-: CORE JAVA :-

****

==========
What is Fullstack
==========

Fullstack Development = Frontend Development + Backend Development

Frontend : User interface

-> Clients / Users will interact with s/w application using frontend

Backend : The hidden part of our application which contains business logic

-> When use perform some operation on frontend then backend logic will execute to handle that operation

-> The programmer who can develop frontend and backend of application is called as fullstack developer

========
Tools
=========
Git Hub : For Code Integration

JIRA : Project Management / Bug Tracking / Work Assignment

SonarQube : For Code Quality Checking

JUnit : For Unit Testing

JMETER : For Performance Testing

JENKINS : For Deployment (Automated Deployment)

==========
Cloud
==========
Amazon ---> AWS
Microsoft ---> Azure
Google -----> GCP
Oracle Cloud
IBM Cloud
VM Ware Cloud
Alibaba CLoud etc.....

=====================================
Roles & Responsibilities of Fullstack Developer
=====================================

1) Understand Requirements of Project
2) Analyze requirements
3) Design / Planning
4) Database Design
5) Development (Backend development)
6) Unit Testing
7) Code Review

8) Code Integration (Git Hub)
9) Frontend Development
10) Frontend + Backend Integration
11) Deployment
12) Support / Maintenence

==========
What is Java
==========

-> Java is a programming language

-> Java language developed by Sun Microsystem in 1991 (OAK)

-> James Gosling is the lead for the team who developed Java Language

-> The first version of java came into market in 1995

Note: Oracle Corporation acquired Sun Microsystem

-> Now java is under license of Oracle corporation

-> Java is a free software & open source

==========
Java is divided into 3 parts
==========

1) J2SE
2) J2EE
3) J2ME

J2SE / JSE ---> JAVA STANDARD EDITION

-> STAND-ALONE APPS
-> RUNS ONLY IN ONE MACHINE

EX: CALC, GAMES, NOTEPAD ETC......

J2EE / JEE ---> JAVA ENTERPRISE EDITION

-> web applications

-> Everybody can access web applications using internet

ex: gmail, youtube, facebook, naukri, irctc etc.........

J2ME / JME ---> JAVA MICRO / MOBILE EDITION

-> Mobile apps

Ex: whatsapp, messgender, phonepay, gpay etc.....

==========
What we can do using Java
==========

1) Stand-alone applications

2) Web applications

3) Mobile Applications

==============
Java Features
==============

1) Simple : The complex topics of C & C++ are eliminated in Java

        Ex: Operators overloading, pointers, memory mgmt etc...

2) Platform Independent

        -> Java programs can be executed on any machine
        -> JVM made java as platform independent
        -> JVM stands for Java Virtual Machine
        -> JVM is responsible to run/execute java programs

3) Robust (Strong)

        -> Automatic Memory Management
        -> Exception Handling

4) OOPS (Object Oriented Programming System)

        -> Everything will be represented in objects format
        -> Code Re-Usability

5) Secure

6) Distributed

7) Portable

8) Dynamic

Java Slogan : WORA (Write Once Run Anywhere)

==================
Environment Setup
==================

1) Download and Install Java Software

                - JDK (Java Development Kit)
                - JRE (Java Runtime Environment)

Q) What is the difference between JDK, JRE & JVM ?

                - JDK contains set of tools to develop java programs
                - JRE providing a platform to run our java programs
                - JVM will take care of program execution

2) Set Path for Java

                Path = C:\Program Files\Java\jdk1.8.0_202\bin

                -> Go To Environment Variables
                -> Go To System Environment Variables
                -> Edit Path
                -> Add JDK BIN path

========================
Java Programs Development
========================

-> We can write java programs in any text editor

                        - Note Pad

- Note Pad++
- Edit Plus

-> In companies we will use IDE to develop java programs/projects

- Integrated Development Environment

  - Eclipse
  - MyEclipse
  - Netbeans
  - STS (Spring Tool Suite)
  - IntelliJ

==========
Java Program Structure
==========

package statements

import statements

class declaration

variables

methods

----------hello.java---------
```
class hello {

 public static void main(String... args) {
     System.out.println("Welcome To Ashok IT...!!");

    System.out.println("Welcome to Java");
 }

}
```
----

javac hello.java

java hello

```
class demo {

   public static void main (String... args){
               System.out.println("Hello World");
             System.out.println("Welcome to Java");
   }
}
```

-> javac measns java compiler which is used to compile java programs

-> java compiler is called as translator

==========
Translators
==========

-> It is used to convert from one format to antoher format

-> 3 types of translators available

1) Interpreter

2) Compiler

3) Assembler

-> Interpreter will convert the program line by line (performance is slow)

-> Compiler will convert all the lines of program at a time (performance is fast)

-> Assembler is used to convert assembler programming languages into machine language

========
JVM
========

-> JVM stands for Java Virtual Machine (We can't see with our eyes)

-> JVM will be part of JRE

-> JVM is responsible for executing java programs

-> JVM will allocate memory required for program execution & de-allocate memory when

it is not used

-> JVM will convert byte code into machine understandable format

================
JVM Architecture
===============

1) Classloader subsystem : It will load .class file into JVM

2) Method Area : Class code will be stored here

3) Heap area : Objects will be stored into heap area

4) Java Stack : Method execution information will be stored here

5) PC Register : It will maintain next line information to execute

6) Native Stack : It will maintain non-java code execution information

7) Execution Engine (Interpreter + JIT) : It is responsible to execute the program

and provide output/result

8) Native Interface : It will load native libraries into jvm

9) Native Libraries : Non-java libraries which are required for native code execution

=======
variables
========

-> variables are used to store the data

name - ashok

age - 30

gender - m

isStudent - false

mysalary - 400.56

-> We need to specify type of the variable to store the data

-> To specify type of data we will use 'data types'

=========
data types
=========

-> Data types are used to specify type of the data

-> Data types are divided into 2 categories

1) Primitive / Pre-Defined Data Types

1) Integral
- byte
- short
- int
- long

2) Decimal
- float
- double

3) Character
- char

4) Boolean
- boolean

2) Non-Primitive / Referenced Data Types

- Arrays
- Strings
- Classes

================
Integral data types
================

--> Integral data types are used to store numbers without decimal points

--> We can store both positive and negative numbers using integral data types

Ex:

age = 30

phno = 66868686868

studentscnt = 40

balance = - 3000

-> We have 4 data types in this category

-> For These 4 data types memory & range is different

1) byte ----> default value is 0 ----> 1 byte

2) short ----> default value is 0 ----> 2 bytes

3) int ----> default value is 0 ----> 4 bytes

4) long ----> default value is 0l ----> 8 bytes

==========
Decimal data types
==========

-> Decimal data types are used to store numbers with decimal values

-> We can store both postive and negative values

Ex:

petrol price = 110.567979
stockPrice = 334.3279797979797979
percentage = 9.8
weight = 55.6
height = 5.6
length = 10.2

-> In this category we have 2 data types

1) float ----> 4 bytes ---> upto 6 decimal points

2) double -----> 8 bytes --> upto 15 decimal points

==========
character data type
==========

-> Character data type is used to store single character

-> Any single character (alphbet / digit / special character) we can store using 'char' data type

-> char datatype will occupy 2 bytes of memory

-> When we are storing data into 'char' data type single quote is mandatory

-> default value is 'u0002'

gender = 'm'

rank = '1'

Note: In C language 'char' will take only 1 byte where as in java 'char' will take 2 bytes

==========
boolean data type
==========

-> It is used to store true or false values only

-> It will occupy 1 bit memory

Note: 8 bits = 1 byte

-> default value for boolean is false

Ex:

isPass;
isFail
isMarried
isOdd
isEven

==========
Variables
==========
-> Variables are used to store the data / value

-> To store the data into variable we need to specify data type

-> To store data into variables we need to perform 2 steps

1) Variable Declaration (defining variable with data type)

Ex: byte age ;

2) Variable Intialization (storing value into variable)

Ex: age = abc;

-> We can complete declaration and intialization in single line

byte age = 20;

================================ Variables Program
==================================

```
class var {

	public static void main (String... args) {

			int age = 20;
			System.out.println(age);

			float a = 25.01f;
			System.out.println(a);

			double price = 120.87;
			System.out.println(price);

			char gender = 'm';
			System.out.println(gender);

			boolean pass = true;
			System.out.println(pass);

	}
}
```

1) Identifiers

2) Keywords

3) Java Naming Conventions

==========
Identifiers
==========

-> All java components requires a name

-> For variables, for classes and for methods we need a name

```
int age ;

class Hello {
        // code
}

main ( ) {
        //logic
}
```

-> The name which we are using for packages, variables, classes & methods is called as identifier

-> We can use any name for identifiers but we need to follow below rules to work with identifiers

Rule-1 : Java will allow only below charaters for identifiers

1) a - z
2) A - Z
3) 0 to 9
4) $ (dollar)
5) _ (underscore)

Ex:

```
name   -----> valid
name@ -----> invalid
age#   ------> invalid
```

Rule-2 : Identifier should not start with digit (first character shouldn't be digit)

```
1age --------> invalid
age2  ------> valid
name3  -----> valid
_name -----> valid
$name ------> valid
@name ------> invalid
$_amt   --------> valid
_1bill -----------> valid
```

Rule-3 : Java reserved words shouldn't be used as identifier (53 reserved words)

```
int byte = 20;    -------> invalid bcz byte is a reserved word

byte for = 25;   -------> invalid bcz for is a reserved word

int try = 30;   ------> invalid bcz try is a reserved word

long phno = 797979799 -----> valid
```

Rule-4 : Spaces are not allowed in identifiers

```
int mobile bill = 400;   // invalid
int mobile_bill = 400 ;   // valid
```

Rule-5 : Java is case sensitive language 'name' & 'NAME' both are not same

==========
Java Naming Conventions (Java Coding Standards)
==========

-> Java language followed some standards/conventions for pre-defined packages, classes

and methods....

-> Java language suggested java programmers also to follow same standards / conventions

-> Following these standards/conventions is not mandatory but highly recommended.

==========
Naming Convention For Class Name
==========

-> A class name can contain any no.of words without spaces
-> Recommended to write every word first character as uppercase in class name

Examples:

```
class Hello  {
}

class HelloWorld {
}

class UserManagementService{
}

class WelcomeRestController {
}
```

Note: Class Names & Interface Names conventions are same.

==========
Variables Naming Convention
==========

-> Variable name can have any no.of words without spaces

-> Recommended to start variable name with lowercase letter

-> If variable name contains multiple words then recommended to write firstword all characters in

lowercase and from second word onwards every word first character in Uppercase

Examples:

```
int age ;

int  userAge;
```

```
long creditCardNumber ;
```

==========
Method Naming Convention
==========

> Method name can have any no.of words without spaces

-> Recommended to start method name with lowercase letter

-> If method name contains multiple words then recommended to write firstword all characters in

lowercase and from second word onwards every word first character in Uppercase

```
main ( )  {
}

save ( ) {

}

saveUser( ) {

}

getWelcomeMsg ( ) {

}
```

Note: Variables & Methods naming conventions are same. But methods will have parenthesis (())

variables will not have parenthesis.

==========
Naming Conventions for Constants
==========

-> Constant means fixed value (value will not change, it is fixed)

-> Recommended to write constant variable all characters in uppercase

-> If constant variable contains multiple words recommended to use _ (underscore) with all

uppercase characters

```
final int MIN_AGE = 21;

final int MAX_AGE = 60 ;

int PI = 3.14;
```

==========
Naming Conventions for Packages
==========

-> Package name can have any no.of characters & any of words

-> Recommended to use only lowercase letters in package names

-> If package name conatins multiple words then we will use . (dot) to seperate words

Eamples:

java.lang
java.io
java.util
in.ashokit
com.oracle
com.ibm

==========
Chaper-1
==========

1) What is Java
2) Java Features
3) Java Environment Setup
4) JDK vs JRE vs JVM
5) Java Programs Execution Flow
6) Java Programs Development (Compilation & Execution)
7) Variables
8) Data Types
9) Identifiers
10) Reserved Words (53)
11) Java Coding Standards (Naming Conventions)
12) Java Comments

==========
Chapter-2 : Operators & Control Statements
==========

==========
Operators
==========

-> Operator is a symbol which performs some operation on operands

```
int a  = 10 ;

int b = 20 ;

int c = a + b;
```

-> We have below operators in java

1) Arithematic Operators
2) Logical Operators
3) Relational Operators
4) Assignment Operators
5) new operator
6) dot (.) operator
7) ternary operator (Conditional Operator)

-> Arithematic Operators are used to perform Arithematic Operations (Calculations)

1) Addition -----> +
2) Substraction ----> -
3) Division ----------> / (quotient)
4) Multiplication ----------> *
5) Modulas ---------> % (reminder)

6) Increment -------> ++
7) Decrement ------> --

--> Increment (++) is used to increase the value of variable by 1

-> Increment is divided into 2 types

1) Post Increment (a ++)
2) Pre Increment (++ a)

--> Decrement (--) is used to decrease the value of variable by 1

-> Decrement is divided into 2 types

1) Post Decrement (a --)

2) Pre-Decrement (--a)

```
class PostIncrement {
   public static void main(String... args){
                        int a = 5;
                        System.out.println(a++); // it will print 5 then it will
become 6
                        a++;  // it will become 7
                        System.out.println(a++); // it will print 7 then it will
become 8
                        System.out.println(a); // it will print 8
   }
}

class PreIncrement {
   public static void main(String... args){
                        int a = 5;
                        System.out.println ( ++ a );  // it will become 6 then it
will print
                        ++ a ;  // it will become 7
                        System.out.println(++a);  // it will become 8 then it will
print
                        System.out.println(a); // it will print 8
   }
}

class PostPreIncrement {
   public static void main(String... args){
                        int a = 5;
                        int b = ++a  + a++ + a++  + ++a;
                        // int b = 6 + 6 + 7 + 9 ==> 28
                        System.out.println(b);
   }
}
```

```
class Decrement {

   public static void main(String... args){
                        int a = 5;
                        System.out.println( a -- );  // it will print 5 then it
will become 4
                        System.out.println( -- a);  // it will become 3 then it
will print 3
   }
}

class PostPreDecrement {

   public static void main(String... args){
                        int a = 5;
                        int b =  a-- + --a + a--;

                        // int b = 5 + 3 + 3
                        System.out.println ( b );
   }
}
```

==========
Relational Operators
==========

-> Relations Operators are used to check relation between two Operands

> , < , >= , <=, !=, ==

==========
Logical Operators
==========

-> To check more than one condition then we will use Logical operators

AND ----> &&

OR -----> ||

NOT -----> !

==========
Assignment Operator
==========

-> Equals (=) is called as assignment operator

-> It is used to assign the value for a variable

```
                int a  = 10 ;
```

==========
new operator
==========

-> It is used to create the object for a class

```
                ClassName refVar = new ClassName  (  );
```

Note: Creating object means allocating memory in heap area

==========
Dot (.) Operator
==========

-> Dot operator is used to access class variables & methods

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

```
java.lang.String

java.util.ArrayList
```

==========
Ternary Operator / Conditional Operator
==========

-> Ternary operator is used for decision making

Syntax:

```
( condition ) ? expression-1 : expression-2
```

-> If condition satisifed then expression-1 will execute otherwise expression-2 will execute

==========
instanceof operator
==========

- > It is used to check object reference belong to a class or not

```
String str = "ashokit";

if (str instanceof String ) {
        //logic
}
```

==========
Control Statements
==========

-> Java program code will execute line by line sequentially (this is default behaviour)
-> In project code should execute based on user operation
-> To satisfy user requirement our code should execute based on some conditions
-> Using Control Statements we can control program execution flow
-> Control Statements are divided into 3 types

1) Decision Making Statements / Conditional Statements
2) Looping Statemetns
3) Transfer / Branching Statements

==========
Conditional Statements
==========

=> Execute the code only once based on condition

1) simple if
2) if - else
3) if - else - if - else -if - else (if else ladder)
4) switch

==========

Looping Statements
==================

=> To execute the code repeatedly

1) while loop
2) do-while loop
3) for loop
4) for-each loop

============================
Branching / Transfer Statements
============================

1) break;
2) continue;
3) return

===========
Simple if
===========

-> To execute the statemetns based on condition

syntax

```
if ( condition )
{
    // stmt - 1
    // stmt - 2
    // stmt - 3
}
```

or

```
if (condition )
    //stmt
```

```
class SimpleIf{

   public static void main(String... args){
                        int a = 100;

                        int b = 20;

                        if( a > b ) {
                                System.out.println("a is greater than b");
                                System.out.println("Completed");
                        }
                        System.out.println("Bye");
   }
}
```

```
class IfElseDemo {

      public static void main (String... args){
                        int age = 16 ;

                        if ( age >= 18 ) {
                                System.out.println("Eligible For Vote") ;
                        } else {
                                System.out.println("Not eligible for Vote");
```

```
                        }
      }
}
```

Requirement :

---

int a = 20;

if a > 0 -----> display msg as 'a is positive number'

if a < 0 ----> display msg as 'a is negative number'

When above both conditions are failed then display msg as 'a is zero'

---

syntax

```
if ( condition_1 ) {
                // stmt - 1
} else if ( condition_2 ) {
                // stmt - 2
} else  if ( condition_3 ) {
                //stmt - 3
} else {
                //stmt-4
}
```

-> if condition_1 is pass then it will execute only stmt-1

-> if condition_1 is fail then it will check condition_2

-> If condition_2 is pass then it will execute only stmt-2

-> If condition_2 is fail then it will check condition_3

-> If condition_3 is pass then it will execute only stmt-3

-> If condition_3 is fail then directley stmt-4 will be executed

---

```
class IfElseLadderDemo {

        public static void main(String... args){

                        int a = 0;

                        if( a > 0) {
                                System.out.println(" a is positive number ");
                        } else if ( a < 0 ) {
                                System.out.println("a is negative number");
                        } else {
                                System.out.println("a is zero");
                        }
                }
}
```

Assignment : Develop a java program to decide role of software engineer based on his/her experience

---

0 - 2 year exp -----> Associate Engineer

3 - 5 years exp -----> Software Engineer

6 - 9 years exp -----> Sr.Software Engineer

10 - 13 years exp ----> Manager

```
class RoleFinder {

        public static void main(String... args) {

                        int exp = 13;

                        if( exp >= 0 && exp <= 2 ){
                                        System.out.println("Associate Engineer");
                        }else if ( exp >= 3 && exp <=5 ){
                                        System.out.println("Software Engineer");
                        }else if(  exp >= 6 && exp <=9 ){
                                        System.out.println("Sr. Software
Engineer");
                        }else if( exp >= 10 && exp <=13 ){
                                        System.out.println("Manager");
                        }else {
                                        System.out.println("Role Not Found");
                        }

      }
}
```

=> In above program we have hardcoded value for the variable
=> If we want to test our program with different values we need compile and execute everytime
=> To overcome this problem we can read the data from keyboard

==========
How to read data from keyboard In Java
==========

1) BufferedReader   (java.io)

2) Scanner (java.util)

3) Command Line Arguments (input for main method)

----------BufferedReader Program----------

```
import java.io.*;

class RoleFinder {

        public static void main(String... args)  throws Exception {

                        InputStreamReader isr = new InputStreamReader(System.in);
                        BufferedReader br = new BufferedReader(isr);
                        String str = br.readLine ( );
                        int exp = Integer.parseInt(str);

                        if( exp >= 0 && exp <= 2 ){
                                        System.out.println("Associate Engineer");
                        }else if ( exp >= 3 && exp <=5 ){
                                        System.out.println("Software Engineer");
                        }else if(  exp >= 6 && exp <=9 ){
                                        System.out.println("Sr. Software
Engineer");
                        }else if( exp >= 10 && exp <=13 ){
                                        System.out.println("Manager");
                        }else {
```

```
                                        System.out.println("Role Not Found");
                        }

      }
}
-------------------------------------------------------------------------------
-
Requirement : Write a java program to find given number is odd or even
Note: Read number from keyboard
-------------------------------------------------------------------------------
-
import java.io.*;

class OddOrEven {

    public static void main(String... args) throws Exception {

                        InputStreamReader isr = new InputStreamReader(System.in);

                        BufferedReader br = new BufferedReader ( isr );
                        System.out.println("Enter Number");

                        String str = br.readLine ( );

                        int num = Integer.parseInt (str);

                        if( num % 2 == 0){
                                System.out.println("It is even");
                        }else{
                                System.out.println("It is odd");
                        }
    }

}

-------------------------------------------------------------------------------
-
Assignment -1 : Write a java program to check given number is a prime number or not

Assignment -2 : Write a java program to check given year is a leap year or not
-------------------------------------------------------------------------------
-
=============
swtich case
==============

-> Using switch case we can make decision

-> When we have upto 5 conditions test then if-else is recommended

-> When we have 10 or 20 conditions to test then switch is recommended

syntax
----------

switch ( case ) {

case 1 :  // stmt - 1
        break;

case 2 : // stmt - 2
        break;
case 3 : // stmt - 3
        break;
...
default : // stmt - default

}
```

Requirement : Write a java program to read a number from keyboard.

Based on the given number print week of the day using 'switch' case

1 - Monday
2 - Tuesday
3 - Wednesday
4 - Thursday
5- Friday
6 - Saturday
7 - Sunday
>7 - No day found

```
import java.io.*;

class WeekDay {

        public static void main(String... args) throws Exception {

                InputStreamReader isr = new InputStreamReader(System.in);
                BufferedReader br  = new BufferedReader(isr);

                System.out.println("Enter number");
                String str = br.readLine ( );

                int num = Integer.parseInt(str);

                switch ( num ) {
                        case 1 : System.out.println("Monday");
                                        break;

                        case 2 : System.out.println("Tuesday");
                                         break;

                        case 3 : System.out.println("Wednesday");
                                        break;

                        case 4 : System.out.println("Thursday");
                                         break;

                        case 5 : System.out.println("Friday");
                                        break;

                        case 6 : System.out.println("Saturday");
                                         break;

                        case 7 : System.out.println("Sunday");
                                        break;

                        default : System.out.println("Day not found");
                }

        }
}
```

---

1) simple if

2) if - else

3) if - else if - else

4) switch

==========
Conclusion
==========

1) 'if' accepts only boolean value (or) boolean expression

2) 'switch' accepts numbers, char & strings (added in java 1.7v)

3) switch will not accept boolean and decimal values

4) switch cases should belongs to same type

5) switch case datatype and switch case input value should belongs to same datatype

6) 'default' case is optional in 'switch case'

7) 'break' keyword is also optional in 'switch case'

==========
Loops in Java
==========
-> Loops are used to execute statements repeteadly

-> In java we have below loops

1) while loop
2) do-while loop
3) for loop
4) for-each loop (arrays & collections)

==========
While loop
==========
-> While loop is used to execute statements until condition is true

-> while loop is called as conditional based loop

-> If condition is true then loop statments will execute otherwise loop will be terminated

syntax

```
while ( condition ){
        //stmts
}
```

Q) Write a java program to print numbers from 1 to 10 using while loop

```
class WhileDemo {
    public static void main (String... args){
                        int i = 1;
                        while ( i <= 10 ){
                                        System.out.println(i);
                                        i++;
                        }
        }
}
```

==========
do-while loop
==========

-> It is used to execute statements first then it will check the condition

-> do-while is also called as conditional based loop only

->

Syntax:

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

Q) Write a java program to print numbers from 1 to 10 using do-while loop

```
class DoWhile{
        public static void main(String... args){

                        int i = 1;
                        do {
                                System.out.println(i);
                                i++;
                        }while (i <= 10);
        }
}
```

Q) What is the difference between while and do-while ?

while ==> It will check the condition first then it will execute the statements

do-while ==> It will execute statement first then it will check condition.

Note: Even if condition is not satisifed our statement will execute once.

==========
for loop
==========
-> It is used to execute statements multiple times

-> For loop is called as Range based loop

syntax

```
for ( initialization ; condition ; increment / decrement ) {

                        //stmts
}
```

Q) Write java program to print numbers from 1 to 10 using for loop

```
class ForLoop {

        public static void main(String... args){
                        for ( int i = 1 ; i <= 10 ; i++ ) {
                                        System.out.println(i);
                        }
        }
}
```

==========
Nested Loops
==========

-> Writing one loop inside another loop is called as Nested loop

```
for ( int i = 1; i <= 5 ; i++ ){
        for ( int j = 1; j< = 5; j++){
        }
}
```

-> As per above program, for every execution of outer loop 5 times inner loop will execute

Q) Write a java program to print below pattern using loops

```
*
* *
* * *
* * * *
* * * * *
```

```
class NestedLoop {
        public static void main(String... args){
                for ( int i = 1; i <=5 ; i++ ){
                        for ( int j = 1; j <= i ; j++ ){
                                System.out.print("*");
                        }
                        System.out.println();
                }
        }
}
```

Q) Write a java program to print below pattern

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
class NestedLoop {
        public static void main(String... args){
                for ( int i = 1; i <=5 ; i++ ){
                        for ( int j = 1; j <= i ; j++ ){
                                System.out.print(j);
                        }
                        System.out.println();
                }
        }
}
```

==========
Branching Statements
==========

break ====> It is used to come out from switch case and from loops

continue ====> It is used to skip one iteration in the loop execution then continue

return ====> To come out from the method

```
class Break {

	public static void main(String... args){

			for (int i = 1; i<= 10; i++ ){
				if (i >= 5 ){
					break;
				}
				System.out.println(i);
			}
	}
}

class Continue {

	public static void main(String... args){

			for (int i = 1; i<= 10; i++ ){
				if(i == 6 ) {
					continue;
				}
				System.out.println(i);
			}
	}
}
```

==========
Chapter-2 : Operators & Control Statements
==========

Operators : To perform some operations
- Arithematic Operators
- Relational Operators
- Logical Operators
- Ternary Operator
- Assignment Operator
- New Operator
- Dot operator

Conditional Statements : Execute the code only once based on condition
- simple if
- if - else
- if - else if - else ladder
- switch case

Loops Concept : Execute the code repeatedly based on condition / range
- while loop
- do-while loop
- for loop
- for each (Arrays & Collections)

Transfer / Branching Statements : To come out from loop, to skip loop iteration, to

come out from method
- break
- continue
- return (used to return some value from the method)

==========
Chapter-2 :: Logical Programs
==========

Q-1) Write a java program to read shoes brand name from keyboard, based on brand name

print brand slogan like below

Nike -> Just do it

Adidas -> Impossible is nothing

Puma -> Forever Faster

Reebok -> I Am What I Am

==========

```
import java.io.*;

class Shoes {

    public static void main(String... args) throws Exception {

                        InputStreamReader isr = new InputStreamReader (System.in);
                        BufferedReader br = new BufferedReader ( isr );
                        System.out.println ("Enter Brand Name");
                        String brand = br.readLine ( );

                        switch ( brand ) {

                                case "Nike" : System.out.println("Just do it");
 break;

                                case "Adidas" : System.out.println("Impossible is
nothing");  break;

                                case "Puma" : System.out.println("Forever Faster");
 break;

                                case "Reebok" :  System.out.println("I Am What I
Am");   break;

                                default : System.out.println("Brand Not Found");
                        }
    }
}
```

==========
==========
Q-2) Write a java program to read person basic salary and calculate Provident Fund amount

from the basic salary

Formula : Provident Fund is 12 % of Basic Salary

==========
==========

```
import java.io.*;
```

```
class EmpPf {

        public static void main(String... args) throws Exception {

                        InputStreamReader isr = new InputStreamReader( System.in );
                        BufferedReader br = new BufferedReader (isr);

                        System.out.println("Enter Basic Salary");
                        String str = br.readLine ( );

                        double basicSalary = Double.parseDouble ( str );

                        double pf = basicSalary * 12 / 100;

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

Q-3) Write a java program to read person age and person salary and print his eligibility

for marriage

Condtion : If person age lessthan 30 and salary greater than 1 lakh then eligible for marriage

---

```
import java.io.*;

class Marriage {

    public static void main (String... args) throws Exception {

                InputStreamReader isr = new InputStreamReader( System.in );
                BufferedReader br = new BufferedReader(isr);

                System.out.println("Enter Your Age");
                String str1 = br.readLine ( );
                int age = Integer.parseInt ( str1 );

                System.out.println("Enter Your Salary");
                String str2 = br.readLine ( );
                double salary = Double.parseDouble(str2);

                if ( age < 30 && salary > 100000 ) {
                                System.out.println("You are eligible for
marriage");
                } else {
                                System.out.println("You are not eligible for
marriage");
                }
   }
}
```

Q-5) Write a java program to print Right Triangle Star Pattern*

```
*
* *
* * *
* * * *
* * * * *
```

```
class RightTriangle {
```

```
                public static void main(String... args) {
                                for( int i = 1; i<=5 ; i ++ ){
                                        for( int j = 1; j<=i ; j++ ){
                                                        System.out.print ("* ");
                                        }
                                        System.out.println();
                                }
                }
}
```

==========

Q-6) Write a java program to print left traingle start pattern*

```
          *
        * *
      * * *
    * * * *
  * * * * *
```

==========

```
class LeftTriangle {

                public static void main(String... args) {
                                for( int i = 1; i<=5 ; i ++ ){
                                        for ( int k = 5-i ; k >= 1 ; k-- ){
                                                System.out.print(" ");
                                        }
                                        for( int j = 1; j<=i ; j++ ){
                                                        System.out.print ("*");
                                        }
                                        System.out.println();
                                }
                }
}
```

==========

Q-7) Write a java program to print Pyramid pattern

```
    *
   * *
  * * *
 * * * *
* * * * *
```

==========

```
class Pyramid {
                public static void main(String... args) {
                                for( int i = 1; i<=5 ; i ++ ){
                                        for ( int k = 5-i ; k >= 1 ; k-- ){
                                                System.out.print(" ");
                                        }
```

```
                                        for( int j = 1; j<=i ; j++ ){
                                                        System.out.print ("* ");
                                        }

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

==========

==========
Chapter-3
==========

1) Arrays

2) Strings

3) StringBuffer

4) StringBuilder

5) Command Line Arguments

Variable ===> It is used to store the data

Datatype ===> It is used to specify type of the data

```
                                // variable declaration
                                int a ;

                                // variable intialization
                                a = 20 ;

                                int i = 45; // this is valid

                                int b = 20, 30 ;    // this is in-valid (we can
store only one value)
```

```
// store one student subject wise marks  (we need 6 variables for 6 subjects for
one student)

int sub1 = 78;

int sub2 = 98;

int sub3 = 79;

int sub4 = 90;

int sub5 = 95;

int sub6 = 87;

// i want to store 50 students subject wise marks ( we need 300 variables to store
all

students marks)
```

=> This is not recommended because so many variables we have to create

=> To overcome this problem we can go for 'Arrays' concept

========
Arrays
========

-> It is a referenced data type

-> It is used to store multiple values

-> In Arrays, size is fixed (we can't change the size in runtime)

-> Arrays supports only Homogenious elements (same type of elements)

Defination : Array is a container which is used to store collection of elements with same

 data type.

Syntax

```
// Array Declaration
datatype [ ]  variableName;

dataype variableName [ ] ;

datatype  [ ]variableName ;


// Array Creation
variableName = new datatype [ size ] ;


datatype[ ] variableName = new datatype [ size ] ;


Ex :      int[ ]  arr = new int [ 5 ];
```

Note: At the time of creating the array the size is mandatory

=> Array Size represents how many values we can store into Array

=> Array will store the data based on indexes

=> Array index always will start from zero (0)

```
class ArrayDemo {

        public static void main(String... args){

                        int [ ]  arr  = new int [ 3 ] ;

                        arr[0] = 100;
                        arr[1] = 101;
                        arr[2] = 102;

                        System.out.println ( arr [ 0 ] ) ;
                        System.out.println ( arr [ 1 ] ) ;
                        System.out.println ( arr [ 2 ] ) ;
        }
}
```

=> We can find size of the array using length property

```
class ArrayDemo {
        public static void main(String... args){

                        int [ ]  arr  = new int [ 5 ] ;
                        arr[0] = 100;
                        arr[1] = 101;
                        arr[2] = 102;

                        System.out.println ( arr.length ) ;
        }
}

-----------------------------------------------------------------
class ArrayDemo {

    public static void main (String... args) {

                        int arr[ ]  = new int [ 3 ] ;

                        arr[0] = 100;
                        arr[1] = 101;
                        arr[2] = 102;

                        for( int i = 0 ; i < arr.length ; i ++ ) {
                                        System.out.println ( arr [i] );
                        }

        }
}

-----------------------------------------------

class ArrayDemo {

        public static void main(String... args) {

                        boolean [ ]  arr  = new boolean [ 3 ] ;

                        arr [2] = true;

                        for( int i = 0 ; i < arr.length ; i ++ ) {
                                        System.out.println ( arr [i] );
                        }
        }
}
-------------------------------------------------------------

class ArrayDemo {
        public static void main(String... args) {

                        int arr [ ]  = { 101, 102, 103, 104 } ;

                        for ( int i = 0; i < arr.length; i++){
                                        System.out.println(arr[i]);
                        }
        }
}

-----------------------------------------------------------------------------------
------

class ArrayDemo {

        public static void main(String... args) {

                        int arr [ ]  = { 101, 102, 103, 104 } ;
```

```
                        arr [ 0 ] = 200 ;       // it will update 0th index value
from 101 to 200

                        arr [ 1 ] = 300 ;  // it will update 1st index value from
102 to 300

                        System.out.println ( arr [ 101 ] ) ;  // AIOBE ( 101 index
not available

 in array )
        }
}
```

Note: Array size should be positive integer only

-> Decimal value can't be used for Array Size

-> Negative value also can't be used for Array Size

---

1) Write a java program to find min and max elements in the array

```
                int arr [ ] = { 15, 8, 9, 2, 11, 4 }
```

```
============================== Program with Sort method
==============================
import java.util.*;

class ArrayDemo {

        public static void main(String... args) {

                        int arr [ ] = { 15, 8, 9, 2, 11, 4 } ;

                        Arrays.sort(arr);

                        System.out.println ("Min Element : " + arr [ 0 ] );

                        System.out.println("Max Element : " + arr [ arr.length - 1
] );
        }
}
=========================Program without sort
method==================================
class ArrayDemo {

        public static void main(String... args) {

                        int arr [ ] = { 15, 8, 9, 2, 11, 4 } ;

                        int min = arr [ 0 ];
                        int max = arr [ 0 ] ;

                        for ( int i = 0 ; i < arr.length ; i++){
                                if ( arr [ i ] > max ) {
                                        max = arr [ i ] ;
                                }
                                if ( arr [ i ] < min ){
                                        min  = arr [ i ];
                                }
                        }
```

```
                        System.out.println ( " Min Element :: " + min );
                        System.out.println (" Max Element :: " + max);
        }
}
```

Q-2) Write a java program to reverse an array ?

```
                int arr [ ] = { 15, 8, 9, 2, 11, 4 } ;


class ArrayDemo {

        public static void main(String... args) {

                        int arr [ ] = { 15, 8, 9, 2, 11, 4, 7 } ;

                        int temp = 0;

                        for ( int i = 0; i < arr.length / 2 ; i++ ){
                                temp = arr [ i ];
                                arr [ i ] = arr [ arr.length - 1 - i ];
                                arr [ arr.length-1 - i ] = temp;
                        }

                        for( int n : arr){
                                System.out.print ( n + " " );
                        }

        }
}
```

========== Array Elements Printing - 3 ways==========

```
import java.util.Arrays;

public class ArraySorting {

        public static void main(String[] args) {

                int arr[] = { 5, 8, 2, 6, 9, 3 };

                for (int i = 0; i < arr.length; i++) {
                        System.out.print(arr[i] + " ");
                }
                System.out.println();

                for (int x : arr) {
                        System.out.print(x + " ");
                }
                System.out.println();

                System.out.println(Arrays.toString(arr));
        }

}
```

==========

```
public class ArraySorting {

        public static void main(String[] args) {

                int arr[] = { 5, 8, 2, 6, 9, 3 };
```

```
                int temp = 0;

                for (int i = 0; i < arr.length; i++) {

                        for (int j = i + 1; j < arr.length; j++) {
                                if ( arr[i] < arr[j]) {
                                        temp = arr[i];
                                        arr[i] = arr[j];
                                        arr[j] = temp;
                                }
                        }
                }
                System.out.println(Arrays.toString(arr));
        }
}
```

==========

```
public class SumMatchingPairs {

        public static void main(String[] args) {

                int arr[] = { 3, 5, 1, 6, 2, 7, 9 };

                int sum = 8;

                for (int i = 0; i < arr.length; i++) {

                        for (int j = i + 1; j < arr.length; j++) {
                                if (arr[i] + arr[j] == sum) {
                                        System.out.println(arr[i] + "+" + arr[j] +
"=" + sum);
                                }
                        }
                }
        }
}
```

==========

```
public class NonRepeatedElements {

        public static void main(String[] args) {

                int arr[] = { 2, 3, 2, 1, 3, 4, 5 };

                for (int i = 0; i < arr.length; i++) {
                        int count = 0;

                        for (int j = 0; j < arr.length; j++) {
                                if (arr[i] == arr[j]) {
                                        count++;
                                }
                        }
                        if (count == 1) {
                                System.out.print(arr[i] + " ");
                        }
                }
        }
}
```

=> Arrays are divided into 2 types

1) Single Dimensional Array

```
                                int arr [ ] =  new int [ size ] ;
```

2) Multi Dimensional Array

```
                                int   arr [ ] [ ]  = new int [ rowsSize ]  [
columnsSize ] ;


import java.util.*;

class MutliDimensionArray{

                public static void main(String... args){

                                        int arr [ ] [ ] = new int [ 2 ] [ 2 ];

                                        arr [0] [0] = 100 ;
                                        arr [0] [1] = 200;
                                        arr [1] [0] = 300;
                                        arr [1] [1] = 400;

                                        for (int i = 0; i < arr.length ; i ++ ){

                                                        for ( int j = 0; j
<arr.length ; j ++ ){

System.out.println( arr[i] [j] );
                                                        }

                                        }

                }
}
-------------------------------------------------------------------------------

import java.util.Arrays;

public class Demo {

        public static void main(String[] args) {
                int arr[][] = new int[2][2];

                arr[0][0] = 100;
                arr[0][1] = 200;
                arr[1][0] = 300;
                arr[1][1] = 400;

                for (int[] ar : arr) {
                        System.out.println(Arrays.toString(ar));
                }
        }
}
-------------------------------------------------------------------------------

import java.util.Arrays;

public class Demo {

        public static void main(String[] args) {
                int arr[][] = { { 100, 200 }, { 300, 400 } };

                for (int[] ar : arr) {
                        System.out.println(Arrays.toString(ar));
                }
        }
}
-------------------------------------------------------------------------------

public class Demo {
```

```
        public static void main(String[] args) {
                int arr[][] = { { 100, 200 }, { 300, 400 } };

                for (int[] ar : arr) {
                        for(int x : ar) {
                                System.out.println(x);
                        }
                }
        }
}
```

=======
Strings
=======

-> String is a pre-defined class available in java.lang package

-> String we can use as a data type also (Referenced Data Type)

        Note: Every java class can be used as a referenced data type

-> String is used to store group of characters

```
                        Ex :   String name = "abc" ;
```

-> String is immutable in java (can't be modified)

-> We can create String object in 2 ways

```
                                // approach - 1  (string literal)
                                String  name = "ashokit";

                                // approach -2 (using new operator)
                                String str = new String ("ashokit");
```

====================
String Constant Pool
====================

-> It is special memory in JVM to store String objects

-> It will not allow us to create duplicate objects

```
                                String s1 =  "hi" ;

                                String s2 = "hi" ;
```

-> s1 and s2 objects are having same content hence only one object will be created and

two variables will be pointed to same object.

```
class StringDemo {

                public static void main(String[ ] args) {

                        String s1 =  "hi" ;
                        String s2 = "hi" ;

                        if ( s1 == s2 ) {
                                System.out.println (" Both are same ");
                        }else {
                                System.out.println(" Both are not same" );
                        }

                }
}
```

=> If we create String objects using 'new' operator always new object will be created

in Heap area.

```
class StringDemo {
                public static void main(String[ ] args) {
                        String s1 =  new String ("hello") ;
                        String s2 = new String ("hello") ;
                        if ( s1 == s2 ) {
                                System.out.println (" Both are same ");
                        }else {
                                System.out.println(" Both are not same" );
                        }
                }
}
```

Note: In Strings == will compare address of the objects not content.

Q) How many objects will be created ?

```
String s1 = "ashokit" ;   // 1 obj
String s2  = "ashokit";  // 1 obj
String s3 = new String("ashokit");  // 2 objs
String s4 = new String("ashokit");  // 3 objs
String s5 = new String("hello"); // 5 objs
String s6 = new String("hi"); // 7 objs
```

s1 == s2 =====> true

s2 == s3 =====> false

s3 == s4 =====> false

s5 == s6 ====> false

String class Methods

charAt ( ) => To get a character based on given index

```
String s1 = "ashokit";
System.out.println(s1.charAt(0));
```

length ( ) => To get size of string (no.of characters available in String)

```
String s1 = "ashokit";
System.out.println(s1.length( ) );
```

concat ( ) => To join two strings (appending)

```
String s1 = "ashok";
String s2 = "it";
```

```
                                                    String s3 = s1.concat(s2);
                                                    // String s4 = s1 + s2 ;
                                                    System.out.println(s3);
```

equals ( ) => To compare content of two Strings

```
                                                    String s1 = "hi";
                                                    String s2 = "hello";
                                                    System.out.println( s1.equals(s2)
);
```

Note: In Strings, == operator will compare address of string objects where as 'equals( ) ' method will compare content of the objects.

replace ( ) => To repalce chars with another chars

```
                                    String s1 = "hyderabad";
                                    String s2 = s1.replace("bad", "good");
                                    System.out.println(s2);
```

toUpperCase ( ) => To convert string to uppercase string

```
                                            s1.toUpperCase( ) ;
```

toLowerCase ( ) => To convert String to lowercase String

```
                                            s1.toLowerCase ( ) ;
```

indexOf ( ) => To get first occurance of char

```
                                    System.out.println ( s1.indexOf('a') ) ;
```

lastIndexOf ( ) => to get last occurance of char

```
                                    System.out.println ( s1.lastIndexOf('a') ) ;
```

Note: If given char is not available then it will return '-1'

substring ( ) => It is used to get some part of the string. It will take start index & end index.

```
                            start-index : inclusive
                            end-index : exclusive
                            System.out.println( s1.substring(0,5)
```

Note: If we don't give end index, it will print from start index to last index.

split ( ) => It is used to split the string based on delimiter (seperator)

```
                                    String s2  = "hi@hello@how are@you";
                                    String [ ] arr = s2.split ("@");
                                    System.out.println(Arrays.toString(arr));
```

valueOf ( ) -> It is used convert any type value into String type

```
int a = 10 ;
int b = 20 ;

a + b   ===> 30

String s1 = "10";
String s2 = "20";

s1 + s2  ==> "1020"

String.valueOf (a) + String.valueOf (b) ==>
1020
                "10"        +  "20"     ==>
1020
```

Note: valueOf ( ) is a static method in String class. Static methods will be called using classname. Non-Static methods will be called using Object.

startsWith ( ) => It is used to check given String is starting with particular char(s) or not

```
String str = "ashokit";

str.startsWith("a") ;    =====>
true

str.startsWith("z") ;     =====>
false
```

endsWith ( ) => It is used to check given String is ending particular char(s) or not

```
String str = "ashokit";

str.endsWith("it");   ===> true

str.endsWith("good") ===> false
```

trim ( ) ==> It is used to remove starting and ending spaces of String

```
String str = "     hello    ";

str.trim ( );
```

intern ( ) => It is used to access the object from scp

```
String s1 = "hi";
String s2 = s1.intern ( );
s1 == s2 ===> true
```

toString ( ) => It is used to convert object into string format.

toCharArray ( ) : This method is used to convert String to char array

```
String s1 = "java";
```

```
                                char arr[ ]  = s1.toCharArray ( );
```

StringBuffer

-> It is a predefined class available in java.lang package

-> It is used to store group of characters

-> StringBuffer is a mutable object (We can modify the content)

-> StringBuffer is thread-safe (only one thread can access at a time)

```
                StringBuffer sb  = new StringBuffer ( "hello" );
```

```
class SBDemo {

                public static void main (String[ ] args){

                                        StringBuffer sb = new
StringBuffer("hello");
                                        sb.append(" hi");
                                        System.out.println(sb);

                                        StringBuffer sb1 = new
StringBuffer("java");
                                        System.out.println(sb1.reverse());

                                        String s1 = "ashok";
                                        s1.concat("it");
                                        System.out.println(s1);

                }
}
```

StringBuilder

-> StringBuilder is a predefined class available in java.lang package

-> This class introduced in JDK 1.5v

-> StringBuilder objects are mutable (content can be modified)

-> StringBuilder objects are not thread safe (Multiple threads can access at a time)

```
                        StringBuilder sb = new StringBuilder ( "java" );

                        sb.length ( );

                        sb.append("program");
```

```
class BuilderDemo {

       public static void main (String[ ]  args){

                        StringBuilder sb =  new StringBuilder("java") ;
                        System.out.println(sb.length());
```

```
                        sb.append("program");

                        System.out.println(sb);
                        System.out.println(sb.length());
                }
}
```

Q) What is the difference between String, StringBuffer & StringBuilder ?

String -> Immutable --> Jdk 1.0

StringBuffer---> Mutable + Thread-Safe --> Jdk 1.0

StringBuilder --> Mutable + Not-Thread-Safe --> Introduced in jdk 1.5v

---

Command Line Arguments

-> Arguments means Values

-> Commadline args are used to supply dynamic values as input for our program

-> Cmd Args will be recieved by main method

-> Cmd Args default data type is String

-> We can pass multiple cmd args, they will be stored into one array (String [ ])

```
// Program with command line arguments
class CmdArgs {
                public static void main (String [ ] a){

                                String s1 = a [0] ;
                                String s2 = a [1];
                                String s3 = a [2];

                                System.out.println( s1 + s2 + s3 );
                }
}
```

```
> javac CmdArgs.java
> java CmdArgs  ashok it hyd
```

```
// Write a java program to perform sum of two numbers using command line Arguments

class CmdArgs {
                public static void main (String [ ] a) {

                                System.out.println ("Total Cmd Args :: " + a.length
);

                                String s1 = a [0];
```

```
                                String s2 = a [1];

                                int x = Integer.parseInt (s1);
                                int y = Integer.parseInt(s2);

                                System.out.println (x + y);

                }

}

> javac CmdArgs.java

> java CmdArgs 10 20

> java CmdArgs 10 20 30


============================
class CharOcc {

                public static void main (String[ ] args){

                                        String s = "java";
                                        char ch  = 'a';

                                        char arr[ ] = s.toCharArray ( );

                                        int count = 0;

                                        for( int i = 0; i < arr.length ; i ++ ) {
                                                        if ( arr[i] == 'a' ) {
                                                                        count ++ ;

                                                        }
                                        }

                                System.out.println(count);
                }
}
==========================================
class StringReverse {

                        public static void main(String[ ] args){

                                        String s = "java";

                                        String rev = "";

                                        for( int i = s.length( ) - 1 ; i >=0 ;  i--
){

                                                rev = rev + s.charAt ( i ) ; //avaj
                                        }
                        }
}
-----------------------------------------------------------
class StringReverse {

                        public static void main(String[ ] args){

                                        String s = args[0];

                                        String rev = "";

                                        for( int i = s.length( ) - 1 ; i >=0 ;  i--
){
```

```
                                        rev = rev + s.charAt ( i ) ;
                                }

                        System.out.println(rev);
                }
}

-------------------------------------------------------------------------
class Palindrome {

        public static void main(String[ ] args){

                                String s = args [0] ;
                                String s1 = "";

                                for(int i = s.length ( ) -1 ; i >=0 ; i --
){
                                        s1 = s1 + s.charAt (i);
                                }

                                if( s.equals(s1) ){

System.out.println("Palindrome");
                                }else{
                                        System.out.println("Not
Palindrome");
                                }
        }
}

----------------------------

class Palindrome {

        public static void main(String[ ] args){

                                String s = args [0] ;

                                StringBuffer sb = new StringBuffer(s);

                                sb.reverse() ;

                                String s1 = sb.toString( );

                                if( s.equals ( s1 ) ) {
                                        System.out.println
("Palindrome");
                                }else{
                                        System.out.println ("Not
Palindrome");
                                }

        }

}

-------------------------------------------------------------------------
import java.util.*;

class Anagram {

        public static void main(String[ ] args){

                        String s1 = args[ 0 ] ;
                        String s2 = args[ 1 ];

                        if ( s1.length ( ) != s2.length () ){
                                System.out.println("Given Strings are not
```

```
anagrams");
                                        return ;
                        }

                        char a[ ] = s1.toCharArray ( );

                        char b[ ] = s2.toCharArray ( ) ;

                        Arrays.sort ( a );
                        Arrays.sort ( b );

                        boolean flag = Arrays.equals(a, b);
                        if( flag  ){
                                System.out.println("Given strings are anagrams");
                        }else{
                                System.out.println("Given Strings are not
anagrams");
                        }


        }

}

------------------------

class SwapStrings{

                public static void main(String[ ] args){


                                String a = "java";   // 4
                                String b = "program"; // 7

                                a = a+b; // 11

                                b = a.substring(0, a.length() - b.length());

                                a = a.substring(b.length());

                                System.out.println(" a = " + a);
                                System.out.println(" b = " + b);

                }

}
-----------------------------------------------------

class RemoveVowels {


                        public static void main(String... args){

                                        String s = "hello, i love my india";

                                        s = s.replaceAll ("[aeiouAEIOU]", "");

                                        System.out.println(s);
                        }

}

-----------------------------------------------------------------------
class ReverseEachWord {

                        public static void main(String... args){

                                        String s = "Hello My Friend";
```

```
                                        String[ ]  arr = s.split(" ");

                                        for(int i = 0; i < arr.length ; i++){
                                                String x = arr [ i ];

                                                StringBuffer sb = new
StringBuffer(x);
                                                sb.reverse( );
                                                System.out.print (sb+" ");

                                        }
                        }

}
-------------------------------------------------------------------------

class WordCount {

                        public static void main(String... args){

                                        String s = "Hello    Hello           My
   Friend";

                                        String[ ] arr = s.split("\\s+");

                                        System.out.println(arr.length);
                        }

}

-----------------------------------------------------------------------------

class WordCount {

                        public static void main(String... args){

                                        String s = "      Hello    Hello
My        Friend";

                                        String[ ] arr = s.trim().split("\\s+");

                                        System.out.println(arr.length);
                        }

}

-------------------------------------------------------------------------------
================================================
Chapter-4 : OOPS (Object Oriented Programming System)
================================================

-> Programming languages are divided into 2 types

1) Procedure Oriented

                Ex: C, Cobol, Pascal etc.....

2) Object Oriented

                Ex: Java, C#, Python etc.....

-> In Procedure Oriented programming language, we will develop functions &
procedures

-> If we want to add more functionalities then we need to develop more functions

-> Maintaining & Managing more functions is difficult task
```

-> In PoP, data is exposed globally

-> In Pop, there is no security

-> If we want to develop a project using OOP lanaguage then we have to use Classes & Objects

-> Any language which follows OOPS Principles is called as OOP Language

-> Object Oriented languages provides security for our data

-> The main advantage of OOPS is code re-usability

==========
OOPS Principles
==========

1) Encapsulation

2) Abstraction

3) Polymorphism

4) Inhertience

==========
Encapsulation
==========

-> Encapsulation is used to combine our variables & methods as single entity / unit

-> Encapsulation provides data hiding

-> We can achieve encapsulation using Classes

```
class  Demo {

        //variables

        // methods
}
```

==========
Abstraction
==========

-> Abstraction means hiding un-necessary data and providing only required data

-> We can achieve Abstraction using interfaces & abstract classes

Ex : we will not bother about how laptop working internally
        We will not bother about how car engine starting internally

==========
Polymorphism
==========

-> Exhibiting multiple behaviours based on situation is called as Polymorphism

Ex:-1 : in below sceario + symbol having 2 diffenent behaviuours

10 + 20 ===> 30   (Here + is adding)

"hi" + "hello" ==> hihello (here + is concatinating)

Ex:-2:

When i come to class i will behave like a trainer

When i go to ofc i will behave like a employee

When i go to home i will behave like a family member

==========
Inheritence
==========

-> Extending the properties from one class to another class is called as Inheritence

Ex: child will inherit the properties from parent

-> The main aim of inhertience is code re-usability

Note: In java, one child can't inherit properties from two parents at a time

======
Class
======

-> Class is a plan or model or template

-> Class is a blue print of object

-> Class is used to declare variables & methods

-> Project means collection of classes

-> Once class is created then we can create any no.of objects for a class

-> 'class' keyword is used to create Classes in java

```
class  <ClassName> {

        // variables
        // methods

}
```

-> Classes will not exist physically

========
Object
========

-> Any real-world entity is called as Object

-> Objects exist physically

-> Objects will be created based on the Classes

-> Without having the class, we can' create object (class is mandatory to create objects)

-> Object creation means allocating memory in JVM

-> 'new' keyword is used to create the objects

```
ClassName   refVariable = new ClassName ( );

User  u1  = new User ( );

User u2 = new User ( ) ;
```

-> Objects will be created by JVM in the runtime

-> Objects will be created in heap area.

-> If object is not using then garbage Collector will remove that object from heap

-> Garbage Collector is responsible for memory cleanup activities in JVM heap area.

-> Garbage Collector will remove un-used objects from heap.

-> Garbage Collector will be managed & controlled by JVM only.

Note: Programmer don't have control on Garbage Collector.

==========
What is Hash Code
==========

-> JVM will assign unique hashcode for every object

-> No two objects will have same hashcode

-> We can get hashcode of the object by calling java.lang.Object class hashCode () method.

```
u1.hashCode ( );
```

Note: java.lang.Object class is by default parent class for all java classes.

```
public class User {

	public static void main(String[] args) {
		User u1 = new User();
		System.out.println(u1.hashCode());

		User u2 = new User();
		System.out.println(u2.hashCode());

		User u3 = new User();
		System.out.println(u3.hashCode());
	}
}
```

==========
Variables
==========

-> Variables are used to store the data

```
int a  = 10 ;
```

User u1 = new User ( );

Student s1 = new Student ( );

-> Variables are divided into 3 types

a) Global Variables / instance variables / non-static variables

b) static variables

c) local variables

==========
instance variables
==========

-> Variables which are declared inside the class and outside the method are called as instance variables

-> instance variables can be accessed by all the methods available in the class thats why the are called as Global Variables.

-> Initialization is optional for instance variables

-> Instance variables are called as Object variables

-> When we create the object, then only memory will be allocated for instance variables

Note: If we create 2 objects, then 2 times memory will be allocated for instace variables

-> If we don't intialize instance variable, it will be intialized with default value based on datatype when the object is created

-> Every Object will maintain its own copy of the instance variable

```
public class User {

	int age;

	public static void main(String[] args) {
		User raju = new User();
		raju.age = 20;
		System.out.println(raju.age);

		User rani = new User();
		rani.age = 25;
		System.out.println(rani.age);

		User ashok = new User();
	}
}
```

==========
Static Variables
==========

-> The variables which are declared inside the class and outside the method with 'static'

keyword are called as static variables

-> Static variables are class level variables

-> When class is loaded into JVM then immediatley memory will be allocated for static variables

-> Memory will be allocated for static variables only once when the class is loaded into JVM

-> All objects of the class will maintain same copy of the static variables

-> Static variables we will access using classname

```
public class Student {

	String name;
	String email;
	long phno;
	static String institute;

	public static void main(String[] args) {

		Student.institute = "ashokit";

		Student ankit = new Student();
		ankit.name = "Ankit";

		Student goutham = new Student();
		goutham.name = "Goutham";
	}
}
```

==========
When to declare variable as static or non-static ?
==========

-> If we want to store different value based on object then use instance variable

-> If we want to store same value for all objects then use static variable

==========
Local Variables
==========

-> The variables which are declared inside the method or constructor or block are called

as Local Variables

-> If we declare a variable with in the method, then that variable can be used / accessed

only with in that method

-> Before using local variables, we have to intialize them

-> If we create a variable with in the method, memory will be allocated for that variable

when that method is executing. After that method execution completed, local variable will

be deleted from memory

```
class Demo {

 public static void main(String[ ] args){
                                int a = 20;
                                int b = 20;

                                System.out.println(a);
                                System.out.prinltn(b);
  }
}
```

========
Methods
========

-> Methods are used to perform some operation / action

-> In a class we can write any no.of methods including main method

Note: JVM will always invokes main ( ) method

-> If we want to execute our method then we have to invoke / call our methods from main

( ) method.

```
returnType   <methodName>   (param1, param2, para3..... paramN) {

                //logic

                return value;

}
```

-> Every method contains 2 parts

1) Method Declaration
2) Method Body

What is Method Declaration ?

Method declaration means we are going to decide what is the name of the method ,

what are the parameters it will take and what kind of value is return by the method.

syntax:

```
                        returntype  methodname (list of parameters);
```

What is returntype ?

-> returntype is data type that indicates what type of value is return by the particular

 method.
-> returntype can be any primitive type or array type or reference type
-> if method does not return any value then return type must be specified using
 a java keyword called " void ".
-> specifying returntype is mandatory

What is method name ?

-> To identify and access the method there is a suitable name is given for a method

which
 is called as method name.
-> a methodname can be any valid java identifier.
-> specifying method name is mandatory

What are method parameters ?

-> parameters are the variables that will receive the values that are passed into the.

 particular method on which data method will perform the operations.

-> we can write 0 or more number of parameters of any primitive type or array type or

 reference type

-> specifying parameters is optional.

Method body / Method Definition / Method implementation

-> Method body means we are going to write the group of statements that are executed by
 the method.
-> A method body can be written in between a pair of curly braces

syntax:

```
returntype methodname(list of parameters)
{
        //statements;
        return value;
}
```

-> here we can write 0 or more number of statements in between the pair of curly braces.
-> when we write 0 statements then it is called as null implementation

-> if the return type is specified as other than void then we must return a value from our method
using java keyword called " return ".

syntax:
```
return value;
```

-       the datatype of the value we return must be match with the datatype that we specify as return type.
-       but if return type specified as void then we must not write any return value statement.

-> In java we can create any number of methods which are in any of the following 4 combinations of methods

1. method without return type, without parameters
2. method without return type, with parameters
3. method with return type, without parameters
4. method with return type, with parameters

// Write a method to print a msg on console

```
void hello ( ) {
        System.out.println("Hello My Friend");
}
```

```
// Take 2 numbers as input and return sum as ouput
int  add ( int a, int b ) {
        int c = a + b ;
        return c;
}



// take 2 names as input, concat them and print on console
void  fullname (String fname, String lname) {
        String name = fname + lname;
        S.o.p(name);
}

// take person age as input, if person age >=18 then return true else return false
boolean  check (int age ){
        if ( age >= 18 ) {
                return true;
        } else {
                return false;
        }
}
```

```
=====================
Types of Methods
=====================
```

```
=> Methods are divided into 2 types
1) instance methods   ---> Object level methods
2) static methods ----> Class level method

->  instance method will be called by using Object
-> static method will be called by using Class

-> When we write methods in java class, by default jvm will not execute them
-> To execute our methods we have to call them

Note: JVM will start our program execution from main method. Main method is called
as entry point
for JVM execution
```

```
// this is valid
void   m1 ( ) {
```

```
}

// this is valid
void m1 (int a, float f){

}

// this is valid
int  add (int a, int b){
        int c = a + b;
        return c;
}

// this is valid
int add (int a, int b){
        return a +  b;
}

// this is valid
int div (int a, int b){
        return a / b ;
}


// below method is invalid (method return type is int but it is trying to return
string value)
int  m1 (String name){

        Stirng s1 = name.toUpperCase( ) ;

        return s1;

}

// this is valid
boolean   m2 ( int a, int b){

        if (a > b )
                return true;
        else
                return false;

}

// this is valid
boolean m2 (int a, int b){

        return a > b ;

}


// this is invalid because it is having 2 return types
String void m2(){

        return "hi";

}

// this is valid
boolean   m3( ) {

        return true ;
}

// this is valid
void   c1 (int [ ] arr ){
```

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


-----------------------------------------------
public class Student {

        public static void main(String[] args) {
                System.out.println("Hi, i am a student");

                Student s1 = new Student();
                s1.hello();

                Student.greet();
        }

        void hello() {
                System.out.println("Hello My Friend...");
        }

        static void greet() {
                System.out.println("Good Evening..");
        }
}
-----------------------------------------------------------------------
import java.util.Arrays;

public class Methods {

        public static void main(String[] args) {

                //object creation
                Methods m = new Methods();

                int[] ar = { 1, 2, 3 };
                m.print(ar);

                m.fullname("ashok", "it");
        }

        void fullname(String fname, String lname) {
                String name = fname + lname;
                System.out.println(name);
        }

        void print(int[] arr) {
                System.out.println(Arrays.toString(arr));
        }
}
-------------------------------------------------------------------------

1)  Write a java method to find max element in given array

                int   findMaxElement (int [ ] arr ) {
                                // logic
                          return maxElement;
                }

2)  Write a java method to find length of given name

                int findLength ( String name) {

                                int length = name.length ( ) ;

                                retunr length;
                }
```

3) Write a java method to perform sum of two numbers

```
int  add (int a, int b){
                //logic
                return  a+b ;
}
```

4) Write a java method to concat firstname and lastname

```
String  concatNames (String fname, String lname){
                return fname + lname;
}
```

5) Write a java method to display person is elgible for vote or not

```
boolean  check (int age){
                return age >= 18 ;
}
void  check(int age){
        if(age > = 18){
                s.o.p("eligible");
        }else{
                s.o.p("not-eligible");
        }
}
```

6) Write a java method to reverse given array

```
                int[ ]  reverseArray (int[ ] arr) {
                                //logic
                }
                void  reverseArray (int[ ] arr ) {  ... }
```

6) Write a java method to convert the name into uppercase characters

```
String   convertUppercase ( String name ) {
        return name.toUpperCase( );
}
```

```
int ar[ ]  = {1,2,3};
String s= Arrays.toString ( ar ) ;

String str = new String("hi");
StringBuffer sb = new StringBuffer(str);
String s1 = sb.reverse ( );
String line = br.readLine ( );
String name = "ashokit";
// int length ( )
```

```
int x  = name.length ( ) ;

// boolean endsWith (String str)
boolean z  = name.endsWith("it");

// char charAt (int index)
char ch   = name.charAt (0);
```

```
public class Driver {

        public static void main(String[] args) {
                Driver d = new Driver(); // obj creation

                int x = d.add(10, 20); // calling the method
                System.out.println(x); // printing the output
        }

        // instance method
        int add(int a, int b) {
                int c = a + b;
                return c;
        }
}
```

==========
Working with Methods using Objects
==========

-> Object means physical entity

-> Objects are used to store the data

-> Object will be created based on class name

-> To create the object we will use 'new' operator

-> Object will be stored in heap area

-> For every Obect jvm will assign one unique hashcode

-> In realtime projects, data will play the major role

-> Java is a OOP language, so everything will be represented in the form of Objects

-> In Realtime Projects, mainely our methods will deal with Objects only

// Write a java method to print data available in the Student object

```
public class Driver {

        public static void main(String[] args) {
                Driver d = new Driver();

                Student s = new Student();
                s.id = 101;
                s.name = "raju";

                d.print(s);
        }
```

```
	void print(Student s) {
		System.out.println(s.id + "  " + s.name);
	}

}

class Student {
	int id;
	String name;
}


//  Take employee class with id and salary as properties
//  Take Driver class and write the method to print employee object data & call the print method from main method

class Employee {
	int id;
	double salary;
}
public class Driver {

	public static void main(String[] args) {
		Driver d = new Driver();

		Employee e = new Employee();
		e.id = 101;
		e.salary = 15000.00;

		d.print(e);
	}

	void print(Employee e) {
		System.out.println(e.id + "--" + e.salary);
	}

}


// Take Product class with productId, productName, productPrice as properties

// Create Driver class with print ( ) method to print product data


class Product {
	int pid;
	String pname;
	double price;
}

class Driver {

	void print(Product p) {
		System.out.println(p.pid + " " + p.pname + " " + p.price);
	}

	public static void main(String[] args) {
		Driver d = new Driver();

		Product p = new Product();
		p.pid = 101;
		p.pname = "Mouse";
		p.price = 450.00;

		d.print(p);
	}
}
```

```
// Take Docter class with docterName, docterAge  as properties
// Create Driver class with print ( ) method to print Docter data


class Docter {
	String name;
	int age;
}

class Driver {

	void print(Docter d) {
		System.out.println(d.name + " " + d.age);
	}

	public static void main(String[] args) {
		Driver d = new Driver();

		Docter d1 = new Docter();
		d1.name = "Rathod";
		d1.age = 29;
		d.print(d1);
	}
}
-------------------------------------------------------------------------------
------------
// Take a Player class with id, name, age properties

class Player {

	int id;
	String name;
	int age;
}

// Take driver class to print Player Data

class Driver {

	void print (Player p1) {
		s.o.p(p1.id +"--"+ p1.name+""+p1.age);
	}

		psvm(String... args){
			Driver d = new Driver( );

			Player p2 = new Player ( );
			//set data

			d.print (p2);
		}

}

-------------------------------------------------------------------------------
---------------

// write a java method which will give Person object with data

package ashokit;

class Driver {

	public static void main(String[] args) {
		Driver d = new Driver();
		Person p = d.m1();
		System.out.println(p.id + "--" + p.name + "--" + p.age);
```

```
        }

        Person m1() {
                Person p = new Person();
                p.id = 101;
                p.name = "Rani";
                p.age = 32;

                return p;
        }

}

class Person {
        int id;
        String name;
        int age;
}
```

```
// Write a java method to return College data (id, name)


package ashokit;

class Driver {

        College m1() {

                College c = new College();
                c.id = 101;
                c.name = "HITM";

                return c;
        }

        public static void main(String[] args) {
                Driver d = new Driver();
                College c = d.m1();
                System.out.println(c.id + "--" + c.name);
        }

}

class College {
        int id;
        String name;
}
```

Raju Data (101, Raju, 30)

Rani Data (102 , Rani, 32)

// Write a java method which will take id as input. if id is 101 then method should return Raju object
if id is 102 then method should return Rani object.

```
package ashokit;

class Driver {
```

```
	public static void main(String[] args) {
		Driver d = new Driver();

		//read id from keyboard

		Person person = d.m1(120);
		System.out.println(person.id + "--" + person.name);
	}

	Person m1(int id) {
		Person p = new Person();
		if (id == 101) {
			p.id = 101;
			p.name = "Raju";
			p.age = 30;
		} else if (id == 102) {
			p.id = 102;
			p.name = "Rani";
			p.age = 32;
		}
		return p;
	}
}

class Person {
	int id;
	String name;
	int age;
}
```

// write a java method which will return cricket player data based on player number

Player Data --> id, name, age

7 ----> Dhoni

18 ---> Kohli

45 ---> Rohit Sharma

```
package ashokit;

class Driver {

	public static void main(String[] args) {
		Driver d = new Driver();
		Player p = d.m1(45);
		System.out.println(p.id + "--" + p.name + "--" + p.age);
	}

	Player m1(int id) {

		Player p = new Player();

		if (id == 7) { //false
			p.id = 7;
			p.name = "dhoni";
			p.age = 40;
		} else if (id == 18) { // false
			p.id = 18;
			p.name = "kohli";
			p.age = 34;
		} else if (id == 45) { // true
```

```
                        p.id = 45;
                        p.name = "Rohit";
                        p.age = 38;
                }
                return p;
        }

}

class Player {
        int id;
        String name;
        int age;
}
```

```
// Write a java method to return University data based on unvesity ID

101 -----> id - 101, name - Oxford

102 ----> id- 102, name - Standford


public class University {

        int id;
        String name;

        public static void main(String[] args) {
                University u = m1(101);
                System.out.println(u.id + "--" + u.name);
                String str = u.m2();
                System.out.println(str);
        }

        String m2() {
                String s = "hello";
                return s;
        }

        static University m1(int id) {
                University u = new University();
                if (id == 101) {
                        u.id = 101;
                        u.name = "Oxford";
                } else if (id == 102) {
                        u.id = 102;
                        u.name = "Standford";
                }
                return u;
        }
}


// Write a java class with two methods

// first method should take 2 Person objects as input

                Approach-1:

                        void m1(Person p1, Person p2){
                                        // logic
                        }
```

```
		Approach-2 :

			void m1 (Person[ ] p){

			}


// second method should give 3 Person Objects as ouput

			Person[ ] m2(){
				// logic
			}
```

```
package ashokit;

public class Person {

	int id;
	String name;

	Person[] m2() {

		Person p1 = new Person();
		p1.id = 101;
		p1.name = "Raju";

		Person p2 = new Person();
		p2.id = 102;
		p2.name = "Rani";

		Person p3 = new Person();
		p3.id = 103;
		p3.name = "Anil";

		Person[] arr = { p1, p2, p3 };

		return arr;

	}

	void m1(Person p1, Person p2) {
		System.out.println(p1.id + "--" + p1.name);
		System.out.println(p2.id + "--" + p2.name);
	}

	public static void main(String[] args) {
		Person p = new Person(); // obj1 created

		Person p1 = new Person(); // obj2 created
		p1.id = 101;
		p1.name = "Raju";

		Person p2 = new Person(); // obj3 created
		p2.id = 102;
		p2.name = "Rani";

		p.m1(p1, p2);

		Person[] arr = p.m2();

		for (Person person : arr) {
			System.out.println(person.id + "--" + person.name);
		}

	}
```

```
}
--------------------------------------------------------------------------------
--------------
```

Class : It is a plan or model. It contains variables & Methods

Object : Physical Entity. It is used to access variables & methods of the class

Variables : Used to store the data

- instance variables (inside class, outside method without static keyword)
  -> When obj created, memory will be allocated for instance variables
  -> Every obj will maintain its own copy of instance variables
- static variables (inside class, outside method with static keyword)
  -> When class is loaded then memory will be allocated for static variables
- local variables (inside the method)
  -> When method is invoked then only memory will be allocated for local variables
  -> Before using local variable, we have to initialize that

Methods : To perform some operation

- instance method ---> We will call by using object
- static method -----> We will call by using Classname
- Method Declaration (Signaure) ---> return type + name + list of parameters
- Method Definition (Body) ----> logic

==========
Constructors
==========

-> Constructor is a special method in java which is used to initialize current class instance variables

-> Constructor name should be same as class name

-> Constructor shoudn't have any return type (not even void)

Note: If we write return type for the constructor then it will become method

Syntax :

```
class Demo {

        Demo ( ) {
                // logic
        }
```

```
}
```

```
Demo d = new Demo ( );
```

Note: At the time of object creation our class Constructor will be executed. Constructor is mandatory to create the object.

-> Object creation means calling the constructor (new Demo( ))

Note: If we don't write the constructor in class, then java compiler will add one default constructor to our class.

-> We can check default constructor for the class using below command

```
> javap classname
```

Note: If we write constructor in the class, then compiler will not add any constructor.

-> Constructors are divided into 2 types

1) Zero Param Constructor / Default Constructor

- Constructor without parameters

```
class Student {

        Student ( ) {
                ...
        }
}
```

2) Parameterized Constructor

- Constructor which contains 1 or more paramters

```
class Student {

        Student (int i, int b ) {
                ...
        }
}
```

```
class Employee {

        public Employee(int i, int j) {
                System.out.println(i + j);
        }

        public static void main (String[] args) {
                Employee emp = new Employee(100, 200);
        }

}
```

==========
this keyword
==========

-> this is a predefined keyword in java

-> It is used to represent current class object

```
class Employee {

	String name;
	float salary;

	public Employee(String name, float salary) {
		this.name = name;
		this.salary = salary;
		System.out.println(this.name + "--" + this.salary);
	}

	public static void main(String[] args) {
		Employee emp = new Employee("Raju", 55000.00f);
	}

}
```

```
public class Student {

	int id;
	String name;
	int age;
	String gender;

	public Student(int id, String name, int age, String gender) {
		this.id = id;
		this.name = name;
		this.age = age;
		this.gender = gender;
		System.out.println(this.id + "-" + this.name + "-" + this.age + "-"
+ this.gender);
	}

	public static void main(String[] args) {
		Student s1 = new Student(1, "Raju", 20, "Male");
		Student s2 = new Student(1, "Rani", 22, "FeMale");
	}

}
```

==========
Constructor Overloading
==========

-> Writing more than one constructor with different parameters is called as Constructor Overloading

```
class Employee {

	Employee(int id){

	}

	Employee(double d){

	}

}
```

```
InputStreamReader isr = new InputStreamReader ( );

BufferedReader br = new BufferedReader( isr );

String s = new String("hi");

StringBuffer sb  = new StringBuffer(s);
```

==========
Access Specifiers / Modifiers
==========

-> These are used to specify accessibility for our classes, constructors, variables & methods

-> In java we have below 4 access specifiers / modifiers

1) public
2) private
3) protected
4) default

public : It is used to specify global access for classes, variables, methods and Constructors

-> We can take class as public
-> We can take constructor as public
-> We can take variables as public
-> We can take methods as public

Note: public means anybody can access from inside and outside the class also.

private: It is used to specify local access (with in the class). private variables, private methods,
private constructors can't be accessed outside of the class.

-> We can't use private for classes (not allowed)
-> We can take variables as private
-> We can take constructor as private (no body can create obj from outside of cls)
-> We can take method as private (no body can call our method from outside of cls)

Note: To make our java class as Singleton, we will use private constructor. The java class which is having
only one object is called as Singleton class.

protected : Protected members can be accessed in same package & its sub classes

default : default members can be accessed in same package. When we don't specify any modifier then it comes
under default modifier.

Note: only public and default modifiers are allowed for classes. private and protected are not allowed.

Note: If we use 'public' for the class name then class name & file name should be same. We should have only
one public class in the java file.

Class Name : EligServiceImpl

Method : public EligResponse executePlanCondtions(Long caseNum, String planName, Integer age)

```
EligServiceImpl    eligService = new EligServiceImpl ( );
```

EligResponse response = eligService.executePlanConditions (. . .);

=======
OOPS
=======

Encapsulation : Combining variables & methods as single unit. It is used for data hiding.

Ex: Java Class

Abstraction : Hiding un-necessary details and providing only useful information.

Ex: Abstract classes & Interfaces

Inheritence : The process of extending properties from one class to another class is called as inheritence.

-> It is used for code re-usability

Polymorphism : Exihibiting multiple behaviours based on the situation is called as Polymorphism.

Ex: Objects will exhibit multiple behaviours

=============
Encapsulation
=============

-> It is used for data hiding

-> We will combine variables & methods as one single unit using Class

-> Java class is one of the best example for Encapsulation

```
public class Account {

	private int accNum;
	private String name;

	public void setAccNum(int accNum) {
		this.accNum = accNum;
	}

	public int getAccNum() {
		return this.accNum;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getName() {
		return this.name;
	}

}


public class Test {

	public static void main(String[] args) {

		Account obj = new Account(); // obj creation
```

```
                obj.setAccNum(797979);
                obj.setName("Ashok");

                int accNum = obj.getAccNum();
                String name = obj.getName();

                System.out.println(accNum + "--" + name);
        }
}
```

==========
Inheritence
==========

-> The process of extending the properties from one class to another class is called as Inheritence

-> To extend the properties we will use 'extends' keyword

-> From which class we are extending the properties that class is called as 'Parent' or 'Super' or 'Base' class

-> The class which is extending the properties is called as 'Child class' or 'Sub class' or 'Derived' class

-> By using inheritence we can achieve code re-usability

```
class User {
        // properties
        // methods
}

class Student extends User {

}
```

Note: In above example 'User' class is acting as Parent class and Student class is acting as Child class.

```
// Inhertience w.r.t to variables

public class User {
        int id;
        String name;
}

public class Student extends User {

        int rank;

        public static void main(String[] args) {

                // child class object creation
                Student s = new Student();
```

```
                s.rank = 1;

                // accessing parent class properties using child cls obj
                s.id = 101;
                s.name = "Bhanu";

                System.out.println(s.id + "--" + s.name + "--" + s.rank);
        }
}

public class Student extends User {

        int rank;

        public static void main(String[] args) {

                // creating parent class obj
                User user = new User();
                user.id = 101;
                user.name = "Raj";

                user.rank = 1; // invalid bcz parent can't access child properties
        }
}

// inhertience w.r.t to methods

public class User {

        int id;
        String name;

        void m1() {
                System.out.println(" Parent class :: m1 ( ) method called");
        }
}

public class Employee extends User {

        void m2() {
                System.out.println("Child class - m2() method called");
        }

        public static void main(String[] args) {

                // creating object for child class
                Employee emp = new Employee();

                // calling parent class method
                emp.m1();

                // calling child class method
                emp.m2();
        }
}


// inhertience w.r.t to constructors

->  Whenever we create child class object, then first it will execute parent class
zero-param constructor and
 then it will execute child class constructor.

Q) Why Parent class constructor is executing first ?

-> Child should be able to access parent propertiece hence parent constructor will
execute first to initialize
 parent class properties.
```

```
public class User {

	int id;
	String name;

	public User() {
		System.out.println("Parent class :: 0-param constructor called ");
	}

}

public class Employee extends User {

	double salary;

	public Employee() {
		System.out.println("Child Class :: 0-Param Constructor called");
	}

	void m2() {
		System.out.println("Child class - m2() method called");
	}

	public static void main(String[] args) {

		// creating object for child class
		Employee emp = new Employee();

		// initializing parent class properties using child obj
		emp.id = 101;
		emp.name = "John";

		// initialing child class properties using its own obj
		emp.salary = 4500.00;

		System.out.println(emp.id + "--" + emp.name + "--" + emp.salary);
	}
}
```

==========
Types of Inheritence
==========

-> Inheritence is divided into multiple types

1) Single Level
2) Multi Level
3) Multiple --------> Not supported by Java due to Ambiguity problem
4) hierarchical

Single level : Class B extends Class A

Multi Level : Class C extends Class B extends class A extends Object

Multiple Inheritence : If one child having more than one parent (Java doesn't support to avoid ambiguity)

Hierarchical : If one parent having multiple childs

Note: For every java class, java.lang.Object class will act as Parent either directley or in-directley

-> If our class doesn't have any parent then java.lang.Object will become parent directley

-> If our class having parent then java.lang.Object will become parent in-directley

Note: Every java class can access methods available in java.lang.Object class.

```
class Parent {
	void m1() {
		System.out.println("Parent - Class - m1() Called");
	}

	void m2() {
		System.out.println("Parent - Class - m2() called");
	}
}

class Child extends Parent {

	public int hashCode() {
		return 101;
	}

	void m1() {
		System.out.println("Child - Class - m1() Called");
	}

	void m2() {
		System.out.println("Child - Class - m2() called");
		super.m2();
	}
}

public class Test {
	public static void main(String[] args) {
		Child c = new Child();
		c.m1();
		c.m2();
		int hashcode = c.hashCode();
		System.out.println("Hash Code :: " + hashcode);
	}
}
```

```
=====================================
Methods Execution Flow w.r.r to Inheritence
=====================================
```

=> When we call a method using Object, first it will check in current class for that method, if available it
 will call that method directley. If method not available in current class then it will check in parent class
(It can be direct or indirect parent). If parent having that method then it will call parent class method.
If parent class also doesn't have method then it will throw Exception.

Note: In inheritence always priority will be given for Child class / Sub class object. If child class doesn't
contain that method then priority will be given to Parent class method.

```
===============
Polymorphism
===============
```

Poly ----> Many

Phism ---> Forms

-> If any object is exhibiting multiple behaviours based on the Situation then it

is called as Polymorphism.

-> Polymorphism is divided into 2 types

1) Static polymorphism / Compile-time Polymorphism

Ex: Overloading

2) Dynamic polymorphism / Run-time Polymorphism

Ex: Overriding

==========
Method Overloading
==========

-> The process of writing more than one method with same name and different parameters is called as Method
Overloading.

```
public class Calculator {

	void add (int i, int j) {
		System.out.println("Sum from 1st method :" + (i + j));
	}

	void add (int i, int j, int k) {
		System.out.println("Sum from 2nd method : " + (i + j + k));
	}

	public static void main(String[] args) {
		Calculator c = new Calculator();

		c.add(10, 20);
		c.add(10, 20, 30);

		c.add(10, 20, 30, 40);   // invalid
	}
}
```

=> When methods are performing same operation then we should give same name hence it will improve code
 readability.

Ex:

```
substring (int start)

substring(int start, int end)
```

=> In Method Overloading scenario, compiler will decide method should be called. For example if we write
like below then program will fail it compilation stage.

==========
Method Overriding
==========

-> The process of writing same methods in Parent class & Child class is called as Method Overriding.

```
class Parent {

	void m1( ) {
		// logic
	}
```

```
}

class Child extends Parent {

        void m1 ( ){
                //logic
        }

        p s v  m (String ... args){
                Child c = new Child ( );
                c.m1 ( );
        }
}
```

Note: When we don't want to execute Parent method implementation, then we can write our own implementation
 in child class using method Overriding.

********** Write a program on Method Overriding **********

```
class RBIBank {

        boolean checkElgibility() {
                // docs verification logic
                return true;
        }

        double getHomeLoanRofi() {
                return 10.85;
        }
}

public class SBIBank extends RBIBank {

        // overriding parent method to give my own rofi
        double getHomeLoanRofi() {
                return 12.85;
        }

        public String applyHomeLoan() {
                boolean status = checkElgibility(); // parent method
                if (status) {
                        double homeLoanRofi = getHomeLoanRofi(); // child method
                        String msg = "Your loan approved with RI as ::" +
homeLoanRofi;
                        return msg;
                } else {
                        return "You are not elgible for home loan";
                }
        }

        public static void main(String[] args) {
                SBIBank bank = new SBIBank();
                String msg = bank.applyHomeLoan();
                System.out.println(msg);
        }
}


// Method Overriding Example with User-Defined Objects and String Objects

public class Demo {

        public static void main(String[] args) {

                SBIBank b1 = new SBIBank();
                SBIBank b2 = new SBIBank();
```

```
                boolean bankObjStatus = b1.equals(b2); // false
                System.out.println("Both Banks Are Equal ?? :: " + bankObjStatus);

                String s1 = new String("ashokit");
                String s2 = new String("ashokit");

                boolean stringObjStatus = s1.equals(s2); // true
                System.out.println("Both Strings Are Equal ?? :: " +
stringObjStatus);

        }
}
```

NOte: Object class equals ( ) method will compare address of the objects where String class equals ( ) method
will compare content of the objects.

Note: String class overriding equals ( ) method.

==========
Types of Relations in Java Classes
==========

-> In java classes we can use below 2 types of relations

1) IS-A relation -----> Inheritence

2) HAS-A relation -----> Composition

==========
IS-A relation Example
==========

-> If one class wants to re-use all the properties of another class then we will go for IS-A relation.

        Ex: Inhertience is the example for IS-A relation

```
class User {

        int id;
        String name;

    void speak ( ){
        System.out.println("Hi, My Id is : "+ id + ", My Name : "+ name);
    }
}

class Student extends User {   // IS-A relation

        public static void main(String... args){
                Student s = new Student ( );
                s.id = 10;
                s.name = "Raju";
                s.speak ( );
        }
}
```

==========
HAS-A Relation Example

======================

-> If one class wants to re-use some properties of another class then we will go for HAS-A relation.

Ex: Composition is the example for HAS-A relation

```
class Engine {

        int id;
        String name;
        String fuelType;

                void start ( ){
                        System.out.println("Engine Starting....");
                }
}


class Car {

        void drive ( ){
                Engine e = new Engine ( );   // HAS-A Relation
                e.start ( );
                System.out.println("Journey Started");
        }

        psvm(String[] args){
                Car c = new Car ( );
                c.drive ( );
        }
}
```

==========
final keyword
==========

-> final is a reserved keyword in java

-> We can use final keyword at 3 places

1) class level
2) variable level
3) method level

-> final classes can't be inherited. We can't extend properties from final classes. Final classes are immutable.

```
public class User extends String { // invalid because String is final class

}
```

-> final variables are nothing but constants. Final variable value can't be modified.

```
public final int pi = 3.14;

        pi = 4.32; // invalid
```

-> final methods can't overriden. We can't override final methods.

==========
Abstraction
==========

-> The process of hiding un-necessary data and providing only useful data is called as Abstraction.

-> We can achieve abstraction using Interfaces & Abstract classes.

==========
Method Types
==========

-> In java we can write 2 types of methods

1) Concrete Method
2) Abstract Method

-> The method which contains body is called as 'Concrete method'

```
public void m1 ( )
{

}
```

-> The method which doesn't contain body is called as 'Abstract method'

```
public abstract void m2 ( ) ;
```

Note: By using 'abstract' keyword we can create abstract methods & abstract classes.

==========
Interfaces
==========

-> Interfaces are used to achieve loosely coupling & abstraction

-> Interfaces contains only abstract methods (upto 1.7v of java)

-> To create a interface we will use 'interface' keyword

-> Interface doesn't contain 'constructor'

-> We can't create Object for the interface

-> Once interface is created then anybody can provide implementation for the interface

-> Implementing interface means overriding 'interface abstract methods'

-> When we are implementing a interface then it is mandatory to implement all abstract methods of that interface.

-> To implement a interface we will use 'implements' keyword

Note: One java class can implement Multiple Interfaces at a time.

Note: One java class can extend properties from only one class and it can implement Multiple interfaces at a time.

==========
Interface Example
==========

```
public interface Bank {

        public void moneyTransfer();

        public void checkBalance();
}
```

```
public class HdfcBank implements Bank {

	public void moneyTransfer() {
		System.out.println("Money Transfer from HDFC....");
	}

	public void checkBalance() {
		System.out.println("Checking Balance from HDFC.....");
	}
}

public class AxisBank implements Bank {

	public void moneyTransfer() {
		System.out.println("Money Transfer from Axis ....");
	}

	public void checkBalance() {
		System.out.println("Check Balance from Axis....");
	}
}


public class BankDemo {

	public static void main(String[] args) {
		Bank b; // reference variable

		b = new AxisBank(); // storing impl obj into ref variable

		b.moneyTransfer(); // axis-bank method will be called
		b.checkBalance(); // axis-bank method will be called

		b = new HdfcBank();  // storing impl obj into ref variable

		b.moneyTransfer(); // hdfc-bank method will be called
		b.checkBalance(); // hdfc-bank method will be called
	}
}
```

=> We can't create Object for interface

=> Interface reference variable can hold its implementation class object.

Ex:

```
Bank b = new AxisBank ( ) ;  // valid

Bank b = new HdfcBank ( ) ;  // valid

Bank b = new Kotak ( ) ; // in-valid because it is not implementation class for Bank
```

=> When method is taking interface a parameter that means that method is expecting interface implementation
class object as parameter.

```
public void m1 (Bank b ) {  // loosely coupled bcz we can pass any impl cls obj

}

public void m1(AxisBank b){ // tightly coupled bcz we have to pass only AxisBank obj

}
```

=> When method is having Interface as a return type that means that method will return interface implementation
object as return type

```
                                public Bank  m2 ( ) { // loosely coupled bcz we can return any impl cls obj

                                }

                                public HdfcBank  m2 ( ) {  // tighthly coupled bcz only HdfcBank obj shud be returned

                                }
```

-> If interface doesn't contain any method then that interface is called as 'Marker Interface'

Ex: Cloneable, Serializable etc.....

-> If our class implements pre-defined marker interface then JVM will treat our classes as special classes and JVM
will provide special functionality based on the marker interface we have implemented.

```
// user defined marker interface
public interface Demo {

}

public class Test implements Demo {

}
```

Note: We can create our own marker interfaces but there is no use because JVM don't know abt our marker interfaces.

Note: In java 1.8, they introduced Functional Interfaces.

-> The interface which contains only one abstract method is called as Functional Interface.
-> Functional Interfaces are introduced to called 'Lambda Expressions'.

-> One interface can't implement another interface.

-> One Interface can extend another interface.

-> In Interface we can declare variables also, by default they are public static final.

```
public interface Bank  {
        public void checkBalance ( ) ;
}

public interface RbiBank implements Bank {   // invalid

}

public interface RbiBank extends Bank {  // valid
                public void applyLoan ( ) ;
}
```

==========
Abstract Classes
==========

-> The class which contains both concrete and abstract methods is called as abstract class.

-> We will use 'abstract' keyword to represent class as abstract class

-> To write abstract method 'abstract' keyword is mandatory

-> We can write constructor in abstract class but we can't create Object

-> Abstract classes can have child classes

-> When we extend properties from abstract class then it is mandatory to override all abstract methods of that class

-> Abstract class constructor will be executed when we create object for child class.

```
abstract class DieselMachine {

	public DieselMachine() {
		System.out.println("DieselMachine-Constructor");
	}

	public void start() {
		System.out.println("Machine starting....");
	}

	public abstract void fillFuel();
}

public class Machine extends DieselMachine {

	public Machine() {
		System.out.println("Machine Constructor");
	}

	@Override
	public void fillFuel() {
		System.out.println("filling fuel tank....");
	}

	public static void main(String[] args) {
		Machine m = new Machine();
		m.fillFuel();
		m.start();
	}
}
```

=> We can't use 'abstract' and 'final' combination because it is illegal.

```
		public abstract final void m1 ( ) ;  // invalid
```

=> abstract means override in child where as 'final' means don't override .

=> To create interface we will use 'interface' keyword
=> To create abstract classes we will use 'abstract' keyword

=> Interface contains abstract methods

=> Abstract classes can contain both abstract methods & concrete methods

=> Interface can't have constructor
=> Abstract class can have constructor

=> We can't create obj for interface
=> We can't create obj for abstract class

=> One interface can't implement another interface
=> One interface can extend another interface

=> When we implement any interface then we have to implement all the abstract methods of that interface
=> When we extend abstract class, we need override all abstract methods

=> Abstract class constructor will be executed when we create object for sub class.

=> If interface contains only one method then it is called Functional Interface
=> If interface doesn't have any method then it is called as Marker interface

=> When we don't know implementation for methods then we will go for Interfaces

=> When we know partial implementation for methods then we will go for abstract classes

==========
Blocks in java
==========

-> Block means some part or some piece of information or some piece of code

-> In java program we can write 2 types of blocks

1) instance block
2) static block

==========
Instance Block
==========

-> If you want to execute some piece of code when object is created then we can go for instance block
-> Instance block will be executed before constructor execution

syntax:

```
{
        // stmts
}
```

==========
static Block
==========

-> If you want to execute some piece of code when class is loaded into JVM then we can go for static block
-> static block will execute before main ( ) method execution

syntax:

```
static
{
        // stmts
}
```

Q) What is static control flow & instance control flow in java program ?

==========
Static Control Flow
==========

-> When class is loaded into JVM then static control flow will start

-> When we run java program, JVM will check for below static members & JVM will allocate memory for them in below order

a) static variables
b) static methods
c) static blocks

-> Once memory allocation completed for static members then it will start execution in below order

a) static block
b) static method (if we call) -- only main method will execute automatically by jvm
c) static variable

-> static variables can be accessed directley in static blocks and static methods.

Note: If we want to access any instance method or instance variable in static area then we should create object and
using that object only we can access. We can't access directley without object.

==========
Instance Control Flow
==========

-> instance means Object

-> Instance control flow will begin when object is created for a class

-> When Object is created then memory will be allocated for

a) instance variables
b) instance methods
c) instance blocks

-> Once memory allocation completed then execution will happen in below order

a) instance block
b) constructor
c) instance methods (if we call)

Note: static members can be access directley in instance areas becase for static memebers memory aready allocated
at the time of class loading.

```
public class Demo {

	{
		System.out.println("i am from instance block");
	}

	public Demo() {
		System.out.println("I am from constructor");
	}

	static {
		System.out.println("I am from static block");
	}

	public static void main(String[] args) {
		System.out.println("i am from main method...");
		Demo d = new Demo();
```

```
        }
}
```

==========
java.lang.Object class
==========

-> Object is a pre-defined class available in java.lang package

-> Object class will act as super class for all the classes in java either directley or in-directley

Note: If our class doesn't have any super class then Object class will become direct Super class. If our class
 having any super class then Object class will become in-direct super class.

-> Object class having 11 methods, those 11 methods are by default available for every java object.

1) protected Object clone ( )

2) boolean equals (Object obj)

3) protected void finalize( )

4) Class <?> getClass( )

5) int hashCode( )

6) String toString( )

7) notify ( )

8) notifyAll ( )

9) void wait ( )

10) void wait(long timeout)

11) void wait (long timeout, int nanos)

Note: Last 5 methods will be used in Multi-Threading concept.

==========
public String toString ( ) method :
==========

-> It is used to represent Object in String format

-> When we print any object or when we call toString ( ) method by default it will call Object class toString ( ) method

---
Object class toString( ) method implementation

```
public String toString ( ) {

        return this.getClass( ).getName ( ) + "@"+
Integer.toHexString(this.hashCode());
}
```

Output : Student@15db9742

-> If we don't like this implementation, then we can override toString ( ) method in our class like below

```
public class Student {

	int id;
	String name;

	public static void main(String[] args) {

		Student s = new Student();
		s.id = 101;
		s.name = "John";

		System.out.println(s); // toString ( ) will be called
		System.out.println(s.toString());

		String s1 = new String("hi");
		System.out.println(s1);
	}

	public String toString() {
		return id + "--" + name;
	}

}
```

-> We will override toString ( ) method to print content of object.

Note: String class is already overriding toString ( ) method.

==========
int hashCode ( ) method
==========

-> When we create object for a class then JVM will assign one unique hashcode for every Object

-> Using hashCode ( ) method we can get hashcode of the object

-> If we don't want to execute Object class hashCode ( ) method then we can override hashCode ( ) method in our class.

```
public class Student {

	int id;
	String name;

	public static void main(String[] args) {

		Student s = new Student();
		s.id = 101;
		s.name = "John";

		System.out.println(s); // toString ( ) will be called
		System.out.println(s.hashCode());
	}

	public String toString() {
		return id + "--" + name;
	}

	public int hashCode() {
		return id;
	}

}
```

==========
boolean equals (Object obj) method
==========

-> equals ( ) method is used to compare one object with another object and returns boolean value. If objects
 are same then it will return true otherwise it will return false value.

Note: Object class equals( ) method will compare address of the object not content.

```
public class Student {

	int id;
	String name;

	public Student(int id, String name) {
		this.id = id;
		this.name = name;
	}

	public static void main(String[] args) {

		Student s1 = new Student(101, "John");
		Student s2 = new Student(101, "John");

		System.out.println(s1.equals(s2)); // false - compares address
		System.out.println(s1 == s2); // false - compares address

		String s3 = new String("hi");
		String s4 = new String("hi");

		System.out.println(s3.equals(s4)); // true - compares content of
objects

	}
}
```

Note: In String class equals ( ) method overriden to compare content of objects thats why in above program for
 Strings we are getting 'true' as ouput. For Student class it is checking address hence we are getting 'false' as output.

-> If we want to compare content of our Student objects then we need to override like below

```
public class Student {

	int id;
	String name;

	public Student(int id, String name) {
		this.id = id;
		this.name = name;
	}

	public static void main(String[] args) {

		Student s1 = new Student(101, "John");
		Student s2 = new Student(101, "John");

		System.out.println(s1.equals(s2)); // true - compares content
(overriden)
		System.out.println(s1 == s2); // false - compares address

		String s3 = new String("hi");
		String s4 = new String("hi");
```

```
                System.out.println(s3.equals(s4)); // true - compares content of
objects

        }

        @Override
        public int hashCode() {
                final int prime = 31;
                int result = 1;
                result = prime * result + id;
                result = prime * result + ((name == null) ? 0 : name.hashCode());
                return result;
        }

        @Override
        public boolean equals(Object obj) {
                if (this == obj) // s1 == s2
                        return true;
                if (obj == null) // s2 == null
                        return false;
                if (getClass() != obj.getClass())
                        return false;
                Student other = (Student) obj; // typecasting
                if (id != other.id) // s1.id != s2.id
                        return false;
                if (name == null) { // s1.name == null
                        if (other.name != null)
                                return false;
                } else if (!name.equals(other.name)) // !s1.name.equals(s2.name)
                        return false;
                return true; // ---> true
        }
}


=======================
Class getClass ( ) method
=======================

-> This method is used to get Runtime instance of class

public class Student {

        public static void main(String[] args) throws Exception {

                Student s = new Student(); // obj creation
                Class clz = s.getClass();
                System.out.println(clz.getName());
                System.out.println(s.getClass().getName()); // method chaining to
get cls name

                Object object = clz.newInstance(); // 2nd approach to create object
for a cls
                System.out.println(object);

        }
}

===============
clone ( ) method
===============

-> This method is used to create duplicate object for the given object

-> If we want to clone any object then that class should implement Cloneable
interface which is marker interface.

-> If your class implements Cloneable interface then only JVM will allow you to
```

clone the object of that class.

```
public class Student implements Cloneable {

        public static void main(String[] args) throws Exception {

                Student s = new Student();
                System.out.println(s);

                Object clone = s.clone(); // cloning (another approch to create obj
for a class)
                System.out.println(clone);

        }
}
```

Q) In how many ways we can create Object for a class ?

Ans)

1) using new operator

2) using newInstance ( ) method

3) using clone ( ) method

==========
finalize ( ) method
==========

-> When garbage collector removing any object from JVM then it will call finalize ( ) method

Note: Garbage Collector is used to remove un-used objects / un-referenced objects from JVM heap area.

finalize ( )
clone ( )
equals ( )
hashCode ( )
toString ( )
getClass ( )

notify ( )
notifyAll ( )
wait ( ) - 3 overloaded methods

class --> This is keyword to create class in Java

Class --> This is predefined class available in java.lang package

Note: newInstance ( ) method available in Class it is used to create obj for a class

Object ---> This is also predefined class available in java.lang package. Default parent for all java classes.
 It contains 11 methods. Every java class can access Object class methods by default.

==========
OOPS Summary
==========

1) Procedural Oriented Language (PoP)
2) Object Oriented Programming Language (OOP)
3) Classes
4) Objects
5) Variables (Instance, static & local)
6) Methods (Parameters & Return Type)
7) Instance methods & static methods
8) Constructors (default, 0-param constructor & parameterized constructor)
9) Constructor Overloading
10) this keyword
11) Access Modifiers (public, private, protected & default)
12) Encapsulation
13) setter methods & getter methods
14) Inheritence
15) Inheritence Types
16) Method Execution Order w.r.t Inheritence
17) Polymorphism
18) Method Overloading
19) Method Overriding
20) Concrete methods & abstract methods
21) Abstraction
22) Interfaces
23) Marker Interface
24) Abstract classes
25) Blocks
26) Static Control Flow
27) Instance Control Flow
28) Object class
29) Object class methods (11 methods)
30) final keyword

=========
Packages
=========

-> Packages are used to group the classes , interfaces, exceptions and errors

-> In java language, so many classes, interfaces, Exceptions and Errors are already available.

Ex: String, StringBuffer, StringBuilder, Arrays, BufferedReader, Scanner etc....

-> Sun people divided predefined classes, interfaces, Exceptions into several packages

1) java.lang

2) java.io

3) java.util

4) java.sql

-> java.lang package is default package and it is available for all java classes by default.

-> If we want to use any predefined class which is not part of java.lang package then we have to import that
class using 'import' keyword.

// Example Of predefined package import

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
```

```
public class Demo {

	public static void main(String[] args) {
		InputStreamReader isr = new InputStreamReader(System.in);
		BufferedReader br = new BufferedReader(isr);
	}
}
```

==========
User Defined Packages
==========

-> In our project we will create our own packages to organize project related classes & interfaces

-> To create a package we will use 'package' keyword

-> In java class, package statement should be in first line and class should have only one package statement.

-> We can create user-defined package like below

ex:

```
package  ashokit;

package com.tcs.aadhar;
```

-> Sample package naming convention : company-name.project-name.module-name

```
com.ibm.irctc.admin
		AdminLogin.java
		AdminService.java
		Request.java

com.ibm.irctc.user
		UserLogin.java
		UserService.java

com.ibm.irctc.reports
		ReportService.java
		Request.java
```

Note: In project we can create 2 classes with same in using 2 different packages.

```
package com.oracle;

public class Engine {

	public void start() {
		System.out.println("Engine starting...");
	}

}
```

```
package com.ibm;

import java.util.Scanner;
import com.oracle.Engine;

public class Car {

	public void drive() {

		Engine eng = new Engine();
```

```
                eng.start();

                Scanner s = new Scanner(System.in);

        }
}
```

Note:

1) In class, package statement should be in first line (top)

2) After package statement we can write multiple import statements

3) import java.io.* (it means all classes, interface, exceptions of io package will be imported) - not recommended

4) If 2 classes are in same package then import not required.

==========
Exception Handling
==========

1) What is Exception
2) Why to handle exception
3) Types of Exceptions
4) Exception Vs Error
5) Exceptions Hierarchy
6) Checked Exceptions
7) Un-Checked Exception
8) Exception Propogation
9) Exception Handling Keywords
    - 8.1 try
    - 8.2 catch
    - 8.3 finally
    - 8.4 throw
    - 8.5 throws
10) Try with Resources (java 1.7v feature)
11) User Defined Exceptions

Succesful / Graceful Termination : Termination after executing program successfully

Abnormal Termination : Termination in middle of program execution

==========
What is Exception ?
==========

-> Un-expected and Un-wanted situation in the program execution is called as Exception.

-> Exception will distrub normal flow of the program execution

-> When Exception occured program will be terminated abnormally

Note: As a programmer we are responsible for programs graceful termination.

-> To Achive graceful termination we need to handle the execptions occured while program executing.

-> The process of handling Exceptions is called as Exception Handling.

-> The main aim of Exception Handling to achieve graceful termination of the program

-> In java we have so many predefined exceptions

Ex:

ArithematicException
NullPointerException
FileNotFoundException
SQLException

==========
Exception Hierarchy
==========

-> In this hierarchy Throwble is the root class

1. Throwble

1.1 Exception

1.1.1 Checked Exception

1.1.2 Un-Checked Exception

1.2 Error

Q) What is the difference between Exception and Error ?

-> Exceptions can be handled where as Errors can't be handled.

==========
Exception Types
==========

-> Exceptions are divided into 2 types

1) Checked Exceptions :: Will be identified at compile time (occurs at run time)

Ex: IOException, FileNotFoundException, SQLException etc....

2) Un-Checked Exceptions : Will occur at Run time (Compiler can't identify these exception)

Ex: NullPointerException, ArithematicException etc...

==========
Exception Handling
==========

-> Java provided 5 keywords to handle exceptions

1) try
2) catch
3) finally
4) throws
5) throw

========
try block
========

-> It is used to keep risky code

syntax:

```
try {
```

```
        // stmts
}
```

Note: We can't write only try block. try block required catch or finally (it can have both also)

try with catch ------> valid combination

try with multiple catch blocks ----> valid combination

try with finally -----> valid combination

try with catch & finally ---> valid combination

only try block ----> invalid

only catch block ---> invalid

only finally block ---> invalid

======
catch
======

-> catch block is used to catch the exception which occured in try block

-> To write catch block , try block is mandatory

-> One try block can contain multiple catch blocks also

syntax:

```
try {

        // logic

} catch ( Exception e ){

        // logic to catch exception info

}
```

Note: If exception occured in try block then only catch block will execute otherwise catch block will not execute

```
public class Demo {

        public static void main(String[] args) {
                System.out.println("main( ) method started...");
                try {
                        System.out.println("try block start");
                        String s = null;
                        s.length(); // NPE
                        System.out.println("try block end");
                } catch (Exception e) {
                        System.out.println("in catch block");
                        e.printStackTrace();
                }
                System.out.println("main( ) method ended...");
        }
}
```

// in below scenario catch block will not be executed bcz there is no exception in try block

```
public class Demo {
```

```
	public static void main(String[] args) {
		System.out.println("main( ) method started...");
		try {
			System.out.println("try block start");
			String s = "hi";
			int i = s.length();
			System.out.println("try block end");
		} catch (Exception e) {
			System.out.println("in catch block");
			e.printStackTrace();
		}
		System.out.println("main( ) method ended...");
	}
}

// we can write one try block with multiple catch blocks also like below
public class Demo {

	public static void main(String[] args) {
		System.out.println("main( ) method started...");
		try {
			System.out.println("try block start");
			String s = "hi";
			int i = s.length();
			System.out.println("try block end");
		} catch (ArithmeticException e) {
			System.out.println("in catch block");
			e.printStackTrace();
		} catch (NullPointerException e) {
			e.printStackTrace();
		} catch (Exception e) {
			e.printStackTrace();
		}
		System.out.println("main( ) method ended...");
	}
}

Note: Catch blocks order should be child to parent

// below program will fail at compile time because of un-reachable second catch
block
public class Demo {

	public static void main(String[] args) {
		System.out.println("main( ) method started...");
		try {
			System.out.println("try block start");
			String s = "hi";
			int i = s.length();
			System.out.println("try block end");
		} catch (Exception e) {
			e.printStackTrace();
			System.out.println("first catch");
		} catch (NullPointerException ne) {
			System.out.println("second catch");
		}
		System.out.println("main( ) method ended...");
	}
}

==============
finally block
==============

-> It is used to perform resource clean up activities

		Ex: file close, db connection close etc....
```

-> finally block will execute always (irrespective of the exception)

try with finally : valid combination

try with catch and finally : valid combination

catch with finally : invalid combination

only finally : invalid combination

```
// java program with try-catch-finally scenario
public class Demo {

	public static void main(String[] args) {
		System.out.println("main( ) method started...");
		try {
			System.out.println("try block - start");
			int i = 10 / 2;
			System.out.println("try block - end");
		} catch (Exception e) {
			System.out.println("catch block");
			e.printStackTrace();
		} finally {
			System.out.println("finally - block");
		}
		System.out.println("main( ) method ended...");
	}
}
```

```
// java program with try-finally scenario

public class Demo {

	public static void main(String[] args) {
		System.out.println("main( ) method started...");
		try {
			System.out.println("try block - start");
			int i = 10 / 0;
			System.out.println("try block - end");
		} finally {
			System.out.println("finally - block");
		}
		System.out.println("main( ) method ended...");
	}
}
```

Q) What is the differnece between   final, finalize( ) and finally ?

final : it is a keyword which is used to declare final variables, final methods and final classes

finalize ( ) : It is predefined method available in Object class, and it will be called by garbage collector
 before removing unused objects from heap area.

finally : it is a block we will use to execute some clean acitivites in exception handling

try : it is used to keep our risky code

catch : It is used to catch the exception occured try block

finally : to execute clean up activities

throws : It is used to hand over checked exceptions to caller method / jvm

Note: We can ignore checked exceptions using throws keyword

```
public class Demo {
	public static void main(String[] args) throws FileNotFoundException {
		FileReader fr = new FileReader("abc.txt");
	}
}
```

Note: We can ignore all checked exceptions like below

```
public class Demo {
	public static void main(String[] args) throws Exception {
		FileReader fr = new FileReader("abc.txt");
		Class.forName("");
	}
}
```

=========
throw :
========

-> throw keyword is used to create the exception

syntax:

```
			throw new Exception("Msg");
```

```
package in.ashokit;

public class Demo {

	public String getName(int id) throws Exception {
		if (id == 100) {
			return "raju";
		} else if (id == 101) {
			return "rani";
		} else {
			throw new Exception("Invalid Id");
		}
	}

	public static void main(String[] args) throws Exception {
		Demo d = new Demo();
		String name = d.getName(101);
		System.out.println(name);

		String name1 = d.getName(200);
		System.out.println(name1);
	}
}
```

// Stack Over Flow Error

```
public class Demo {

	void m1() {
		m2();
	}

	void m2() {
		m1();
	}

	public static void main(String[] args) {
		System.out.println("main ( ) method - start");
		Demo d = new Demo();
		d.m1();
		System.out.println("main( ) method - end");
	}
}
```

==========
User Defined Exceptions
==========

-> In java language we have several pre-defined exception classes

Ex:

IOException
FileNotFoundException
ClassNotFoundException
SQLException
AirthematicException
ArrayNegativeSizeException
NullPointerException
ClassCastException etc...

-> Based on Project requirement, sometimes we need to create our own exceptions those are called as
User Defined Exceptions

Ex:

InvalidCredentialsExceptions
NoRecordsFoundException
NoDataFoundException
InvalidInputException

-> To create our own Exception we need to extend the properties from Exception or RuntimeException class

```
public class NoDataFoundException extends RuntimeException {

	public NoDataFoundException() {

	}

	public NoDataFoundException(String msg) {
		super(msg);
	}

}
```

```
package in.ashokit;

public class Demo {

	private String getName(int id) {
```

```
                if (id == 100) {
                        return "Raju";
                } else if (id == 101) {
                        return "Rani";
                } else {
                        throw new NoDataFoundException("Invalid Id");
                }
        }

        public static void main(String[] args) {
                Demo d = new Demo();
                d.getName(200);
        }
}
```

throws : It is used to ignore checked exceptions

throw : It is used to create the exception

```
throw new Exception (" ");
```

```
public class InvalidIdException extends RuntimeException {

                public InvalidIdException(String msg){
                                super (msg);
                }
}
```

```
int  id = scanner.nextInt ( );

if( id <= 0 ){
        throw new InvalidIdException("invalid id");
}
```

==========
Method Stack
==========

```
void m1 ( int a, int b ) {
        int c = a / b;
        Sysout( c);
}

public static void main (String... args){
                Demo d = new Demo ( );
                d.m1 ( );
}
```

```
public class Demo {

        void m1(int a, int b) {
                System.out.println("m1() - started");
                try {
                        int c = a / b;
                        System.out.println(c);
                } catch (Exception e) {

                }
                System.out.println("m1() - ended");
        }

        public static void main(String[] args) throws Exception {
```

```
                System.out.println("main() - method started");
                Demo d = new Demo();
                d.m1(10, 0);
                System.out.println("main() - method ended");

        }
}
```

```
package in.ashokit;

public class InvalidNumberException extends Exception {

        public InvalidNumberException(String msg) {
                super(msg);
        }
}
package in.ashokit;

public class Demo {

        void m2(int a, int b) throws Exception {
                System.out.println("m2() - started");
                try {
                        int c = a / b;
                        System.out.println(c);
                } catch (Exception e) {
                        throw new InvalidNumberException("invalid number");
                }
                System.out.println("m2() - ended");
        }

        void m1(int a, int b) throws Exception {
                System.out.println("m1() - started");
                m2(a, b);
                System.out.println("m1() - ended");
        }

        public static void main(String[] args) {
                System.out.println("main() - method started");
                try {
                        Demo d = new Demo();
                        d.m1(10, 0);
                } catch (Exception e) {
                        e.printStackTrace();
                }
                System.out.println("main() - method ended");
        }
}
```

==========
static import
==========

-> Static import is used to access static members directley without using class name.

```
package com.ibm.aadhar;

public class Person {
```

```
	public static void speak() {
		System.out.println("Hi, i am ashok");
	}

	public static void m1() {
		System.out.println("hi, i am from m1()");
	}

	public static void m2() {
		System.out.println("hi, i am from m2()");
	}

}

/// Java Program with Normal Import to use Person class

package in.ashokit;

import com.ibm.aadhar.Person;

public class Demo {

	public static void main(String[] args) {

		Person.speak();
		Person.m1();
		Person.m2();

	}
}

/// java program with static import

package in.ashokit;

import static com.ibm.aadhar.Person.*;

public class Demo {

	public static void main(String[] args) {

		speak();
		m1();
		m2();

	}
}
```

==========
variable arguments   (var args)
==========

-> var-args concept introduced in java 1.5 v

-> When we don't know how many parameters to take for a method then we can use var-args

```
public int add (int...  x) {

}
```

1) Only 3 ellipses (...) allowed to declare variable argument
2) Variable argument should be the last parameter of the method
3) A method should contain only one variable argument

```
package in.ashokit;
```

```
public class Calculator {

	public void add(int... a) {
		int sum = 0;
		for (int x : a) {
			sum = sum + x;
		}
		System.out.println(sum);
	}

	public static void main(String[] args) {
		Calculator c = new Calculator();
		c.add(10, 20);
		c.add(10, 20, 30);
		c.add(10, 20, 30, 40);
		c.add(1, 2, 3, 4, 5);
	}
}
```

```
public   void  m1 (String s, int... i) ;   // valid

public void  m1 (String s, boolean... b) ; // valid

public void  m1 (int i, String... s) ; // valid

public void  m1( int... i ) ; // valid

public void m1 ( double.... d) ; // invalid bcz 4 dots

public void m1 (int... i , int j ) ; // invalid bcz var-arg shud be last

public void m1 ( int i, int... j ) ; // valid
```

```
===============
Wrapper Classes
===============
-> Java is a Object Oriented Programming Language

-> In java we can represent everything in the form of Object

-> To represent primitive types data in object format java provided Wrapper classes

-> For every primitive type corresponding Wrapper class is available

-> All wrapper classes are part of java.lang package

byte  ----> Byte

short  ---> Short

int  --->   Integer

long  ---> Long

float   ----> Float

double   ---->Double

char    ---> Character
```

boolean   ---> Boolean

==========
What is Boxing & Un-Boxing?
==========

-> The process of converting primitive data type into Wrapper Object is called as Boxing.

-> The process of converting Wrapper object into primitive type is called as Un-Boxing.

-> From Java 1.5, boxing and unboxing process automated hence they are called as Auto Boxing & Auto UnBoxing.

```
public class Calculator {

        public static void main(String... args) {

                byte b = 20;
                System.out.println(b);

                Byte b1 = new Byte(b);  // Auto-boxing
                System.out.println(b1);

                byte b2 = b1; // Auto-un-boxing
                System.out.println(b2);
        }
}
```

```
Integer   i1 = new Integer ( 10 ) ;
Integer   i2 = new Integer ( "20" );

Double  d1  = new Double ( 10.05 );
Double  d2 =  new Double ( "20.10" );

Character c1 = new Character ('a');
Character c1 = new Character ("a");   // invalid
```

==========
Type Casting
==========

-> Converting data from one data type to another data type is called as Type casting.

-> Type Casting is divided into 2 types

1) Widening / Up Casting

2) Narrowing / Down Casting

-> The process of converting data from lower data type to higher data type is called as Widening.
-> As we are converting lower data type to higher data type there is no data loss
-> For widenining no need to speciy type casting explicitley (JVM will take care of that)

```
byte  b = 20 ;
```

```
                                int i = b ; // widening
```

-> The process of converting data from higher data type to lower data type is called as Narrowing
-> As we are converting higher data type to lower data type there is a chance of data loss
-> For narrowing we need to specify type casting manually otherwise program can't be compiled

```
                                long  num = 100 ;

                                int  i = num ;  // compile time error

                                int x = ( int ) num ; // narrowing
```

==========
Working with String type
==========

-> String type data can be converted to Integer using Integer.parseInt (xx) method.

-> parseInt (xx) is a static method available in Integer wrapper class.

```
public class Calculator {

        public static void main(String[ ] args) {

                String s1 = "10" ;
                String s2 = "20";

                int i = Integer.parseInt ( s1 ) ;
                int j = Integer.parseInt ( s2 );


                System.out.println ( i + j );   // 30

        }
}
```

```
String s1 = "10.56" ;

double d = Double.parseDouble (s1);
```

```
String s1 = "hi" ;
int i = Integer.parseInt ( s1 ) ;  // NumberFormatException
```

```
// converting int data to String data

public class Calculator {

        public static void main(String[] args) {

                int i = 10;
                int j = 20;

                String s1 = String.valueOf(i);
                String s2 = String.valueOf(j);

                System.out.println(s1 + s2); // 1020
```

```
	}
}

-----------------------------------------------------------------------
Type casting w.r.t to Reference Types
-----------------------------------------------------------------------

public class Demo implements Cloneable {

	public static void main(String[] args) throws Exception {

		// child object
		Demo d = new Demo();

		// storing child obj into parent class reference variable
		Object obj = d; // widening / up casting   ( low to high )

		// cloning - getting parent object
		Object object = d.clone();

		// Storing parent object into child class reference variable
		Demo d1 = (Demo) object; // narrowing  ( high to low )

	}
}

-------------------------------------------------------------------------------------
---------------------------------

1) Java Introduction
2) Data Types
3) Variables
4) Operators
5) Control Statements
6) Arrays
7) String, String Buffer, StringBuilder
8) Command Line Arguments
9) Classes
10) Objects
11) Variables (non-static, static & local)
12) Methods ( method params & method return types)
13) Constructors
14) Blocks ( IB and SB)
15) Encapsulation
16) Inheritence
17) Polymorphism
18) Method Overloading
19) Method Overriding
20) Abstraction
21) Interfaces
22) Abstract Classes
23) Marker Interfaces
24) Variable Arguments
25) static keyword
26) this keyword
27) super keyword
28) final keyword
29) java.lang.Object class & method
30) Wrapper Classes
31) Type Casting
32) Packages
33) static import
34) Exception Handling
35) try-catch-finally- throws-throw
36) Exceptions Hierarchy
37) Checked Exceptions
38) Un-Checked Exceptions
```

1) File Handling
2) Multi-Threading
3) Collections
4) Generics
5) Inner Classes
6) Reflection API
7) Garbage Collection
8) Java 1.8 features

==========
File Handling
==========

-> File Handling is Very important area in every programming language

-> Below are the common file operations in the project

1) Create a file
2) Write the data to file
3) Read the data from file
4) Delete the file

-> To perform File operations java language provided one predefined class java.io.File

-> java.io package contains set of classes & interfaces to perform input and output operations

```
File  f = new File (String name);

File f1 = new File (File parent, String child ) ;
```

-> We have several methods in the file class

boolean createNewFile ( ) : It is used to create a new empty file

boolean mkdir ( ) : It is used to create new empty directory

String[ ] list ( ) : It is used to read the content of the given path

boolean delete ( ) : It is used to delete the file / directory based on given name

isFile ( ) : To check weather it is a file or not

isDirectory ( ) : To check weather it is a directory or not

```
import java.io.File;
import java.io.IOException;

public class Demo {

        public static void main(String... args) throws IOException {

                        File  f = new File("ashokit.txt");
                        boolean fstatus = f.createNewFile ( );
                        System.out.println(fstatus);

                        File f1 = new File("java.txt");
```

```
                        boolean f1status = f1.createNewFile ( );
                        System.out.println(f1status);

                        File f2 = new File("mywork");
                        boolean f2status = f2.mkdir( );
                        System.out.println(f2status);

                        File f3 = new File("data");
                        f3.mkdir ( );

                        File f4 = new File(f3, "test.txt");
                        f4.createNewFile ( );
        }

}
```

```
// Java program to display all the files and directories in given path


import java.io.File;
import java.io.IOException;

public class Demo {

        public static void main(String... args) throws IOException {

                        File f = new File ("C:\\Users\\ashok\\classes\\19-JRTP");

                        String [ ]  arr = f.list ( );

                        for ( String name : arr) {
                                System.out.println (name);
                        }
        }

}
```

```
Assignment : Write a java program to display content of given directory.
For filename it should display prefix as File :
For directory name it should display prefix as Directory :

Ex:

File : abx.txt
File: demo.txt
Directory: one
Directory: two
```

```
package in.ashokit;

import java.io.File;
import java.io.IOException;

public class Demo {

        public static void main(String... args) throws IOException {

                File f = new File("C:\\Users\\ashok\\classes\\19-JRTP");

                String[] arr = f.list();

                for (String name : arr) {
```

```
                        File f1 = new File(f, name);

                        if (f1.isFile()) {
                                System.out.println("File :: " + name);
                        }

                        if (f1.isDirectory()) {
                                System.out.println("Directory  :: " + name);
                        }
                }
        }
}
```

==========
IO Streams
==========

-> To perform operations on the file we need to use I/O Streams.

-> Using I/O streams we can establish link between java program and file (physical file)

```
                                                        write
                                java program <--------------------> file

                                                                  read
                                java program <----------------------> file
```

-> IO streams are divided into 2 types

1) Byte Stream : To read/write binary data (images, audios, videos, pdfs etc...)

2) Character Stream : To read/write character data (text files)

-> Byte Stream providing 2 types of classes

1) Input Stream Related Classes (Ex: FileInputStream)

2) Ouput Stream realted classes (Ex: FileOuputStream)

-> Character Stream providing 2 types of classes

1) Reader classes (Ex: FileReader)
2) Writer classes (Ex: FileWriter)

```
// Java Program to write the data to a file using FileWriter class
import java.io.*;

public class Demo {

        public static void main(String... args) throws IOException {

                        FileWriter  fw = new FileWriter("data.txt");

                                fw.write("Hi, good evening");

                                fw.write("\n");  // it represents new line

                                fw.write("How are you?");

                                fw.flush ( );
```

```
                                fw.close( ) ;
        }

}


// Java program to read file data using FileReader class

import java.io.*;

public class Demo {

        public static void main(String... args) throws IOException {

                FileReader fr = new FileReader ("data.txt");
                int  i = fr.read ( );

                while ( i != -1 ){
                        System.out.print( (char) i );
                        i = fr.read ( );        // read next character and
re-initialize i var
                }
                fr.close ( );
        }

}


// Java Program to read file data using BufferedReader

import java.io.*;

public class Demo {

        public static void main(String... args) throws IOException {


                FileReader fr = new FileReader("data.txt");

                BufferedReader br = new BufferedReader(fr);

                String line = br.readLine ( ); // reading first line data

                while ( line != null ) {
                        System.out.println( line );
                        line = br.readLine ( ) ; // reading next line and
re-initialzing line variable
                }
        }
}
```

Note: FileReader will read the data character by character where as BufferedReader will read the data Line by Line.

File Handling Assignments :

1) Write a java program to find how many characters, how many words and how many lines available in the file

2) Write a java program to read 2 files data and write 2 files content into 3rd file.

3) Write a java program to find names which are available in 2 files.

==========

PrintWriter
==========

System.out.println ("hi");

System ----> It is a class available in java.lang package

out ---> it is a static variable available in System class.
The data type of 'out' variable is PrintWriter class

println ( ) ---> it is a method available in PrintWriter class

```
// Writing data to console using PrintWriter object

package in.ashokit;

import java.io.PrintWriter;

public class Demo {

	public static void main(String[] args) {
		PrintWriter pw = new PrintWriter(System.out);
		pw.print("hi");
		pw.println("hello");

		pw.flush();
		pw.close();
	}
}
```

```
// Writing data to file using PrintWriter object

package in.ashokit;

import java.io.PrintWriter;

public class Demo {

	public static void main(String[] args) throws Exception {
		PrintWriter pw = new PrintWriter("f1.txt");
		pw.write("this is my f1 file data");
		pw.flush();
		pw.close();

	}
}
```

1) File
2) File class methods
3) FileWriter
4) FileReader
5) BufferedReader
6) PrintWriter

==========

Serialization & De-Serialization
==========

-> When we store data in the object, that data will be available if our program is running. If our program got
terminated then we will loose our objects and data available in the objects.

-> If we don't want to loose the data even after program got terminated then we should go for Serialization.

-> The process of converting java object into file data in the form of bits and bytes is called as Serialization.

-> The process of converting file data back to java object is called De-Serialization.

Note: To perform Serialization & De-Serialization we have to implement java.io.Serializable interface which
 is marker interface

```
// Java program on Serialization & De-Serialization

package in.ashokit;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Person implements Serializable {

	/**
	 *
	 */
	private static final long serialVersionUID = -100l;

	int id;
	String name;

	public static void main(String[] args) throws Exception {

		Person p = new Person();
		p.id = 100;
		p.name = "Raju";

		System.out.println("====Serialization Started ========");

		FileOutputStream fos = new FileOutputStream("person.ser");
		ObjectOutputStream oos = new ObjectOutputStream(fos);
		oos.writeObject(p);
		oos.flush();
		oos.close();
		System.out.println("====Serialization completed========");

		System.out.println("==========De-Serialization Started==========");

		FileInputStream fis = new FileInputStream("person.ser");
		ObjectInputStream ois = new ObjectInputStream(fis);
		Object object = ois.readObject();
		Person p1 = (Person) object;
		System.out.println("Id : " + p1.id);
		System.out.println("Name : " + p1.name);
		ois.close();

		System.out.println("==========De-Serialization Ended==========");
```

```
        }

}
```

```
========================
What is SerialVersionUID ?
========================
```

-> For every .class file JVM will assign one random number that is called as serialVersionUID.

-> When we serialize the object, JVM will assign .class file serialVersionUID to serialized file

-> When we de-serialize JVM will compare serialized file UID and .class file UID. If both ids are matching
 then only de-serialization will happen otherwise it will throw INvalidClassException.

-> To overcome this problem we can write our own serialVersionUID then jvm will not assign that.

```
================
transient keyword
================
```

-> If we have any sensitive / secret data then we shouldn't serialize those fileds.

-> transient keyword is used to ignore variables from serialization process

```
public class Person implements Serializable {

        /**
         *
         */
        private static final long serialVersionUID = -100l;

        int id;
        String name;
        String email;
        transient String pwd;

}
```

Note: If we serialize the above person class object, id, name and email will be serialized and pwd will
not get serialized because it is transient variable.

```
package in.ashokit;

import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;

public class Person implements Serializable {
```

```
	/**
	 *
	 */
	private static final long serialVersionUID = -100l;

	int id;
	String name;
	String email;
	transient String pwd;

	public static void main(String[] args) throws Exception {

		Person p = new Person();
		p.id = 100;
		p.name = "Raju";
		p.email = "raju@gmail.com";
		p.pwd = "raj@123";

		System.out.println("====Serialization Started ========");

		FileOutputStream fos = new FileOutputStream("person.ser");
		ObjectOutputStream oos = new ObjectOutputStream(fos);
		oos.writeObject(p);
		oos.flush();
		oos.close();
		System.out.println("====Serialization completed========");

		System.out.println("==========De-Serialization Started==========");

		FileInputStream fis = new FileInputStream("person.ser");
		ObjectInputStream ois = new ObjectInputStream(fis);
		Object object = ois.readObject();
		Person p1 = (Person) object;
		System.out.println("Id : " + p1.id);
		System.out.println("Name : " + p1.name);
		System.out.println("Email : " + p1.email);
		System.out.println("Pwd : " + p1.pwd);
		ois.close();

		System.out.println("==========De-Serialization Ended==========");

	}

}
```

==========
Multi Threading
==========

Task : Work

Single Tasking : Performing only one task at a time is called as Single Tasking

Ex:

1) Explain the topic
2) Dictate the notes
3) Ask questions

-> If we perform single tasking then it will take lot of time to complete all our work.

Multi Tasking : Performing multiple tasks at a time is called as Multi Tasking

Ex:

1) Walking & listening music

2) Speaking and Writing

3) Reading book & eating

-> If we perform multi tasking then we complete multiple works at a time.

-> Multi Tasking we can achieve in 2 ways

1) Process Based Multi Tasking

Ex: Windows OS

2) Thread Based Multi Tasking

-> To execute our program logics paralelly then we need to go for Thread Based Multi Tasking

-> Using Thread Based Multi Tasking our program can complete the work quickly

-> To implement Thread Based Multi Tasking we will use Multi Threading

-> Java Supports Multi Threading

==========
Use case to go for Multi Threading
==========

1) Send sms to all customers at a time
2) Send Email to all customers at a time
3) Generate & Send Bank Statements to all customers in email

Note: The main aim of Multi Tasking is used execute our program logic paralelly so that we can complete more work in less time.

-> For Every Java program execution, JVM will create one thread by default. That thread is called as Main thread.

```
// Java Program to get the details of Main thread


public class Demo {

	public static void main(String... args) {

		Thread currentThread = Thread.currentThread();
		System.out.println(currentThread.getName()); // main

	}
}
```

Note: Thread is a predefined class available in java.lang package. In Thread class we have a static method currentThread ( ).

==========
User Defined Threads
==========

-> In Java we can create Thread in 2 ways

1) By extending Thread class

2) By Implementing Runnable interface

```
// Java program to create user defined thread using Thread class

public class Demo extends Thread {

	public void run() {
		System.out.println("run () method called...");
	}

	public static void main(String... args) {

		Demo d = new Demo();

		Thread t = new Thread(d);
		t.start();
	}
}


// Java program to create the thread using Runnable interface

public class Demo implements Runnable {

	public void run() {
		System.out.println("run () method called...");
	}

	public static void main(String... args) {

		Demo d = new Demo();

		Thread t = new Thread(d);
		t.start();
	}
}
```

==========

Q) What is the difference between extending Thread class and implementing Runnable interface, which is recommended ?

==========

-> If we extend properties from Thread class then we can't extend properties from any other class because

java doesn't support mulitple inheritence. (We are closing gate for Inheritence)

-> If we implement Runnable interface then in future we can extend properties from any class based on requirement.

(Our gate is open for inheritence)

Note: Implementing Runnable interface is always recommended.

==========

What is Thread Schedular

==========

-> Thread Schedular is a program in the JVM which is responsible to schedule Threads execution and

resources allocation required for the thread.

-> When we call start ( ) method then Thread Schedular will start its operation.

1) Allocating Resources

2) Thread Scheduling

3) Thread Execution by calling run ( ) method

==========
start ( ) method vs run ( ) method
==========

-> To start thread execution we will call start ( ) method

t.start ( )

-> once start ( ) method is called then Thread Schedular will come into picture to execute our thread

-> start ( ) method will call run ( ) method internally

-> inside run ( ) method we will write the logic which should be executed by the thread.

==========
Can we call run ( ) method directley without calling start ( ) method
==========

-> Yes, we can call run ( ) method directley but it will execute like a normal method (there is no use)

by "main" thread.

-> If we want to execute run ( ) method as a thread method then we should call start ( ) method then

internally it will call run ( ) method (Thread Schedular will take care of thread execution)

```
public class Demo implements Runnable {

	public void run() {
		System.out.println("run () method started...");

		Thread t = Thread.currentThread();
		System.out.println(t.getName());

		System.out.println("run () method ended...");
	}

	public static void main(String... args) {
		Demo d = new Demo();

		Thread t = new Thread(d);
		//t.start();
		// t.run();
	}
}
```

=> If we call start ( ) method then run ( ) method will be executed by our user defined thread
(we can see thread name as Thread-0)

=> if we call run ( ) method then run ( ) method will be executed by "main" thread
(we can see thread name as main)

```
======================
What is Thread Life Cycle
======================

-> Thread Life cycle contains several phases of Thread execution

1) New

2) Runnable

3) Running

4) Blocked

5) Terminated
```

New: A thread begins its life cycle in the new state. Thread remains in the new state until we will
call start ( ) method.

Runnable : After calling start ( ) method, thread comes from new state to runnable state.

Running : A thread comes to running state when Thread Schedular will pick up that thread for execution.

Blocked : A thread is in waiting state if it waits for another thread to complete its task.

Terminated : A thread enters into terminated state once it completes its task.

```
// Java Program on Thread Sleep

package in.ashokit;

public class Demo implements Runnable {

	public void run() {
		System.out.println("run () method started...");

		try {
			Thread.sleep(5000); // blocked state
		} catch (InterruptedException e) {
			e.printStackTrace();
		}

		System.out.println("run () method ended...");
	}

	public static void main(String... args) {
		Demo d = new Demo();

		Thread t = new Thread(d); // new state
		t.start(); // runnable state
	}
}


// Java program to start mutliple threads to perform same activity

package in.ashokit;
```

```
public class Demo implements Runnable {

	public void run() {
		System.out.println("run () method started..." +
Thread.currentThread().getName());
		try {
			Thread.sleep(15000); // blocked state
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		System.out.println("run () method ended..." +
Thread.currentThread().getName());
	}

	public static void main(String... args) {
		Demo d = new Demo();

		Thread t1 = new Thread(d);
		t1.setPriority(Thread.MAX_PRIORITY); // 10
		t1.setName("Thread-1");

		Thread t2 = new Thread(d);
		t2.setPriority(Thread.NORM_PRIORITY); // 5
		t2.setName("Thread-2");

		Thread t3 = new Thread(d);
		t3.setPriority(Thread.MIN_PRIORITY); // 1
		t3.setName("Thread-3");

		t1.start(); // runnable state
		t2.start(); // runnable state
		t3.start(); // runnable state
	}
}
```

Note: We shouldn't start one thread more than one time.

```
public static void main(String... args) {
		Demo d = new Demo();

		Thread t1 = new Thread(d);

		t1.start();
		t1.start(); // java.lang.IllegalThreadStateException
	}
```

==========
Callable Interface
==========

-> This interface introduced in java 1.5

-> Using Callable interface also we can create the Thread

-> This interface contains call ( ) method.

Syntax:

```
			public Object call ( )
```

==========
What is the difference between Runnable & Callable interfaces
==========

-> Runnable is a functional interface which contains run ( ) method
-> Callable is a functional interface which contains call ( ) method

-> Runnable run ( ) method returns void (no return type)
-> Callable call ( ) method returns Object

-> Runnable interface present in java.lang package
-> Callable interface present in java.util.concurent package

==========
ExecutorService
==========

-> Executor Service introduced in java 1.5v

-> Using ExecutorService we can implement multi threading

-> Using Executors we can create thread pool

-> Using Executor Service we can submit tasks to pool of threads.

-> ExecutorService will re-use threads available in the pool to complete all submitted tasks.

```
// Java Program on Executor Service with Callable interface

package in.ashokit;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

public class Demo implements Callable {

	public Object call() throws Exception {
		System.out.println("call ( ) - method executed...");
		return "success";
	}

	public static void main(String[] args) throws Exception {
		Demo d = new Demo();

		ExecutorService exService = Executors.newFixedThreadPool(10);

		for (int i = 1; i <= 15; i++) {
			Future submit = exService.submit(d);
			System.out.println(submit.get().toString());
		}
		exService.shutdown();
	}
}
```

==========
Daemon Thread
==========

We have 3 types of threads in java

1) Default thread created by JVM (main thread)

2) User Defined Threads (Thread class, Runnable interface, Callable interface)

3) Daemon Threads

Note: The thread which runs in the background is called as Dameon Thread. Daemon

Threads also called
as low priority threads.

Ex: Garbage Collector is a daemon thread

-> We can make our thread as Daemon Thread using setDaemon( ) method

```
// Java Program To Make thread as Daemon

package in.ashokit;

public class Demo implements Runnable {

	@Override
	public void run() {

		if (Thread.currentThread().isDaemon()) {
			System.out.println("Daemon Thread Executed...");
		} else {
			System.out.println("Normal Thread Executed...");
		}

	}

	public static void main(String[] args) {
		Demo d = new Demo();
		Thread t1 = new Thread(d);
		t1.setDaemon(true);
		t1.start();
	}
}
```

-> When JVM reaches end of main method, it will terminate our program. If JVM founds Daemon thread running
 it terminates that daemon thread and then it will shutdown the program.

-> JVM will not care about Daemon Threads running status to stop the program execution.

==========
Synchronization
==========

String ----> Immutable class

StringBuffer ----> Mutable class & synchronized class (Thread safe class)

StringBuilder ---> Mutable class & not-synchronized class (Not Thread Safe class)

Synchronized means Thread safe ===> Only one thread can access the object / resource at a time

Not-Synchronized means Not Thread Safe   => Multiple threads can access same resource / object at a time

```
public class MovieTicketBooking  {

		int avilableTickets = 100;

		public void run ( ) {
			if ( availableTickets > 0 ) {
```

```
                                // logic to bookTicket;
                                -- avilableTickets ;
                        }
                }

                psvm ( ) {
                                Thread t1 = new Thread();
                                Thread t2 = new Thread();
                                Threa  t20 = new Thread();

                                t1..... t20 ---start
                }
}
```

-> In the program, multiple threads are trying to book tickets at a time

Note: If multiple threads access the same object at a time then there is a chance of getting data
 inconsistency problem.

=> To avoid data inconsistency problem, we need to use Synchronization concept

=> Synchronization means allowing only one thread to execute our resource / object / logic at a time

Note: By Using Synchronization we can achieve Thread Safety but it will slow down our execution process.

==========
How to achieve synchronization
==========

-> Using 'synchronized' keyword we can implement synchronization

-> synchronized keyword we can use at two places

1) At method level

2) At block level

Syntax For Synchronized Block:

```
public void m1( ){

        // pre-logic

        synchronized ( object ) {
                // imp business logic
        }

        // post-logic

}
```

Syntax For Synchronized Method :

```
public synchronized void m1( ) {

        // important business logic

}
```

```
// Java Program with Synchronized Method

public class Demo implements Runnable {

	public synchronized void printNums() {
		for (int i = 1; i <= 10; i++) {
			System.out.println(Thread.currentThread().getName() + "=> "
+ i);
			try {
				Thread.sleep(1000);
			} catch (Exception e) {
				e.printStackTrace();
			}
		}
	}

	public void run() {
		printNums();
	}

	public static void main(String[] args) {
		Demo d = new Demo();

		Thread t1 = new Thread(d);
		t1.setName("Thread-1");
		t1.start();

		Thread t2 = new Thread(d);
		t2.setName("Thread-2");
		t2.start();
	}
}
```

Note: In the above program we are starting 2 threads. two threads will access printNums ( ) method
to print the numbers from 1 to 10.

-> If printNums ( ) method having synchronized keyword then two threads will execute the method
sequentially one after other .

-> if we remove synchronized keyword from the printNums ( ) method then two threads will access
that method at a time.

Note: We can see the difference in the output.

==========
Working with Threads using Anonymous Implementation
==========

```
package in.ashokit;

import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MyThread {

	public static void main(String[] args) {

		Thread t1 = new Thread() {
			public void run() {
				System.out.println("run ( ) method logic-1");
```

```
                }
        };
        t1.start();

        Runnable r = new Runnable() {
                @Override
                public void run() {
                        System.out.println("run method() logic-2");
                }
        };

        Thread t2 = new Thread(r);
        t2.start();

        Callable c = new Callable() {
                public Object call() throws Exception {
                        System.out.println("call( ) method logic - 3");
                        return null;
                }
        };

        ExecutorService exService = Executors.newFixedThreadPool(1);
        exService.submit(c);
    }
}
```

==========
Dead Lock
==========

-> Dead Lock means ambiguity problem among the threads

-> If 2 threads are waiting for each other to release the resources is called as dead lock.

-> Once we get into dead lock situation then we can't do anything

Ex:

Thread-1   holding resource-1 and waiting for resource-2

Thread-2 holding resource-2 and waiting for resource-1

Note:

Thread-1 will not release resource-1 hence thread-2 will be in waiting state forever for resource-1

Thread-2 will not release resource-2 hence thread-1 will be in waiting state forever for resource-2

```
// Java program which will give dead lock

package in.ashokit;

public class DeadLock {

    public static void main(String[] args) {

        String s1 = "hi";
        String s2 = "hello";

        Thread t1 = new Thread() {
                public void run() {
```

```
                                synchronized (s1) {
                                        System.out.println("Thread-1 locked
resource-1");
                                        try {
                                                Thread.sleep(100);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                        synchronized (s2) {
                                                System.out.println("Thread-1
waiting for resource-2");
                                        }
                                }
                        }
                };

                Thread t2 = new Thread() {
                        public void run() {
                                synchronized (s2) {
                                        System.out.println("Thread-2 locked
resource-2");
                                        try {
                                                Thread.sleep(100);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                        synchronized (s1) {
                                                System.out.println("Thread-2
waiting for resource-1");
                                        }
                                }
                        }
                };
                t1.start();
                t2.start();
        }
}


==============
join ( ) method
==============

-> join ( ) method is used to hold second thread execution until first thread
execution got completed


package in.ashokit;

public class Demo {

        public static void main(String[] args) throws Exception {

                Thread t1 = new Thread() {
                        public void run() {
                                for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " => " + i);
                                        try {
                                                Thread.sleep(100);
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                }
                        }
                };
                t1.setName("Thread-1");
```

```
                Thread t2 = new Thread() {
                        public void run() {
                                for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " => " + i);
                                        try {
                                                Thread.sleep(100);
                                                Thread.yield();
                                        } catch (InterruptedException e) {
                                                e.printStackTrace();
                                        }
                                }
                        }
                };
                t2.setName("Thread-2");

                t1.start();
                t1.join();
                t2.start();
        }
}
```

==========
yield ( ) method
==========

-> yield ( ) method is used to give chance for other equal priority threads to execute

```
// Java program with yield ( ) method

package in.ashokit;

public class YieldDemo {

        public static void main(String[] args) {
                Thread producer = new Producer();
                Thread consumer = new Consumer();

                producer.start();
                consumer.start();
        }
}

class Producer extends Thread {
        public void run() {
                for (int i = 0; i < 3; i++) {
                        System.out.println("Producer : Produced Item " + i);
                        Thread.yield();
                }
        }
}

class Consumer extends Thread {
        public void run() {
                for (int i = 0; i < 3; i++) {
                        System.out.println("Consumer : Consumed Item " + i);
                        Thread.yield();
                }
        }
}
```

==========

Inter Thread Communication
==========

-> It is used to establish communication among the threads

-> To achieve inter thread communication we have below 3 methods in Object class

1) wait ( )
2) notify ( )
3) notifyAll ( )

Q) Why these 3 methods available in Object class, why not in Thread class ?

-> If these methods available in Thread class then we have to extend Thread class. In future we can't
extend from any other java class bcz java is against for Multiple Inheritence.

-> If these methods available in Runnable interface then everybody should implement these method even
if they don't need inter thread communication.

-> To overcome all these problems, java kept these methods in Object class so that every class will have
access for these methods.

```
// Java Program to establish inter thread communication

package in.ashokit;

public class Customer {

	int amount = 10000;

	synchronized void withdraw(int amount) {
		System.out.println("going to withdraw...");
		if (this.amount < amount) {
			System.out.println("Less balance; waiting for deposit...");
			try {
				wait();
			} catch (Exception e) {
			}
		}
		this.amount -= amount;
		System.out.println("withdraw completed...");
	}

	synchronized void deposit(int amount) {
		System.out.println("going to deposit...");
		this.amount += amount;
		System.out.println("deposit completed... ");
		notify();
	}

	public static void main(String args[]) {
		final Customer c = new Customer();

		new Thread() {
			public void run() {
				c.withdraw(15000);
			}
		}.start();

		try {
			Thread.sleep(20000);
		} catch (InterruptedException e) {
			e.printStackTrace();
```

```
                }

                new Thread() {
                        public void run() {
                                c.deposit(10000);
                        }
                }.start();
        }
}
```

```
================================
Multi Threading Summary
================================
1) What is Multi Tasking
2) What is Multi Threading
3) Advantages of Multi Threading
4) Default Thread in JVM (main)
5) Getting info of main thread ( Thread.currentThread( ) )
6) Creating User Defined Threads
7) By Extending Thread class
8) By Implementing Runnable interface
9) By implementing Callable interface
10) run ( ) method vs call ( )
11) Executor Service
12) run ( ) vs start ( ) method
13) Thread Life Cycle
14) Thread Schedular
15) Synchronization (method & block)
16) What is Thread Safety
17) Thread creation with Anonymous implementation
18) Dead Lock (Java Program to give dead lock)
19) join ( ) method vs yield ( ) method
20) Inter Thread Communication
21) Daemon Threads

--------------------------------------------------------------------------------
--------------------------------

==============
Reflection API
==============

-> Using Reflecting api we can analyze our java classes

-> Reflection api is used to analyze the software


package in.ashokit;

public class Student {

        private int id;
        private String name;

        public void display() {
                System.out.println("hi");
        }

        public void m1() {
                System.out.println("this is m1()");
        }
```

```
	public void m2() {
		System.out.println("this is m2()");
	}
}

package in.ashokit;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.Method;

public class Demo {

	public static void main(String[] args) throws Exception {

		Class clz = Class.forName("in.ashokit.Student");

		Field[] fields = clz.getDeclaredFields();
		for (Field f : fields) {
			System.out.println(f.getName());
		}

		Method[] methods = clz.getDeclaredMethods();
		for (Method m : methods) {
			System.out.println(m.getName());
		}

		Constructor[] constructors = clz.getDeclaredConstructors();
		for (Constructor c : constructors) {
			System.out.println(c.getName());
		}

	}

}

// Accessing private variable outside of the class using Reflection API


package in.ashokit;

public class Student {

	private int age = 20;

	public void getAge() {
		System.out.println("Age : " + age);
	}

}



package in.ashokit;

import java.lang.reflect.Field;

public class Demo {

	public static void main(String[] args) throws Exception {

		// loading class into jvm
		Class clz = Class.forName("in.ashokit.Student");

		// creating object for the loaded class
		Object obj = clz.newInstance();

		// getting the filed whose name is age
```

```
                Field field = clz.getDeclaredField("age");

                // making variable accessible outside of the class
                field.setAccessible(true);

                // set value to field
                field.set(obj, 35);

                Student s = (Student) obj;
                s.getAge();

        }

}
```

```
/// Invoking the method using Reflection API

package in.ashokit;

import java.lang.reflect.Method;

public class Demo {

        public static void main(String[] args) throws Exception {

                Class<?> clz = Class.forName("in.ashokit.Student");

                Method method = clz.getDeclaredMethod("getAge");

                Object obj = clz.newInstance();

                method.invoke(obj, null);
        }
}
```

Q) What is the difference between ClassNotFoundException and NoClassDefFoundError ?

-> If we give wrong class name in Class.forName (" ") then we will get ClassNotFoundException

-> NoClassDefFoundError will occur if jvm not able to find .class file of particular class to load

==========
Garbage Collection
==========

-> Garbage Collection is the process of removing un-used / un-referenced objects from JVMs heap area.

-> Garbage Collection is an in-built process in JVM

-> In JVM, garbage collector available to perform Garbage Collection.

-> Garbage Collector is a daemon thread (runs in background)

```
package in.ashokit;

public class Student {
```

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

                // Object created
                Student s1 = new Student();

                // nullifying (making obj eligible for GC)
                s1 = null;

                System.gc();

        }

        protected void finalize() throws Throwable {
                System.out.println("finalize( ) called...");
        }

}
```

=> When we assign s1=null then s1 become un-referenced object and it is elgible for GC

=> When GC identify un-referenced obj then it will call finalize ( ) method and it will remove that object

from HEAP area (memory clean up)

==========
How to invoke GC in java?
==========

-> In java we have 2 ways to invoke GC

```
                System.gc();

                Runtime.getRuntime().gc();
```

Note: Even if we call above methods there is no gaurantee that JVM will start GC immediatley.
GC execution process will be managed by JVM only.

==========
How GC works internally in JVM ?
==========

-> Garbage Collection works in below phases

1) Stop the world
2) Marking
3) Sweeping
4) Compaction

-> When GC starts it will trigger STOP THE WORLD (all running threads will be stopped for few milli secs)

-> GC will go to JVM Heap area and it will identify un-referenced objects and it will mark them for sweep.

-> GC will sweep marked objects

-> After sweeping completed memory holes will be created in Heap area to clear that GC will perform Compaction
 (it will adjust memory holes).

-> After Compaction, GC will give signal to JVM to continue the execution.

Note: GC process will slow down our program execution hence Sun Microsystem didn't give the chance for

programmers to perform Garbage Collection. It will be managed by JVM.

9 AM - 11 AM IST : Online

Generics
Inner Classes
Enums
Class Loaders

==========
Generics In Java
==========

-> Generics introduced in java 1.5 version

-> Using Generics, we can write our classes / variable / methods which are indendent of data type

-> Generics are used to achieve type safety

```
public void m1 ( Integer i ){

}

m1 (10) ;  // valid

m1 ("hi") ; // in-valid
```

-> The above method is taking Integer as a parameter (It is tightly coupled). If we want to pass String as

a parameter for m1 ( ) method it is not possible. Compiler will not accept it.

-> To overcome above problem we can use Generics in Java.

```
package in.ashokit;

public class Demo<T> {

	public void m1(T arg) {
		System.out.println("Param Recieved : " + arg);
	}

	public static void main(String[] args) throws Exception {
		Demo d = new Demo();

		d.m1(10); // passing int value
		d.m1("hi"); // passing String value
		d.m1(100.51); // passing double value
		d.m1(true); // passing boolean value
	}
}
```

```
// Java program with Generic Type to achieve Type Safety

package in.ashokit;

public class Demo<T> {

	T obj;

	void add(T obj) {
		this.obj = obj;
	}

	T get() {
		return obj;
	}

	public static void main(String[] args) throws Exception {

		Demo<Integer> d1  = new Demo<>();
		d1.add(10);
		System.out.println(d1.get());

		Demo<String> d2 = new Demo<>();
		d2.add("Hi");
		System.out.println(d2.get());
	}
}


// Java Program with Generic Parameters for Constructor

package in.ashokit;

public class Demo<T1, T2> {

	T1 obj1;
	T2 obj2;

	Demo(T1 obj1, T2 obj2) {
		this.obj1 = obj1;
		this.obj2 = obj2;
	}

	void print() {
		System.out.println(obj1 + ", " + obj2);
	}

	public static void main(String[] args) throws Exception {
		Demo<Integer, String> d1 = new Demo<>(101, "Ram");
		d1.print();

		Demo<String, Long> d2 = new Demo<>("Ashok", 798686868l);
		d2.print();

		Demo<String, Boolean> d3 = new Demo<>("Raju", true);
		d3.print();
	}
}
```

```
======================
Generics with wild Cards
======================

Demo< ? extends Number >

=> The above wild card represents Demo class should take any class obj which is
extending from Number class.
```

Auto Boxing
Auto Unboxing
Var Args
Callable
ExecutorService
Generics
Enums

==========
Enums in Java
==========

-> Enum introduced in java 1.5v

-> Enum is a special data type in java

-> Enum data type is used to create pre-defined Constants

-> To declare constants using Enum we will use 'enum" keyword

-> Enum stands for Enumeration

```
	enum WEEKDAYS {
		MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY;
	}

	enum WEEKENDDAYS {
		SATURDAY, SUNDAY;
	}
```

-> When we want to declare pre-defined constants then we will use Enums concept.

==========
Few Points To Remember Related To Enums
==========

1) Enum constants we can't override
2) Enum doesn't support object creation
3) Enum can't extend classes
4) Enum can be created in seperate file or we can create in existing class also

```
package in.ashokit;

public enum Course {

	JAVA, PYTHON, DEVOPS, AWS, DOCKER, KUBERNETES;

}
```

```
package in.ashokit;

public class Demo {

	enum WEEKDAYS {
		MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY;
	}

	enum WEEKENDDAYS {
		SATURDAY, SUNDAY;
	}

	public static void main(String[] args) throws Exception {
```

```
                Course[] values = Course.values();

                for (Course c : values) {
                        System.out.println(c);
                }
        }
}
```

==========
Inner Classes
==========

-> Creating one class inside another class is called as Inner class.

-> Inner classes are also called as Nested Classes.

-> The class which contains other class is called as Outer class

-> The Class which is declared inside outer class is called as Inner class.

```
class  A {

        class B {

        }

}
```

class A - is called as outer class

class B - is called as inner class

-> Inner classes are called as Helper classes

-> Inner classes are hidden from other classes (only outer class can access inner class functionality)

==========
Types of Inner classes
==========

1) Non static inner classes

                        1.1 Regular Inner classes
                        1.2 Method local inner class
                        1.3 Anonymous inner class

2) Static inner classes

```
package in.ashokit;

public class Outer {

        void outerMethod() {
                Inner i = new Inner();
                i.innerMethod();
        }

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

```
                Outer o = new Outer();
                o.outerMethod();
        }

        private class Inner {
                void innerMethod() {
                        System.out.println("inner method called...");
                }
        }
}
```

=> We can't use private / protected modifier for outer classes

-> We can use private modifier for inner classes

==========
Approach-1 : Primitive Data Type ""Collections""
==========

variables --> to store the data

data type --> type of data that we can store in variable

1) primitive data types (byte, short, int, long, float, double, char, boolean)

2) referenced data types (Array, String, Class etc..)

```
int   a = 10;

int   b = 20 ;

int   c = 30 ;
```

I want to store 1000 values ? ---- 1000 variables we need

I wan to store 1 lakh values ? ----> 1 lakh variables ---> Not recommended

-> To overcome this problem we are using Arrays concept in java

==========
Approach-2 : Arrays
==========

-> We can store group of values in single variable

```
int [ ]  a  = new int [ 5000 ] ;

arr[0] = 100;
arr[1] = 200;
arr[2] = 300;
                ....
arr[4999] = 7799;
```

Limitations
---

1) Array size is fixed

2) We can store only homogenious values (same type of values)

```
Student s [ ]  = new Student [ 100 ];

s[ 0 ] = new Student(10, "Raju");  // valid

s[1] = new Student(11, "Rani");  // valid

s[2] = new Employee(101, "Ramesh");  /// invalid
...
s[99] = new Student(999, "john"); // valid

======================
Approach-3 : Object Array
======================

Object [ ]   a = new Object [ 100 ];

a[0] = new Student(101, "Raju");
a[1] = new Student(102, "Rani");

a[2] = new Employee(101, "Raju", 1000.00);
a[3] = new Employee(102, "Rani", 2000.00);

a[2] = new Customer(101, "Raju", 1000.00);
a[3] = new Customer(102, "Rani", 2000.00);


---> Size is fixed
---> insert, update, retrieve, sort the data


To overcome the problems of Arrays we are going to use Collections


-> Collections are used to store group of objects / values

-> Collections are growable in nature
 (dynamically collection size can be increased and decreased based on data)

-> We can store any type of data in Collection
        (homegeneuous & heterogeneuous)

-> Collections providing predefined methods to insert,update, delete, retrieve,
sort etc....

-> Collections is a entity / container which is used to store group of Objects

Collections ----> Collections Framework

Framework means ready made software

Collections is called as framework because it is providing predefined interfaces,
classes and methods to perform
operations on data.



1) Why we need to for Collections ?

2) What is Collection Framework ?

3) Collection Hierarchy

        0) Iterable (I)
        1) Collection (I)
        2) List (I)
        3) Set (I)
```

4) Queue (I)

5) Map (I)

List : It is used to store group of objects (duplicates are allowed)

1) ArrayList
2) LinkedList
3) Vector
4) Stack

Set : It is used to store group of objects (duplicates are not allowed)

1) HashSet
2) LinkedHashSet
3) TreeSet

Queue : It is used to store group of objects (FIFO)

1) PriorityQueue

Map : It is used to store group of objects (Key - Value pair)

1) HashMap
2) LinkedHashMap
3) Hashtable
4) TreeMap
5) IdentityHashMap
6) WeakHashMap

========
Cursors
========

-> Cursors are used to iterate collections (retreive data from collections)

1) Iterator
2) ListIterator
3) Enumeration

-> Collections framework related classes & interfaces are part of java.util package

=================
Collection interface
=================

-> It is super interface for List, Set and Queue

-> Collection interface providing several methods to store and retrieve objects

============
List Interface
============

-> Exetending properties from Collection interface

-> Allow duplicate objects

-> It will maintain objects insertion order

-> It is having 4 implementation classes

```
                1) ArrayList
                2) LinkedList
                3) Vector
                4) Stack


                List l = new List ( );   // invalid

                List l = new ArrayList ( ) ; // valid

                List l = new LinkedList ( ) ;  // valid

========
ArrayList
========
-> Implementation class of List interface
-> Duplicate objects are allowed
-> Insertion order preserved
-> null values are accepted
-> Internal data structure of ArrayList is growable array
-> Default Capacity is 10

=====================
ArrayList Constructors
=====================

1) ArrayList al = new ArrayList ( ) ;

2) ArrayList al = new ArrayList (int capacity);

3) ArrayList al = new ArrayList (Collection c);

==================
Methods of ArrayList
==================
1) add (Object obj )  ---> add object at end of the collection
2) add(int index, Object) --> add object at given index
3) addAll (Collection c) ---> to add collection of objects at end of the collection
4) remove(Object obj)  ---> To remove given object
5) remove(int index)   ----> to remove object based on given index
6) get(int index)     --> to get object based on index
7) contains(Object obj)   ---> To check presense of the object
8)  clear( )  ---> to remove all objects from collection
9) isEmpty ( )  ---> to check collection si empty or not
10) retainAll(Collection c)  --> keep only common elements and remove remaining
object
11) indexOf(Object obj) --> to get first occurence of given obj
12) lastIndexOf(Object obj)  ---> to get last occurance  of given object
13) set(int index, Object obj) ---> replace the object based on given index
14) iterator ( ) --> forward direction
15) listIterator ( ) --> forward & back




public class Demo {

        public static void main(String[] args) {

                ArrayList<Integer> al = new ArrayList<>();
                al.add(10);
                al.add(20);
                al.add(30);
                al.add(40);

                System.out.println("======For Loop Approach=======");
```

```
                // Approach-1
                for (int i = 0; i < al.size(); i++) {
                        System.out.println(al.get(i));
                }
                System.out.println("======For-Each loop Approach=======");

                // Approach-2
                for (Object obj : al) {
                        System.out.println(obj);
                }

                System.out.println("=====Iterator Approach=====");
                // Approach-3
                Iterator iterator = al.iterator();
                while (iterator.hasNext()) {
                        System.out.println(iterator.next());
                }

                System.out.println("=====ListIterator Approach=====");

                // Approach-4
                ListIterator listIterator = al.listIterator();
                while (listIterator.hasNext()) {
                        System.out.println(listIterator.next());
                }

                System.out.println("=====forEach ( ) Approach=====");

                // Approach-5
                al.forEach(i -> {
                        System.out.println(i);
                });

        }
}


public class Student {

        int id;
        String name;

        public Student(int id, String name) {
                this.id = id;
                this.name = name;
        }

        public String toString() {
                return id + " - " + name;
        }
}


public class Demo {

        public static void main(String[] args) {

                ArrayList<Student> al = new ArrayList<>();

                al.add(new Student(1, "Raju"));
                al.add(new Student(2, "John"));
                al.add(new Student(3, "Smith"));
                al.add(new Student(4, "Rani"));

                ListIterator<Student> li = al.listIterator();

                while (li.hasNext()) {
```

```
                        System.out.println(li.next());
                }

                System.out.println("====================");

                while (li.hasPrevious()) {
                        System.out.println(li.previous());
                }
        }
}
```

1) ArrayList class is not recommended for insertions because it has to perform lot of shiftings

2) ArrayList class is recommended for retriveal operations because it will retrieve based on index directley

3) internal data structure is growable array

4) duplicate are allowed

5) homegeneuous & hetereogenious data supported

==========
LinkedList
==========

-> Implementation of List interface

-> Internal data structure is double linked list

-> insertion order preserved

-> duplicate objects are allowed

-> null objects also allowed

-> homogenious & hetereogenious data we can store

```
public class Demo {

        public static void main(String[] args) {

                LinkedList<Integer> ll = new LinkedList<>();

                ll.add(10); // 1 node
                ll.add(20); // 1 node
                ll.add(30); // 1 node
                ll.add(40); // 1 node

                System.out.println(ll); // 10, 20, 30, 40
                ll.add(1, 15);
                System.out.println(ll); // 10, 15, 20, 30, 40

                System.out.println(ll.getLast());

        }
}
```

=======
Vector
======

-> Implementation class of List interface

-> Internal data structure is growable array
-> duplicates are allowed
-> insertion order preserved
-> This is synchronized
-> Vector is called as legacy class (jdk v 1.0)

-> To traverse vector we can use Enumeration as a cursor
-> Enumeration is called as Legacy Cursor (jdk 1.0v)

```
public class Demo {

	public static void main(String[] args) {

		Vector<Integer> v = new Vector<>();

		v.add(100);
		v.add(200);
		v.add(300);
		v.add(null);

		Enumeration<Integer> elements = v.elements();
		while (elements.hasMoreElements()) {
			System.out.println(elements.nextElement());
		}
	}
}
```

====
Stack
=====

-> Implementation class of List interface

-> Extending from Vector class

-> Data Structure of Stack is LIFO (last in first out)

push ( ) ---> to insert object
peek ( ) ---> to get last element
pop ( ) ---> to remove last element

1) ArrayList ---------> Growable Array

2) LinkedList ----------> Double Linked List

3) Vector --------------> Growable Array & Thread Safe

4) Stack ----------> L I F O

1) Iterator ----> forward direction (List & Set)

2) ListIterator ---> forward & backward direction (List impl classes)

3) Enumeration ----> forward direction & supports for legacy collection classes

======
Set
======

-> Set is a interface available in java.util package

-> Set interface extending from Collection interface

-> Set is used to store group of objects

-> Duplicate objects are not allowed

-> Supports Homogenious & hetero genious

==========
Set interface Implementation classes
==========

1) HashSet

2) LinkedHashSet

3) TreeSet

==========
HashSet
==========

-> Implementation class of Set interface

-> Duplicate Objects are not allowed

-> Null is allowed

-> Insertion order will not be maintained

-> Initial Capacity is 16

-> Load Factor 0.75

-> Internal Datastructure is Hashtable

==========
Constructors
==========

HashSet hs = new HashSet( );

HashSet hs = new HashSet(int capacity);

HashSet hs = new HashSet(int capacity, float loadFactor);

```
// Java Program on HashSet

public class Demo {

        public static void main(String[] args) {

                HashSet<String> hs = new HashSet<>(100, 10.05f);

                hs.add("one");
                hs.add("two");
                hs.add("three");
                hs.add("four");
                hs.add("one");
                hs.add(null);

                System.out.println(hs);
                hs.remove("three");
```

```
                System.out.println(hs);

                Iterator<String> iterator = hs.iterator();
                while (iterator.hasNext()) {
                        System.out.println(iterator.next());
                }
        }
}

public class Demo {

        public static void main(String[] args) {

                HashSet<Student> hs = new HashSet<>();

                hs.add(new Student(101, "Raju"));
                hs.add(new Student(102, "Rani"));
                hs.add(new Student(103, "John"));

                Iterator<Student> iterator = hs.iterator();
                while (iterator.hasNext()) {
                        System.out.println(iterator.next());
                }
        }
}
```

==========
LinkedHasSet
==========

-> Implementation class for Set interface

-> Duplicates are not allowed

-> Insertion order will be preserved

-> Internal Data Structure is Hash table + Double linked list

-> Initial capacity 16

-> Load Factory 0.75

```
public class Demo {

        public static void main(String[] args) {

                LinkedHashSet<Integer> lhs = new LinkedHashSet<>();
                lhs.add(10);
                lhs.add(20);
                lhs.add(30);
                lhs.add(null);
                lhs.add(40);
                lhs.add(20);

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

Note: HashSet will not maintain insertion order where as LinkedHashSet will maintain insertion order.
HashSet will follow Hastable data structure where as LinkedHashSet will follow Hashtable + Double Linked List data structure.

========
TreeSet
========

-> Implementation class for Set interface

-> It will maintain Natural Sorting Order

-> Duplicates are not allowed

-> null values are not allowed

Note: When we add null value it will try to compare null value with previous object then we will get NullPointerException.

-> It supports only homogeniuous data

Note : TreeSet should perform sorting so always it will compare newly added object with old object. In order to
 compare the objects should be of same type other wise we will get ClassCastException.

-> Internal data structure is binary tree.

```
public class Demo {

	public static void main(String[] args) {

		TreeSet ts = new TreeSet();

		ts.add("raja");
		ts.add("raja");
		ts.add("rani");
		ts.add("ashok");

		System.out.println(ts); // ashok, raja, rani

		Iterator iterator = ts.iterator();
		while(iterator.hasNext()) {
			System.out.println(iterator.next());
		}
	}
}
```

List interface & implementation classes

- duplicates allowed
- insertion order maintained
- homogenious & heterogenious data allowed

Ex: ArrayList, LinkedList, Vector & Stack

Set interface & implementation classes

- Duplicates not allowed
- only LHS will maintain insertion order
- TreeSet supports only homogenious data (For sorting)

Ex: HashSet, LinkedHashSet & TreeSet

Cursors : To traverse collection objects

1) Iterator
2) ListIterator
3) Enumeration

=====
Map
=====
-> Map is an interface available in java.util package

-> Map is used to store the data in key-value format

-> One Key-Value pair is called as one Entry

-> One Map object can have multiple entries

-> In Map, keys should be unique and values can be duplicate

-> If we try to store duplicate keys in map then it will replace old key data with new key data

-> We can take Key & Value as any type of data

Ex:-1 (Map<Integer,String>)
---
101 - John
102 - Smith
103 - David
104 - Robert
105 - Orlen
101 - Charles

Ex:-2 (Map<String, Integer>

India - 120
USA - 30
UK - 20

-> Map interface having several implementation classes

1) HashMap
2) LinkedHashMap
3) TreeMap
4) Hashtable
5) IdentityHashMap
6) WeakHashMap

==========
Map methods
==========

1) put (k,v) ---> To store one entry in map object

2) get(k) ---> to get value based on given key

3) remove(k) ---> to remove one entry based on given key

4) containsKey(k) ---> to check presense of given key

5) keySet ( ) ---> To get all keys of map

6) values ( ) ----> To get all values of the map

7) entryset ( ) --> to get all entries of map

8) clear ( ) --> to remove all the entries of map

9) isEmpty ( ) --> To check weather map obj is empty or not

10) size ( ) --> to get size of the map (how many entries avaiable)

```
public class Demo {

	public static void main(String[] args) {

		Map<Integer, String> map = new HashMap<>();

		map.put(101, "John");
		map.put(102, "Smith");
		map.put(103, "Orlen");
		map.put(102, "David");

		System.out.println("Map size :: " + map.size()); // 3

		System.out.println(map.get(101)); // john
		System.out.println(map.get(300)); // null

		Collection<String> values = map.values();
		for(String v : values) {
			System.out.println(v);
		}

		Set<Integer> keySet = map.keySet();
		for (Integer key : keySet) {
			System.out.println(key +"--"+map.get(key));
		}


		Set<Entry<Integer, String>> entrySet = map.entrySet();
		/*Iterator<Entry<Integer, String>> iterator = entrySet.iterator();
		while(iterator.hasNext()) {
			Entry<Integer, String> entry = iterator.next();
			System.out.println(entry.getKey()+"--"+entry.getValue());
		}*/

		for(Entry<Integer,String> entry : entrySet) {
			System.out.println(entry.getKey()+"--"+entry.getValue());
		}

		System.out.println(map.containsKey(102));
		System.out.println(map.containsKey(200));
		System.out.println(map.isEmpty());

		map.clear();

		System.out.println(map.size());
	}
}

public class StudentMapDemo {

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

```
		Student s1 = new Student(101, "John");
		Student s2 = new Student(102, "Smith");
		Student s3 = new Student(103, "Orlen");

		Map<Integer, Student> map = new HashMap<Integer, Student>();
		map.put(1, s1);
		map.put(2, s2);
		map.put(3, s3);
		/*
		Set<Integer> keySet = map.keySet();
		for(Integer key : keySet) {
			System.out.println(map.get(key));
		}*/

		Set<Entry<Integer,Student>> entrySet = map.entrySet();
		for(Entry<Integer,Student> entry : entrySet) {
			System.out.println(entry.getKey());
			System.out.println(entry.getValue());
		}
	}
}
```

==========
HashMap
==========

-> It is impl class for Map interface

-> Used to store data in key-value format

-> Default capacity is 16

-> Load factor 0.75

-> Underlying datastructure is hashtable

-> Insertion Order will not be maintained by HashMap

==========
LinkedHashMap
==========

-> Implementation class for Map interface

-> Maintains insertion order

-> Data structure is hashtable + double linkedlist

==========
TreeMap
==========

-> Implementation class for Map interface

-> It maintains natural sorted order for keys

-> Internal Data structure for Tree map is binary tree

==========
Hashtable
==========

-> It is implementation class for Map interface
-> Default capacity is 11
-> Load factor 0.75

-> key-value format to store the data
-> Hashtable is legacy class (jdk 1.0 v)
-> Hashtable is synchronized

-> If thread safety is not required then use HashMap instead of Hastable.
-> If thread safety is important then go for ConcurrentHashMap instead of Hashtable.

=======
Queue
=======

-> It is extending properties from Collection interface

-> It is used to store group of objects

-> Internal Data structure is FIFO (First in First out)

-> It is ordered list of objects

-> insertion will happen at end of the collection

-> Removal will happen at beginning of the collection

```
// java program with Priority Queue
public class QueueDemo {

	public static void main(String[] args) {

		PriorityQueue<String> queue = new PriorityQueue<>();

		queue.add("john");
		queue.add("smith");
		queue.add("orlen");
		queue.add("charles");

		System.out.println(queue);

		System.out.println(queue.element());
		System.out.println(queue.peek());

		Iterator<String> iterator = queue.iterator();
		while(iterator.hasNext()) {
			System.out.println(iterator.next());
		}

		queue.remove();
		queue.poll();

	}

}

// Java program with ArrayDequeue

public class ArrayDequeueDemo {

	public static void main(String[] args) {

		ArrayDeque<String> ad = new ArrayDeque<>();

		ad.add("one");
		ad.add("two");
```

```
                ad.add("three");
                ad.addFirst("ashok");

                System.out.println(ad);

                ad.pollFirst();
                System.out.println(ad);
                ad.pollLast();

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

1) Collections Sorting (Comparator)
2) Fail-Fast & Fail-Safe Collections
3) Properties class
4) Collection Usecases
5) java.util.Date & java.util.Calendar
6) java.util.StringTokenizer

==========
Properties Class In Java
==========

-> Properties is a predefined class available in java.util package
-> Properties class extending properties from Hashtable class
-> Properties class is used to avoid hardcoding in the project

Note: Hardcoding means fixing values in the program (Ex; database properties)

==========
Use-case
==========

-> If java application wants to communicate with database, then we need to configure database credentials
in java program.

-> If we hardcode database credentials in java program then project maintenence will become difficult why
because in future if database credentials modified then we need to modify our java program also.

-> If java program is modified then we need to re-compile and re-execute our program/project which will take
 lot of time and it may break existing code.

=> To overcome above problems we should not do hardcoding.

=> To avoid hard coding in java projects we will use java.util.Properties class.

-> Properties class is used to read the data from properties file

-> properties file contains data in key-value format (like map)

Ex:

```
uname=ashokit
pwd=ashokit@123
```

Note: Properties file extension will be .properties

-> Create database.properties file with below data in project folder

```
------------ database.properties ------------
uname=ashokit
pwd=ashokit@123
---------------------------------------------------------
```

-> Create below java class to read data from properties file

```
package in.ashokit;

import java.io.FileInputStream;
import java.util.Properties;

public class DatabaseApp {

	public static void main(String[] args) throws Exception {

		FileInputStream fis = new FileInputStream("database.properties");

		Properties p = new Properties();
		p.load(fis); // load all the properties from properties file

		System.out.println(p);

		String uname = p.getProperty("uname");
		String pwd = p.getProperty("pwd");
		String driver = p.getProperty("driver"); // key not present

		System.out.println("Username: " + uname);
		System.out.println("Password: " + pwd);
		System.out.println("Driver: " + driver); // null

		fis.close();

	}

}
```

==========
Collections Sorting
==========

Collection is a container which is used to store group of objects

Note: Collection interface is available in java.util package

List, Set & Queue interfaces are extending properties from Collection interface

-> In Collections framework we have a class called Collections class

Note: Collections is a predefined class available in java.lang package

-> Collections class provided several static methods to perform operations on data like below

```
Collections.sort(al);

Collections.reverse(al);
```

Q) What is the difference between Collection, Collections & Collections Framework ?

Defintion : Collection is a container to store group of objects

We have an interface with a name Collection (java.util). It is root interface in Collections framework.

Collections is a class available in java.util package
(Providing ready made methods to perform operations on objects)

Collection interface & Collections class are part of Collections framework. Along with these 2 classes there are
 several other classes and interfaces in Collections framework.

```
// Java program on Collections class
package in.ashokit;

import java.util.ArrayList;
import java.util.Collections;

public class Demo {

	public static void main(String[] args) {

		ArrayList<Integer> al = new ArrayList<>();

		al.add(5);
		al.add(3);
		al.add(4);
		al.add(1);
		al.add(2);

		System.out.println("Before Sort : " + al);

		// Sort the collection
		Collections.sort(al);

		System.out.println("After Sort : " + al);

		// Reverse the collection
		Collections.reverse(al);

		System.out.println("After Reverse : " + al);
	}
}
```

Note: In the above program we have added Integer values in the collection. Integer is a class wrapper and it is
implementing Comparable interface already.

=> If we want apply sorting on User-Defined objects like Student, Employee, Product, Customer etc... then we have
 2 approaches

	1) Comparable (java.lang)

	2) Comparator (java.util)

==========
Comparable
==========

-> Comparable is a predefined interface available in java.lang package
-> Comparable interface having compareTo (Object obj) method
-> compareTo ( ) method is used to compare an object with itself and returns int value

```
                        if( obj1 > obj2 ) ----> returns +ve no

                        if( obj1 < obj2 ) ----> return -ve no

                        if ( obj1 == obj2 ) ----> return zero (0)


// Student class

package in.ashokit;

public class Student implements Comparable<Student> {

        int id;
        String name;
        int rank;

        public Student(int id, String name, int rank) {
                this.id = id;
                this.name = name;
                this.rank = rank;
        }

        @Override
        public int compareTo(Student s) {
                return this.id - s.id;
                // return this.name.compareTo(s.name);
                // return this.rank - s.rank;
        }

        @Override
        public String toString() {
                return "Student [id=" + id + ", name=" + name + ", rank=" + rank +
"]";
        }

}


package in.ashokit;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class StudentDemo {

        public static void main(String[] args) {

                List<Student> al = new ArrayList<>();

                al.add(new Student(101, "John", 3));
                al.add(new Student(104, "Anil", 4));
                al.add(new Student(102, "Smith", 2));
                al.add(new Student(103, "Robert", 1));

                Collections.sort(al);

                for (Student s : al) {
                        System.out.println(s);
                }

        }
}
```

Note: Comparable interface will allow us to sort the data based on only one value.
If we want to change our sorting
 technique then we need to modify the class which is implementing Comparable
interface. Modifying the code everytime
is not recommended.

==========
Comparator
==========

=> Comparator is a predefined interface available in java.util package
=> Comparator interface having compare(Object obj1, Object obj2) method

```
package in.ashokit;

public class Employee {

	int id;
	String name;
	double salary;

	public Employee(int id, String name, double salary) {
		super();
		this.id = id;
		this.name = name;
		this.salary = salary;
	}

	@Override
	public String toString() {
		return "Employee [id=" + id + ", name=" + name + ", salary=" +
salary + "]";
	}
}
```

```
package in.ashokit;

import java.util.Comparator;

public class EmpIdComparator implements Comparator<Employee> {

	@Override
	public int compare(Employee e1, Employee e2) {
		return e1.id - e2.id;
	}

}
```

```
package in.ashokit;

import java.util.Comparator;

public class EmpNameCompartor implements Comparator<Employee> {

	@Override
	public int compare(Employee e1, Employee e2) {
		return e1.name.compareTo(e2.name);
	}

}
```

```
package in.ashokit;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
```

```
public class EmpDemo {

	public static void main(String[] args) {

		ArrayList<Employee> emps = new ArrayList<>();

		emps.add(new Employee(101, "David", 15000.00));
		emps.add(new Employee(105, "Putin", 25000.00));
		emps.add(new Employee(103, "Cathy", 45000.00));
		emps.add(new Employee(104, "Anny", 35000.00));

		// Collections.sort(emps, new EmpIdComparator());
		// Collections.sort(emps, new EmpNameCompartor());

		Collections.sort(emps, new Comparator<Employee>() {
			@Override
			public int compare(Employee e1, Employee e2) {
				if (e1.salary > e2.salary) {
					return -1;
				} else if (e1.salary < e2.salary) {
					return 1;
				} else {
					return 0;
				}
			}
		});

		for (Employee e : emps) {
			System.out.println(e);
		}

	}

}
```

```
============================
Fail Fast and Fail Safe Collections
============================

-> Collections are divided into 2 types

1) Fail Fast Collections

2) Fail Safe Collections


-> Fail Fast collections will throw error immediatley when we modify collection
object while traversing the collection

	Ex: ArrayList, LinkedList, Vector, HashSet, LHS etc...

Note: Fail Fast collections will throw concurrent modification exception when
collection is modified


-> FailSafe collections will not throw any error even if we modify collection
object data (Add / Remove) while traversing

	Ex: CopyOnWriteArrayList,  ConcurrentHashMap etc...
```

```
public class Demo {
```

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

		// Fail Fast Collection
		ArrayList<Integer> al = new ArrayList<>();

		al.add(100);
		al.add(200);
		al.add(300);
		al.add(400);

		for (int i : al) {
			System.out.println(i);
			if (i == 100) {
				al.add(150);
			}
		}
	}
}

public class Demo1 {

	public static void main(String[] args) {

		// Fail Safe Collection
		CopyOnWriteArrayList<Integer> al = new CopyOnWriteArrayList<>();

		al.add(100);
		al.add(200);
		al.add(300);
		al.add(400);

		for (int i : al) {
			System.out.println(i);
			if (i == 100) {
				al.add(150);
			}
		}
		System.out.println(al);
	}
}



package in.ashokit;

import java.util.HashMap;
import java.util.Iterator;
import java.util.Set;

public class Demo2 {

	public static void main(String[] args) {

		HashMap<Integer,String> map = new HashMap<>();

		map.put(101, "one");
		map.put(102, "two");
		map.put(103, "three");

		Set<Integer> keySet = map.keySet();
		Iterator<Integer> iterator = keySet.iterator();
		while(iterator.hasNext()) {
			System.out.println(iterator.next());
			map.put(104, "four");
		}

	}
```

```
}


package in.ashokit;

import java.util.Iterator;
import java.util.Set;
import java.util.concurrent.ConcurrentHashMap;

public class Demo3 {

	public static void main(String[] args) {

		ConcurrentHashMap<Integer, String> map = new ConcurrentHashMap<>();

		map.put(101, "one");
		map.put(102, "two");
		map.put(103, "three");

		Set<Integer> keySet = map.keySet();
		Iterator<Integer> iterator = keySet.iterator();
		while (iterator.hasNext()) {
			System.out.println(iterator.next());
			map.put(104, "four");
		}

		System.out.println(map);

	}
}
```

```
============================================================
Q) What is the difference between HashMap and IdentityHashMap ?
============================================================

=> HashMap will use equals ( ) method to compare content of keys to find duplicate
keys

=> IdentityHashMap will use == operator to compare address of keys to find
duplicate keys
```

```
package in.ashokit;

import java.util.HashMap;
import java.util.IdentityHashMap;

public class Demo4 {

	public static void main(String[] args) {

		HashMap<String, Integer> hm = new HashMap<>();

		// HM will compare content of keys to find duplicate keys
(equals())

		hm.put("ashok", 101); // 1 entry added
		hm.put("raja", 102); // 1 entry added
		hm.put("rani", 103); // 1 entry added
		hm.put(new String("ashok"), 104); // it will replace first entry
value bcz key is duplicae

		System.out.println("HM - Size :: " + hm.size());
		System.out.println(hm);

		System.out.println("=================");
```

```
                IdentityHashMap<String, Integer> ihm = new IdentityHashMap<>();

                // IHM will compare address of keys to find duplicate keys (==)

                ihm.put("ashok", 101); // 1 entry added (scp)
                ihm.put("raja", 102); // 1 entry added
                ihm.put("rani", 103); // 1 entry added
                ihm.put(new String("ashok"), 104); // 1 entry added
                ihm.put("ashok", 105); // it will replace first entry value

                System.out.println("IHM - Size :: " + ihm.size());
                System.out.println(ihm);
        }
}


============================================================
Q) What is the difference between HashMap and WeakHashMap ?
============================================================

=> HashMap keys will have strong reference that means they will maintain a
reference hence they are not elgible for
 Garbage COllector

=> WeakHashMap keys will have weak reference that means they are eligible for
Garbage Collection.
=> GC will dominate WeakHashMap


public class Demo5 {

        public static void main(String[] args) {

                WeakHashMap<Integer, String> whm = new WeakHashMap<>();

                whm.put(1, "hi");
                whm.put(2, "hello");
                whm.put(4, "java");
                whm.put(3, "bye");

                System.out.println(whm);
        }
}


============================
Collections Framework Summary
============================

1) Why Collections ?
2) What is Collection
3) What is Collection Framework ?
4) Collections Hierarchy
5) List interface

                a) ArrayList
                b) LinkedList
                c) Vector
                d) Stack

6) Set interface

                a) HashSet
                b) LinkedHashSet
                c) TreeSet

7) Queue
                a) PriorityQueue
```

b) ArrayDeque

8) Cursors

a) Iterator
b) ListIterator
c) Enumeration

9) Collection (I) & Collections (C)

10) Map interface

a) HashMap
b) LinkedHash
c) TreeMap
d) Hashtable
e) IdentityHashMap
f) WeakHashMap

11) Properties class

12) Collections Sorting

a) java.lang.Comparable (compareTo(Object obj1))
c) java.util.Comparator (compare(Object obj1, Object obj2)

