This inheritance which contain only one supper class and only one sub class is called simple inheritance.

Syntax

```
Class super
{

}
Class sub extends ~~class~~ super
{

}
```

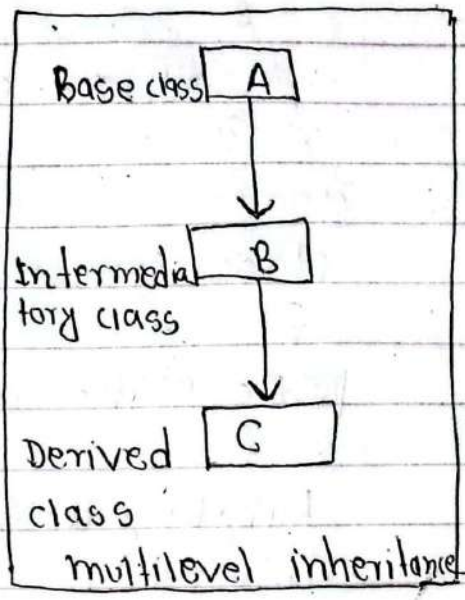


eg:-

```
class student
{
    int roll, marks;
    String name;
    void input()
    {
     System.out.println("Enter roll name & marks: ");
    }
}
  class ganesh extends student
{
     void disp()
     {
        roll= 1; name = "ganesh"; marks= 89;
  System.out.println(roll+" "+name".("+marks)"
}
Public static void main (String[]. args (
{
 ganesh g= new ganesh();
 g.input();
 g.disp();
}
}
```

* multilevel inheritance:

→ In multilevel inheritance we have only one super class and multiple sub classes called multilevel inheritance.



Syntax-

```
class super
{

}
class sub1 extends super
{

}
class sub2 extends sub1
{

}
```

example:-

```
class animal
{
void eat()
{
System.out.println("barking");
}
}
Class Dog extends Animal
{
void bark()
{
system.out.println("barking---");
}
}
```

```
Class Baby Dog extends Dog
 {
void weep()
    {
      System.out.println("weeping ----");
    }
}
 Class ganesh
  {
public static void main(String[] args)
    {
 BabyDog d= new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
```

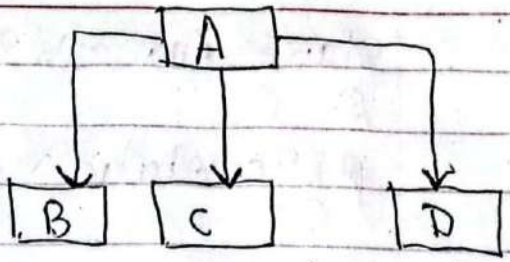
output :-
weeping
barking
eating

* Hierarchical inheritance :-

→ When two or more classes inherits a single class is called hierarchical inheritance.

→ A inheritance which contain only one super class and multiple sub class. All sub class directly extends super class called hierarchical inheritance.

Syntax:-




```
Class A
{

}
Class B extends A
{

}
```

```
class C extends A
{

}
```

```
Class Animal
{
void eat()
{
   System.out.println("eating");
}
}
Class Dog extends Animal
{
Void bark()
{
   System.out.println("barking");
}
}
Class cat extends Animal
{
void meow()
   {
   System.out.println("meowing");
   }
}
```

```
Class Ganesh 2
{
public static void main (string[] args)
  {
Cat c= new cat();
 c. meow();
 c. eat();
 }
}
```

Output :-

meowing
eating.

5.3 Method Overriding :-

→ A polymorphism which exists at the time of execution of program is called runtime polymorphism.

→ whenever we writing method in super and sub classes in such a way that method name and parameter must be same called method overriding.

syntax:

```
Class A
 {
 Void shows()
      {
      }
}
```

```
Class B extends A
  {
     Void Show()
      {
      }
  }
```

→ If subclass (child class) has the same method as declared in the parent class,

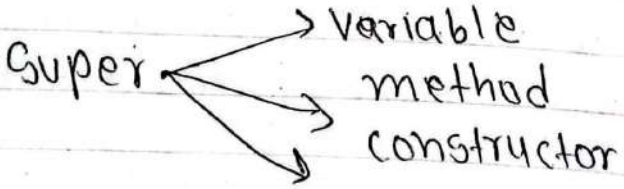
eg:-

```
Class Animal
  {
Void makesound()
  {
   System.out.println("some generic sound");
  }
}
Class Dog extends Animal {
Void makesound()
  {
System.out.println("woof! woof!...");
  }
}
 Class cat extends Animal
{
void makesound()
   {
System.out.println("meow!");
  }
}
```

```
Public class Methodoverriding
{
public static void main (string [] args)
{
Animal g = new Animal ();
my Dog = new Dog ();
my cat = new cat ();
g. make sound ();          // output: some generic sovn
my Dog . make sound ();    // output: woof! woof! --
my cat . make sound ();    // output: meow!
}
}
```

5.4 Using Super key :-

→ Super keyword refers to the objects of super class, It is used when we want to call the super class variable, method & constructor through sub class object.



Syntax

```
Class A          // super
{
  A ()
  {
  .
  }
}
```

```
Class B extends A  // Sub Class
  {
      B()
       {
          // super ()
       }
  }
```

eg:-

```
Class Animal
  {
String color = "white";
  }
Class Dog extends Animal
  {
String color = "black";
 Void print color()
 {
System.out.println (color);
System.out.println (Super.color);
 }
  }
Class Test super1
 {
  public static Void main (String args[])
{
  Dog d = new Dog();
  d.printcolor ();
 }
}
```

5.5 Execution of constructors in multilevel inheritance.

→ Multilevel inheritance is when a class inherits a class which inherits another class.

eg:-

```
class A {
    A()
{
    System.out.println("This is constructors of class A");
}
}
class B extends A
 {
    B()
   {
    System.out.println("This is constructors of class B");
   }
}
class C extends B {
        C() {
 System.out.println("This is constructors of class C");
 }
}
   public class Demo {
public static void main(String[] args)
 {
    C obj = new C();
   }
}
```

// output
This is constructors of class A
" " " " " B
" " " " " C

5.6 Abstract Classes and method

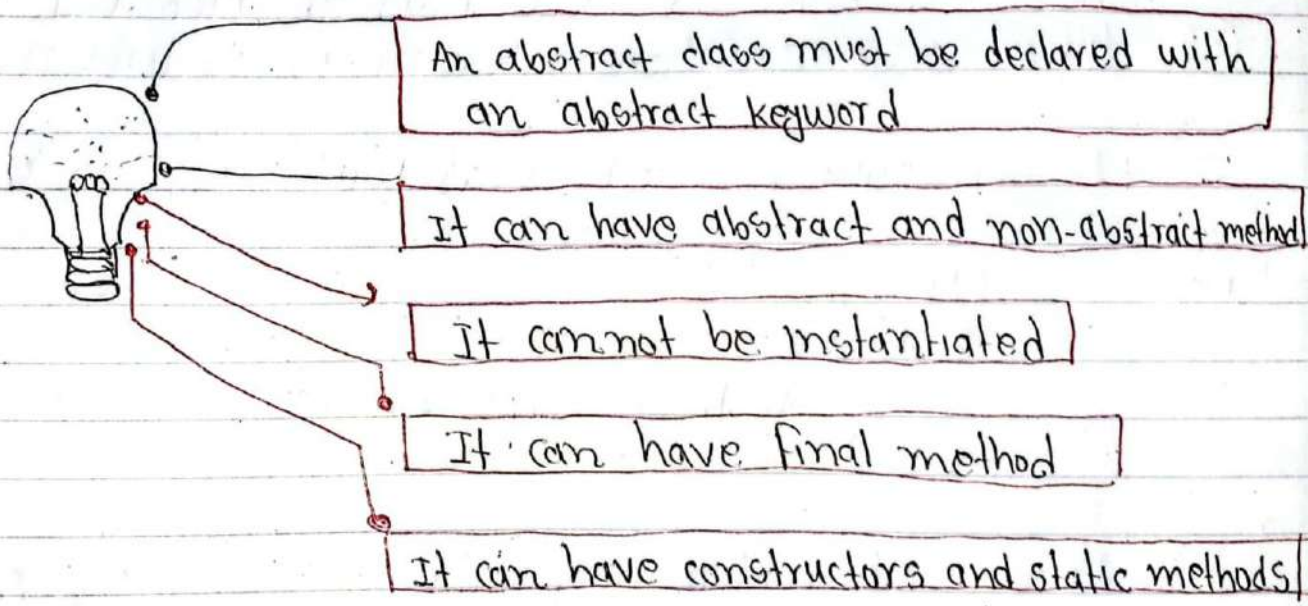
Abstract classes

→ A class which is declared with the abstracts keyword is known as abstract class in Java

→ It can have abstract and non-abstract method. It needs to be extended and its method implemented

→ It can not be instantiated

Rules for Java abstract class



Example

```
abstract Class Bike {
        abstract void run();
}
Class Honda extends Bike
{
    Void run();
    {
        System.out.println("running safely");
    }
```

```
public static void main(String[] args)
    {
        Bike obj = new Honda4();
        obj.run();
    }
}
```

Abstract method :-

→ A method which is declared as abstract and does not have implementation is known as abstract method.

→ It can only be used in abstract class

Syntax

```
class A
  {
      abstract void main();
  }
```

→ It doesn't contain any body "{ }" and always ends with ";".

Example:

```
abstract class programming
  {
      public abstract void Developer();
  }
class HTML extends programming
{
    @ override
    public void Developer ()
```

```
    {
System.out.println ("Tim Berner Lee");
    }
}
class Java extends programming
{
    @Override
   public void Developer()
    {
      System.out.println ("James Gosling");
    }
}
class main
 {
   public static void main (String[] args
    {
      HTML h= new HTML();
        h.Developer();
      Java J=new Java();
        J.Developer();
    }
}

// output
        Tim Berner lee
        James Gosling
```

UNIT 6 INTERFACE AND PACKAGE

6.1 Defining Interface:

→ Interface is just like a class, which contain only abstract method

→ An interface in Java is a blueprint of a class.

→ An interface is declared by using the interface keyword.

6.2 Extending Interface

→ An interface extends other interfaces, just as a class subclass or extends another class

6.3 Implementing interface:

example

```
interface printable
{
void print();
}
class ganesh implements printable
{
public void print()
{
```

```
System.out.println("Hello");
    }
public static void main (String [] args)
  {
    ganesh obj = new ganesh();
        obj.print();
    }
}
```

Output Hello

eg2:-

```
interface client
 {
      void input(); // public + abstract
      void output();
 }
class ganesh implements client
 {
   String name;
   double sal;
public void input()
    {
      Scanner r = new Scanner(System.in);
      System.out.println("Enter user name");
name=r.nextLine();
      System.out.println("Enter Salary");
   sal=r.nextDouble();
    }
public void output()
```

```
    {
        System.out.println(name+" "+sal);
    }
    public static void main(String[] args)
        {
        Client c=new Ganesh();
         c.input();
         c.output();
        }
```

6.4 Accessing Interface Variable:

→ Java interface provide a way to define a contract or blueprint for classes to implement. In addition to method interfaces can also include variable.

example

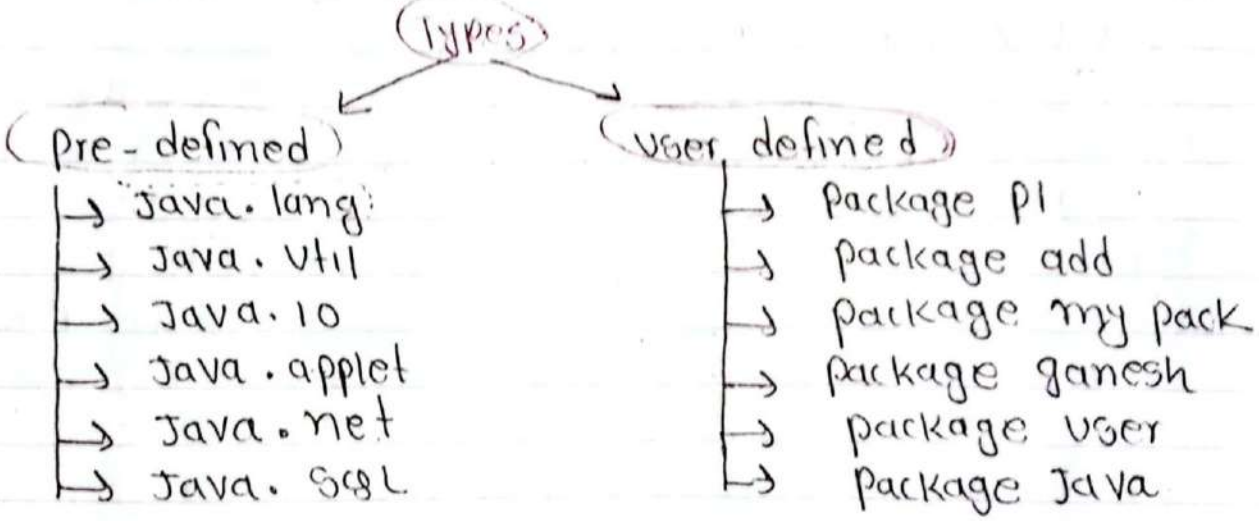
```
interface customer Raj
{
        int amt=5;// public, static, final
        void eat(); // public abstract
}
class seller Sanju implements customer Raj
{
    @override
   public void eat()
    {
      System.out.println("Raj need "+amt" kg rice");
    }
}
```

```
Class Check
{
   public static void main (String[] args )
   {
     Customer Raj C = new seller SanJU();
       C. bal();
   }
}
```

6.5 Introduction to Java package

→ A Java package is used to group of related classes

→ A package arrange number of class interface and sub-package of same type into a particular group.



6.6 Creating a package and naming Convention

→ The package makes the search easier for the classes and interface

→ It provide a fully qualified name that avoids naming conflicts.

also
→ It control access
→ It organizes classes in a folder structure
→ It improve code reusability
→ programmer ~~contain~~ can group classes and interfaces into a related package.

* Naming convention
→ Name of the package must be same as the di
→ under which this file is saved

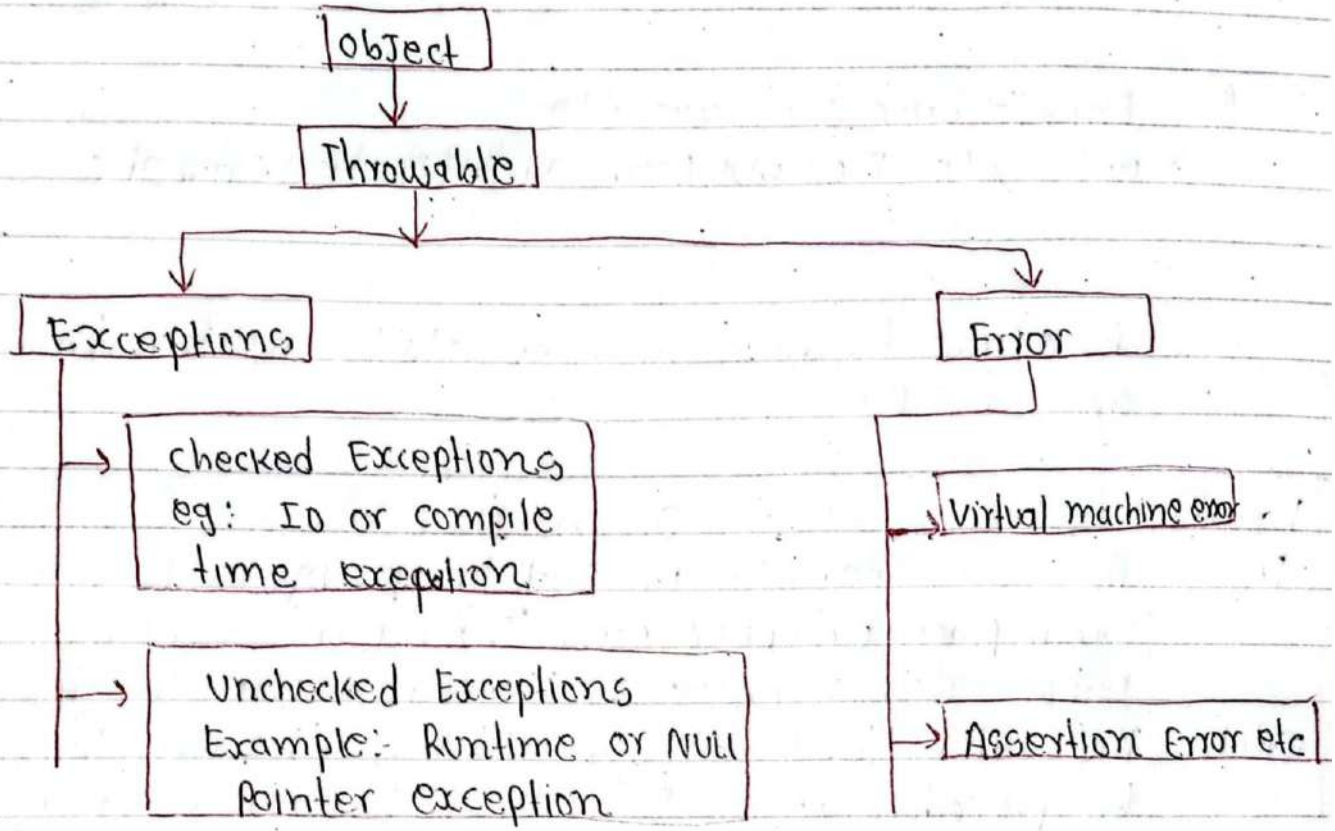
example:
```
package mypackage;
public myclass
{
    public void getNames (String s)
    {
        System.out.println (s);
    }
}
```

UNIT 7

EXCEPTION HANDLING

7.1 Exception and its type :-

→ An exception is an unwanted event that interrupts the normal flow of the program.



→ Types of Exception

* Checked Exception
* Unchecked Exception
* Error

* Checked Exception :- Exceptions other than Run time exceptions are known as checked exceptions as the compiler.

→ eg :- SqlException, IOException, class not found exception etc

* Unchecked Exceptions :- Run time exceptions are also known as unchecked exceptions.

→ Example :- Arithmetic exception, Null pointer exception etc

* Error :- Error is irrecoverable

→ eg : Out of memory Error, Virtual Machine Error, etc

7.2 Exception handling fundamentals (try, catch, throw, throws and finally :

Keyword	Description
Try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by final block later
finally	The "finally" block is used to ~~handle~~ execute the important code of the program. It is executed whether an exception is handled or not.
Throw	The "throw" keyword is used to throw an exception
Throws	The "throws" keyword is used to declare exception. It doesn't throw an exception. It specifies that there may occur an exception in the method. It always used with method signature

7.3 Using try and catch:

→ Java try block is used to enclose the code that might throw an exception.

→ It must be used within the method

→ Syntax of Java try-catch

```
try {
    // code that may throw an exception
}
catch ( Exception_Class_Name ref )
{
}
```

→ Java catch block is used to handle the exception by declaring the type of exception within the parameter.

eg:-

```
public class JavaException
{
    public static void main (String args[])
{
  try
    {
      int data = 100/0;
    }
  catch ( Arithmetic Exception e)
  { system.out.println(e);
  }
  system.out.println (" rest of the code....");
}
}
```

O/P : Exception in thread main java.lang.ArithmeticException: / by zero
rest of the code....

eg 2: Let's simple example of Java multi-catch block

```
public class Ganesh
{
 public static void main (String [] args)
   {
try
   {
       int a[]= new int[5];
       a[5] = 30/0;
   }
catch (Arithmetic Exception e)
     {
       System.out.println("Arithmetic exception occurs");
     }
catch( ArrayIndex Exception e)
     {
      System. out. println(" Array Index Exception occurs");
     }
 catch(Exception e )
     {
       System.out.println ("Exception occurs");
     }
  System. out. println ("rest of the code");
 }
}
```

output:-

Arithmetic Exception occurs
rest of the code

extra:- Java finally:

finally block in Java can be used to put "cleanup" code such as closing a file, closing connection etc.

eg:-

```
class final
{
   public static void main(string args[])
    {
try
    {
       int data=25/5;
        System.out.println(data);
    }
catch (Exception e)
    {
       System.out.println(e);
    }
finally
       { System.out.println("finally block is always executed");
       }
System.out.println("rest of the code----");
 }
}
```

O/P:

```
5
finally block is always executed
rest of the code----
```

7.4 Using throw and throws:

→ The Java throw keyword is used to explicitly (स्पष्ट) throw an exception.
→ We can throw either check or unchecked ~~in Java~~ exception in Java by throw keyword.

Syntax
```
throw exception;
```
Let's see the example of throw IO-Exception.

```
throw new IO-Exception("sorry device error");
```

eg:-
```
public class Throw
{
  static void validate(int age)
    {
  if(age<18)
throw new ArithmeticException("not valid");
else
  System.out.println("welcome to vote");
    }
public static void main(String[] args)
    {
validate(13);
System.out.println("rest of the code---");
}
}
```

Throws

```
public class Throws1
{
static void Validate (int age)
throws ArithmeticException
{
    try
    {
    if (age<18)
throw new ArithmeticException ("not valid");
   else
System.out.println ("welcome to vote");
    }
  catch (ArithmeticException e)
{
    system.out.println(e);
    System.out.println ("rest of the code...");
  }
}
    public static void main (String [] args);
  {
    Validate (12);
  }
}
```

UNIT 8

MULTITHREADING

8.1 Introduction of Thread

→ Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU.

→ A thread in Java at any point of time exists in any one of the following states.

8.2 Creating a Thread

→ Thread can be created by using two mechanisms.

* Extending the thread class
* Implementing the Runnable Interface

* Thread Creation by extending the thread Class.

→ We create a class that extends the Java.lang.Thread Class. This class override the run() method available in the thread Class. A thread begins its life inside run() method. We create an object of our new class and call start() method to start the executation of a thread.

eg:—

```
Class Multithreading Demo extends Thread
    {
public void run()
    {
    System.out.println("Thread" + Thread.currentThread()
    .getID() + " is running");

    }
}

public class Multithread
    {
public static void main(String [] args)
 {
  int n = 8; // Number of threads
  for (int i=0; i<n; i++)
{
  Multithreading Demo object = new MultithreadingDemo();
     object.start();
  }
}
}
```

* Thread Creation by immplementing the Runnable Interface

→ we create a new class which implements Java.lang. Interface and override run() method. Then we instantiate a thread object and call start() method on this object.

```
Class MultithreadingDemo implements Runnable
 {
public void run()
{
 System.out.println("Thread" + Thread.currentThread()
 .getId() + " is running");
}
}

Class Multithreadrunnable
 {
public static void main (string[] args)
 {
  int n=8;
for(int i=0; i<n; i++)
 {
Thread object = new Thread (new multithreadingDemo());
  }

}

}
```

8.3 Thread priorities :- Each thread has a priority. priorities are represented by a number between 1 and 10. In most cases, the thread scheduler schedules the threads according to their priority.

Types of priority:-

→ In Java, a threads priority is an integer in the range 1 to 10. The larger the integer, the higher the priority. The thread scheduler uses this integer from each thread thread to determine which one should be allowed to execute.

→ The Thread class defines three types of priorities

* Minimum priority:- It is a MIN_PRIORITY with values is 1.

* Normal priority:- It is a NORM_PRIORITY with values is 5.

* Maximum priority:- It is a MAX_PRIORITY with values is 10.

eg:-
```
import java.lang.*;
class ThreadDemo extends Thread
{
  public static void main(String[] args)
{
    ThreadDemo t1 = new ThreadDemo();
    ThreadDemo t2 = new ThreadDemo();
    t1.setpriority(NORM_PRIORITY);
    t2.setpriority(MIN_PRIORITY);

  System.out.println("t1 thread priority:"+t1.getpriority());
System.out.println("t2 thread priority:"+t2.getpriority());
```

```
System.out.println("currently Executing Thread: "+Thread.
                    CurrentThread().getName());
Thread.CurrentThread().setpriority(MAX-PRIORITY);
System.out.println("main thread priority: "+Thread.currentthread-
    ().getpriority());

  }
}
```

output:-

t1 thread priority: 5
t2 thread priority: 1
Current Executing Thread: main
Main thread priority: 10

3.4 Life Cycle of Thread (Thread States)

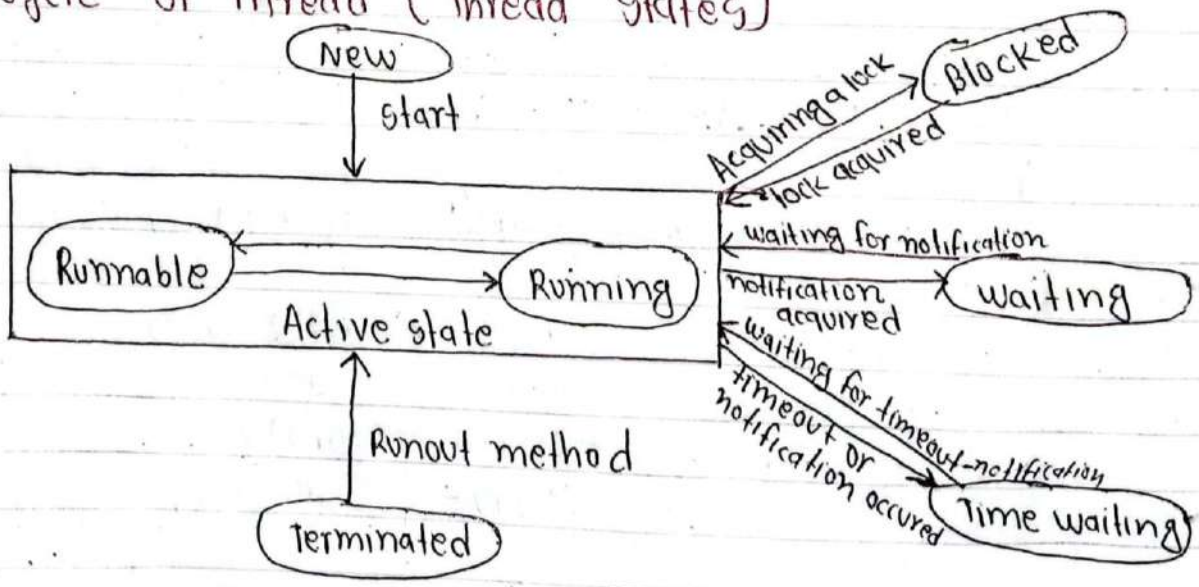


fig:- States of Thread in its lifecycle

→ A thread goes through various stages in its life cycle. For example a thread is born, started, runs and then dies. The above diagram shows the complete life cycle of a thread. Following are the stages of the life cycle.

a New states
b Runnable states
c Blocked states
D Waiting states
E Timed waiting states
F Terminated state.

a) New :- A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.

b) Runnable - After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its tasks.

c) Waiting :- Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transition back to the runnable state only when another thread signals the waiting thread to continue executing

D) Time waiting :- A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that

time interval expires or when the event it is waiting for occurs.

5 Terminated (Dead):- A runnable thread enters the terminated state when it completes its task or otherwise terminates.

UNIT 9:- I/O PACKAGE

9.1 Java I/o package:- Java input and output is used to process the input and produce the output based on the input

→ Java uses the concept of Stream to make I/O operation fast. The Java.io package contains all the classes required for input and output operation

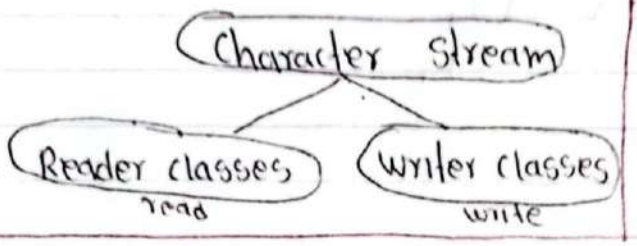
→ We can perform file handling in Java by Java IO API

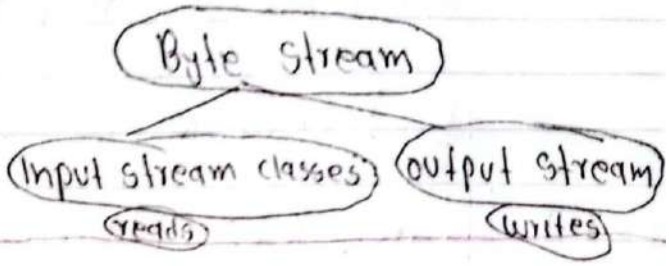
9.2 Byte stream and character stream classes

→ Byte stream in Java are used to perform input and output operations of 8 bit byte

→ Character Stream is used to perform input and output operation for 16-bit unicode.

Character Stream	Byte Stream
A character stream will access a file character by character	A byte stream access the file byte by byte.
These handle data in 16 bit unicode	These handle data in bytes(8 bit)
Using these you can read and write text data only.	Using these you can store character, videos, audios, image etc





9.3 Using file input strem and file output Stream classes.

Java file input Stream classes :-

→ Java file input Stream class obtains input bytes from a file.

→ It is used to readings streams of raw bytes such as image data.

→ eg :- read image, audio, video, etc

→ ~~Program ##~~

eg:-

```
import java.io.*;
class SimpleRead
{
public static void main(String args[])
  {
try
  {
FileInputStream fin = new FileInputStream("abc.txt");
  int i=0;
while (i= fin.read()!=-1)
  {
System.out.println((char)i);
}
  fin.closec();
}
catch (Exception e)
  {
System.out.println(e);
 }
}
}
```

Java file output Stream class.

→ Java fileOutputStream is an output stream for writing data to a file.

→ If you have to write primitive values then use FileOutput Stream.

eg:-

```
import Java.io.*;
Class Test
  {
public static void main (String [] args)
  {
try
   {
   file output Stream fout = new file output Stream ("abc.txt");
  String s = " programme is my life";
   byte b[] = s.get Bytes (); // converting string into byte array
   fout.write(b);
   fout . close ();
  System.out.println ("success");
}
Catch (Exception e)
   {
  System.out.println(e);
  }
}
}
```

9.4 Using FileReader and file writer classes

Using file Reader classes

→ It is used to read data from the file. It returns data in byte format like fileinput Stream class.

method

public int read() :- returns a character in Ascii form. It returns -1 at the end of file

Public void close() :- closes file Reader.

eg:- In this example we are reading the data from file abc.txt file.

```
import Java.io.*;
class Simple
{
public static void main(String [] args )
throws Exception
{
FileReader fr = new FileReader("abc.txt");
 int i;
while (i= fr.read())!=-1)
System.out.println((char)i);
fr.close();
}
}
```

O/P:- my name is ganesh

Java file writer class

→ Java file Writer class is used to write characteroriented data to the file

method// public void write(String txt):- writes the String into file Writer

Public void write(char c):- writes the char into filewriter

Public void write(char [] c):- write char array into filewriter.

eg

```
import Java.io.*;
class ganesh kapadi
  {
public static void main (String [] args)
    {
try
  {
fileWriter fw = new fileWriter ("abc.txt");
 fw.write ("my name is ganes");
 fw.close();
 }
Catch (Exception e)
    {
System.out.println(e);
 }
System.out.println ("success");
 }
}
```

output :-
success.

THE END