Chapter : 1

Introduction to Java and Elemantary programming

(1) Define API, JDK, IDE, JRE, JIT, Java virtual machine, Byte Code, Unicode.

→ * **API** : API stands for Application programming Interface
→ API is a collection of Java library and classes and Interface.
→ This classes and Interfaces are very useful for devloping Java programs.
→ This API is still expanding that means new, classes, interfaces and methods are still getting introduced to support Java application devlopment.
→ Java programming language comes with three editions:

(i) Java Standard edition : It is mainly used to devlope client side applications. These are simple Java programs or applets. Applets program run with help of web browser.

(ii) Java Enterprise edition : It is mainly used to devlope serverside application. Application is devloped in JSP.

(iii) Java micro edition : It is used to devlope application for mobile device.

* JDK

-> JDK stands for Java devlopment kit.

-> It is a software required for running Java programs.

-> Oracle releases each version of JDK.

-> JDK is a cross platformed software devlopment enviroment that offers a collection of tools and library necessary for devloping Java based software application.

* IDE

-> IDE stands for Integrated devlopment enviroment.

-> It provides enviroment to devlope Java program effectively and efficiently.

-> Editing, compiling, debuging and online help are Integrated in graphical user interface.

-> The most commonly used IDEs are NetBeans, Eclips, InteliJ and so on.

* JRE

-> JRE stands for Java run time enviroment.

-> JRE is a part of JDK. It is freely avalaible software which has Java class library and specific tool.

-> It is most common enviroment avalaible on devices to run Java program.

-> Java codes get compile and converted to Java bytecode. If you want to run Java byte code on any devices you require JRE.

-> It includes library like Java managment extension (JMEx), Java native interface (JNI), and Java for xML processing (JAx-ws).

* JIT

-> JIT stands for Just in time.
-> JIT compiler is an essential part of JRE. that is responsible for perfomance optimization of Java based application at run time.
-> Compiler is one of the key aspect in deciding perfomance of an application for both parties the end user and the application devloper.

* JVM

-> JVM stands for Java virtual machine.

-> It is an abstract machine. It is a specification that provides runtime enviroment in which Java bytecode can be executed.

-> It is avalaible for many hardware and software platforms.

-> JVM performs following operations, Loads code, Verifies code, Executes code, provide run time enviroment.

* Byte code

-> Java bytescode is the instruction set of JVM. It acts similar to an assembler which is an alias representation of C++.

-> As soon as a Java program is compiled, Java bytescode is genrated.

-> Java bytecode is the machine code in the form of a class file. with the help of Java bytecode we achieve platform independence in Java.

* Unicode

→ Unicode is a universal international standard character encoding that is capable of representing most of the world's written languages.

→ Before Unicode there were many languages standard :

ASCII for U.S.
ISO 8859-1 for western europe language.
KOI-8 for russian.
BIG-5 for chinese.

→ These languages cause problems like,
(i) A particular code value corresponds to different letters in the various language standards.
(ii) The encoding for languages with large character sets have variable length. Some common character encoded as single byte.

→ To solve these problems new language standard Unicode system was devloped. In this character holds 2 bytes, so Java also uses 2 bytes for characters..

(2) Feautures of Java.

→ (1) Java can be compiled and interpreted.

→ Normally programming language can be either compiled or interpreted but Java is a language which can be compiled as well as interpreted.

→ First, Java compiler translate the Java source program into special code called bytecode. Then Java interpreter interprets this bytecode to obtain equvaliant machine code. This machine code is directly executed to obtain the output.

→ (2) Java is a platform independent and portable programming language.

→ Platform independence is the most exciting feauture of Java program. That means programs in Java can be executed on variety of platforms.

→ This feauture based on the goal - write once, run anywhere and at anytime forever.

→ The primitive data types used in Java are machine independent.

→(3) Java is known as OOP language.

→ Java is a true object oriented language as everything in Java is an object.

→ In Java all the codes and data lines within the classes and object.

→(4) Java is robust and secure.

→ Javed ensure the reliable code. The data types are strictly checked at compile time as well as run time in Java.

→ The memory managment technique is supported by garbage collection technique. This technique eliminates various memory managment problems.

→ The exception handling feauture of Java helps the programmer to handle serious error without crashing overall system.

→(5) Java is designed for distributed system.

→ This feauture is very much useful in networking enviroment. In Java two different object on different computer can communicate ~~ech~~ each other. This can be achieved by Remote method Invocation. This feauture is useful in client-server communication.

→(6) Java is simple and small programing language.

→ Java is a simple programing language. Even though you have no programming background you can learn this language easily.

→(7) Java is a multithread and interactive languge.

→ Java supports multi threaded programming which allows a programmer to write such a program that can perform many task similarly.

→ This allows programmer to devlope interactive systems.

→(8) Java is known for it's high perfomance, scalability, monitoring and manageability.

→ Due to use of bytecode Java is high perfomance. The use of multi-threading also improve perfomance of Java.

→ The J2SE helps to increase scalability.

→ For monitoring and managment Java has large numbers of API.

→(9) Java is a dynamic extensible language.

→ This language is capable of dynamically linking new classes, libraries, methods and objects.

→ Java also support function written in C and C++. This function is called native methods.

→(10) Java can be devloped with ease.

→ There are various features of Java such as Generics, Static import, annotations and so on which help Java programmer to create a error free revsable code.

(3) Explain Programming errors.

→ There are three types of programming error.
(1) Syntax errors (3) Logic errors
(2) Run time errors

→ (1) Syntax errors : The error detected by compiler are called syntax or compile error.

Syntax error occurs due to writing wrong syntax of the code such as missing paranthesis, mistyping the keywords.

```
public class Test
{
  public static void main (String [] args)
   {
     System.out.println ("welcome );
   }
}
```

The above program will cause syntax error as closing double quote is missing in last statment.

→(2) Run-time error : This error occurs during the execution of program.
This error occurs when the program finds it difficult to execute for given input condition.

Ex. If program expect Integer number and user enter string value then run time error occures.

→(3) Logic error : It is error that does not perform the way it is intended to.

→ Missing Braces : The braces define area of block. But brace is missing that it cause syntax error.

→ Missing semicolon : The every line in the program must be terminated by semicolon except control structure. If semicolon is missing it cause syntax error.

→ Missing quotation : String is surrounded by quotation marks. Many time programmer forget to place terminating quote and then this error occurs.

→ Misspelling names : Java is case sensitive. Normally programmer make comon mistake of misspelling the names.

(4) Define Identifiers and Variables.

→ * Identifiers

→ Identifiers are kind of tokens defined by programmer. They are used for naming classes, method, variable, interfaces in the program.

→ Rules :

- It can be written using alphabates, digits, underscore and dollar sign.
- They should not contain any other special character within them.
- There should not be space within them.
- It can't start with digit, It should always start with alphabate.
- Identifires are case-sensitive.
- Identifires can be of any length.

* Variable

- It is an identifire that denotes storage locati
- Variable is a fundumental unit of storage in Java.
- The variables are used in combination identifires.
- The variable must be declare before it's use.

- Syntax.

data-type name-of-variable [= Initialization];

- Ex.

```
int a, b;
char m = 'a';
byte k=12, p, t=22;
```

* Keywords

→ Keywords are reserved words by Java and compiler.

abstract, assert, boolean, byte, break, case, catch, char, class, const, continue, do, default, double, else, extends, final, for, float, goto, if, import, instanceof, int, long, native, new, Package, Private, protected, public, return, Short, Static, super, this, Switch, try, void, volatile, while, true, false

(5) Data types and it's Operation.

-> byte : This is smallest integer data type.
width : 8-bits
range : -128 to 127

```
byte i,j;
```

-> Short : It is used for defining the signed numerical variables.
width : 16-bit
range : -32,768 to 32,767

```
Short a,b;
```

-> int : It is used to define the numerical data. most commonly used data type.
width : 32-bit
range : 2,147,483,648 to 2,147,493,647

```
int c,d;
```

-> long : Some times int is not efficient for declaring data then long is used.
Range of long is very long.

```
long e;
```

Last

-> boolean : Boolean is simple data type which denotes value to be either true or false.

-> float : It is used to represent real number.
width : 32-bit,
range : 1.4e-045 to 3.4e+0.38

-> double : It is used to represent real number of large range.
width : 4.9e-324 to 1.8e+308

-> char : It is used to represent character type of data, range is 0 to 65,536.
width : 16-bit.

```
Class Datatype Demo {
    public static void main (String arg[]) {
        byte a = 10;
        short b = 10*128;
        int c = 10000*128;
        long d = 1000*1000*128;
        double e = 99.998;
        char f = 'a';
        boolean g = true;
    System.out.println ("Value of a = " +a);
    System.out.println ("Value of b = " +b);
    System.out.println ("Value of c = " +c);
    System.out.println ("Value of d = " +d);
    System.out.println ("Value of e = " +e);
    System.out.println ("Value of f = " +f);
    f++;
    System.out.println ("Value of f++ = " +f);
    System.out.println ("Value of g = " +g);
    }
}
```

Output

```
Value of a = 10
Value of b = 1280
Value of c = 1280000
Value of d = 1280000000
Value of e = 99.998
Value of f = a
Value of f++ = b
Value of g = true
```

(6) Explain types of operator and Precedence relation.

* Types of Operator

Type	Operator	Meaning	Example
Arithmetic	+	Addition	c = a + b
	-	Substraction	d = -a
	*	Multiplication	c = a * b
	/	Devision	c = a / b
	%	Mode	c = a % b
Relational	<	Less than	a < b
	>	Greater than	a > b
	<=	Less than equal to	a <= c
	>=	Greater than equal to	a >= d
	==	Equal to	a == a
	!=	Not equal to	a != b
Logical	&&	And operator	0 && 1
	||	OR operator	0 || 1
Assignment	=	is assign to	a = 5
Incrament	++	Increment by one	++i, i++
Decreament	--	Decrement by one	--K, K--

* Precedence Relation

→ It specifies which operation must be done first during expression evolution.
→ The operator in the same row indicates the equal precedence.
→ In below table operator are arranged in highest priority to lowest priority.

Name	Operator
Post Increment/decrement	++ , --
Unary operator	~ , !
Multiplicative	* , /
Additive	+ , -
Shift	<< , >> , >>>
Relation	< , > , <= , >=
Equality	== , !=
bitwise end	&
bitwise exclusive OR	^
bitwise inclusive OR	|
Logical ~~en~~ AND	&&
Logical OR	||
ternary	? :

→ Program for arithmetic Operator,

```
class arithOperDemo
{
public static void main(String [] args)
{
 int a = 10, b = 20, c ;
 System.out.println ("a = " + a) ;
 System.out.println ("b = " + b) ;
 c = a + b ;
 System.out.println("\n Addition = " + c) ;
 c = a - b
 System.out.println ("\n Substraction = " + c) ;
 System.out.println ("\n Multiplication = " + a*b) ;
 System.out.println ("\n Division = " + a/b) ;
 }
}
```

Output

```
a = 10
b = 20
Addition = 30
Substraction = -10
Multiplication = 200
Division = 0
```

→ Program for relational operator. and Increment Decrement operator.

```
class Demo
{
public static void main (String [] args)
{
System.out.println ("\n performing Relation operator");
int a = 10, b = 20, c = 10;
System.out.println (" a<b is" + (a<b));
System.out.println (" a>b is" + (a>b));
System.out.println (" a==c is" + (a==c));
System.out.println (" a<=b is" + (a<=b));
System.out.println (" a>=c is" + (a>=c));
System.out.println (" a!=b is" + (a!=b));
System.out.println ("Increment and
                     decrement operation");
System.out.println ("Pre Increment a is" ++a);
System.out.println ("Post Increment b is" b++);
System.out.println ("After settlement value of
                     b is" +b);
} }
```

Output:

```
Performing Relation operator
a<b is true
a>b is false
a==c is true
a<=b is true
a>=c is true
a!=b is true
Increment Decrement operation
PreIncrement a is 11
Post Increment b is 20
After settlement value of b is 21
```

(3) Explain Number type conversion with example. [or]
Define : Casting, widening a type, narrowing a type.

→ Number type conversion is also called type casting.

→ Casting : It converts value of one data type into a value of another data type.

→ Widening a type : Casting a type with small range to a type with large range is called

→ narrowing a type : Casting a type with large range to a type with small rang is called

→ Java will automaticaly widen a type but you must narow a type explicitly.

→ Ex. System.out.println ((int) 2.5);
output = 2

→ Casting from large range type to smaller range causes loss of precision. Example in above ex., the value 2.5 will be converted into 2. Hence while perfoming arithmetic operation result will differ by 0.5.

→ Similarly we can convert smaller data type value to larger data type value.

Ex- System.out.println(1/2);

Output = 0.5

Because 1 is converted into 1.0 and 2 is converted into 2.0 Hence output will be 0.5.

(8) Compare sequential programing and object oriented programming.

Sequential Programming	OOP
→ Program is divided into small parts called function.	Program is divided into Parts called Object.
→ Follows Top down approach.	Follows bottom up approach
→ Data can move freely from function to function	Object can move and communicate with each other.
→ Add new data and function is not easy.	Add new data and function in easy way.
→ Does not have proper way for hiding data.	OOP have proper way for hiding data
→ less secure	More secure
→ overloading is not possible.	overloading is possible in the form of function.
→ Not suppor Inheritance	Support Inheritance. extending existing property of class.
→ Ex. C, VB, Pascal	Ex. C++, Java, VB.NET

(9) Define following terms. OR Define OOP concept.

→ Object: Any entity that has state and behaviour known as object.
Ex. chair, pen, table, keyboard

Class collection of object is called class.

→ Abstraction: Hiding internal detail and showing functionality is called abstraction.
A Java class is a example of abstraction. In Java we use abstract class and interface to achieve abstraction.

→ Encapsulation: Binding code and data together into a single unit is called encapsulation.
Java class is a example of encapsulation. Java has been fully encapsulated class because all data members are private.

→ Inheritance: When one object acquires all properties and behaviour of parent object known as Inheritance.

→ Polymorphism : When one task is performed by different ways is known as polymorphism.
In Java we use method oveloading and overriding to acheive polymorphism.

→ Abstraction : Hiding the access properties (class keyword)

Encapsulation : Binding data and method which operates one data in single body. (keyword - class, interface)

Inheritance : Hiring the properties of existing class (keyword - extends, implements)

Polymorphism : Single thing is avalaible in many form. (Poly - many, morphism - forms)

Ch - 3

Methods and Array

(1) Explain method overloading and overriding with suitable example.

→ Overriding : If subclass has same method as declare parents class, it is known as method overriding.

- Used to provide the specific implementation of method which is already provided by superclass.
- Rules : Method must have same name as parant class.
  - Method must have same parameter as parant.
  - There must be IS-A Relationship.

→ Overloading : If a class have multiple method having same name but different parameters it is known as method overloading.

- Method overloading increase reachability of program.
- Ways : By changing number of argument.
  By changing number of data type.

* Example of Overriding

```
Class Bank {
   int getRateofIntrest ()
        {return 0;}
}
Class SBI extends Bank {
   int getRateofIntrest ()
        {return 8;}
}
Class ICIC extends Bank {
   int getRate of Intrest ()
        {return 9;}
}
Class Test 2 {
public static void main (String args[]){
   SBI s = new SBI ();
  ICIC i = new ICIC ();
System.out.println ("SBI RoI:"+
                        s.getRateofIntrest());
System.out.println ("ICIC RoI:"+
                        i.getRate of Intrest());
  }
}
```

-> Output:

```
SBI RoI : 8
ICIC RoI : 7
```

Example of overloading : (changing datatype)

```
class Adder {
static int add (int a, int b) {
        return a+b ;
    }
static double add (double a, double b) {
        return a+b ;
    }
}

class TestOverloding2 {
public static void main (String [] args){
 System.out.println (Adder.add (11, 11) ) ;
 System.out.println (Adder.add (12.3, 12.6)) ;
}
}
```

→ Output :

```
22
24.9
```

(2) Explain following terms.
(1) Abstract method (2) Command line arguments
(3) Ragged Array

→ (1) Abstract Method : A method which is declared as abstract and does not have implementation is knows as an abstract method.

-
```
abstract class Shape {
    abstract void draw();
}

class Rectangle extends Shape {
    void draw()
    System.out.println("drawing Rectangle");
    }
class Circle extends shape {
    void draw()
    System.out.println("drawing circle");
    }
}
```

-
```
class TestAbstraction {
public static void main(String args[]){
Shape s = new Circle();
  s.draw();
 }
}
```

→ Output

drawing circle

(2) Command line arguments : Command line arguments allow us to pass the arguments to the main method from command line.

- The main method of a Java program has the parameter args[] String type. This is bassically an array of String.

```
class CommandLineDemo {
    public static void main (String args[]){
    System.out.println (" Entered following data
                        using command line");
      for (int i = 0 ; i<args.length ; i++){
        System.out.println (args[i]);
      }
    }
}
```

```
D:\> Java CommandLineDemo Hello How Are You?
Enter following data using command line
Hello
How
Are
You?
```

→ (3) Ragged Array: More than 1-D array in which each dimension have different size is called Ragged Array.

```
public class RaggedArray {
 public static void main (String args[]){
 int A[][] = new int[3][];
 A[0] = new int[4] ;
 A[1] = new int[2] ;
 A[2] = new int[3] ;
 System.out.println(" Total rows :"+A.length);
 A[0][0] = 1 ;
 A[0][1] = 2 ;
 A[0][2] = 3 ;
 A[0][3] = 4 ;
2nd row → A[1][0] = 5 ;
 A[1][1] = 6 ;
3rd row → A[2][0] = 7 ;
 A[2][1] = 8 ;
 A[2][2] = 9 ;
 System.out.println("\n Array Representation:") ;
 for (int i = 0 ; i<A.length ; i++) {
     for(int j = 0 ; j< A[i].length ; j++){
     System.out.println (A[i][j] + " ") ;
     }
     System.out.println () ; }
 } }
```

output:

```
Total rows : 3

Array Representation:
1 2 3 4
5 6
7 8 9
```

(3) Explain passing Arguments by Value.

-> When we call a method, then we can invoke a a method by passing arguments by value.
-> When calling a method we need to provide the arguments in the same order in which they appear in method signature. This is called parameter order association.

```
class Test {
  public static void display (int r, String nm){
    System.out.println ("Roll no :" +r) ;
    System.out.println("Name :" +nm) ;
  }
  public static void main (String args[]){
    Scanner input = new Scanner (System.in) ;
    System.out.println ("Enter Roll no:") ;
    int rollno = input.nextInt() ;
    System.out.println("Enter Your name :") ;
    String name = input.next() ;
    display (rollno, name) ;
  }
}
```

Output:

```
Enter Roll no :
101
Enter Name :
AAA
Roll no: 101
Name : AAA
```

(4) Explain Arrays.

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

→ The elements of an array are stored in contiguous memory location.

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

→ Array is a index-based first element is stored at 0^{th} index and second element stored at 1^{th} index so on,

→ Advantage : It makes code optimized.
- We can retrcive or sort data effciently.
- We can get any data located at an index position.

→ Disadvantage : We can store only fixed size of element in the array.

→ Types : Single Dimensional Array
Multi Dimensional Array

→ Method : Declare, Print, Create an Array list, Convert Array list into arrays, Reverse array, Add element, remove element

Chapter : 4

Object and classes

(1) Explain constructor and give example of constructor overloading.

→ It is a special type of method which is used to initialize the object.

→ It is called when an instance of the class is created. At the time of calling constructor memory for the object is allocated in the memory.

→ Every time object is created using the [new()] keyword, at least one constructor is called.

→ It is call default constructor if there is no constructor avalaible in the class.

→ Rules : Constructor name must be same as class name.
- A constructor must have no explicit return type.
- A Java constructor can't be abstract, static and final.
- It must be declare in public mode.
- It should not have any return type.
- It can have default arguments.
- It can not be virtual.

```
Class student {
 int ID;
String Name;
 int age;

 student (int i, String n) {
    ID = i ;
    Name = n ; }

 Student (int i, String n, int a) {
    ID = i ;
    Name = n ;
    age = a ; }
  Void display () {
  System.out.println(ID+" "+Name+" "+age); }

 public static void main (String args[]) {
  student s1 = new student (111, "karan");
  student s2 = new student(121, "Rahul", 25);
  s1.display ();
  s2. display () ; }
 }
```

output →

111	karan	0
121	Rahul	25

(2) Using classes from Java library.

→ Java library which is also called as Java API is collection of rich set of Java classes that are used for devloping programs.

* Date class

→ The date class is defined in the pacakage java.util

→ It is used to represent date and time in Java.

→ Constructor of date class,

Date() → Create an object that represent current date and time.

Date(long milisecond) → Creates an object that give miliseconds since January 1, 1970.

→ Method :

getTime() → Return number of ms since January 1, 1970.

setTime() → Set new elaps time in the object.

toString() → Return string Representating date & time.

```
class Demo {
 public static void main (String args[]) {
    Date date = new Date();
    System.out.println (" Date is :" +date);
 }}
```

output: Date is : Tue April 26 01:20:31 IST 2022

* The Random class

-> It is used to genrate random numbers.
-> It belongs to util library of Java API.
-> Constructor:

Random() -> creates new random number genrator.

-> Methods:

nextInt() -> Return random integer value.
nextInt(n) -> Return random integer value 0 to n.
nextLong() -> Return a random long value.
nextDouble() -> Return a random double value.
nextFloat() -> Return a random float value.
nextBoolean() -> Return a random boolean value.

```
class Random {
  public static void main (String args[]){
    Random r = new Random();
    int value = r.nextInt();
    System.out.println("Random value: "+ value);
  }
}
```

```
Random value: 211528346
```

(3) Explain Access modifiers. or
Explain ~~Visibilty~~ visibility modifiers.

-> Access modifiers are control access to data field, method and classes. There are three modifier use in Java.
public, private, default modifier, protected

-> pubic allows classes, method and data field accesible from any class.
-> private allows classes, method and data field accesible only from within the own class.
-> default allows classes, method and data field accesible from any class in the same package. This is called package-private or package-access.
-> protected allows classes, method and data field ~~of~~ from all classes and subclasses present in same package. as well as subclasses from anothe package.

SPecifier	class	Subclass	Package
Private	✓	–	–
Public	✓	✓	✓
defaut	✓	✓	✓
Protected	✓	✓	✓

-> If some variable declared as protected, then the class it self can access it, it's subclass can access it and any class in the same package also access it.

```
package Test:

public class class1 {
 public int a;
  int b;
 private int c;
 public void fun1() { }
  Void fun2() { }
 private void fun3() { }
}
public class class2 {
 void My-method() {
 class1 obj = new class1();
 obj.a; // ✓
 obj.b; // ✓
 obj.c; // x
 obj.fun1(); // ✓
 obj.fun2(); // ✓
 obj.fun3(); // x
```

```
package another-Test

public class class3 {
 void My-method() {
 class1 obj = new class1();
 obj.a; // ✓
 obj.b; // x
 obj.c; // x
 obj.fun1(); // ✓
 obj.fun2(); // x
 obj.fun3(); // x
  }
}
```

→ Explain the program.

There are 3 data field : a, b, c
and 3 function : fun1(), fun2(), fun3().

a - public, b - default, c - private
fun1() -> public, fun2() -> defaut, fun3() -> private.

→ a and fun1() both accesible from any classes.
→ b and fun2() both accesible from any classes in same package.
→ c and fun3() both can't accesible from any classes because it is declare as private.

(4) Explain Immutable object.

→ The immutable classes are created to create immutable objects. The content of immutable object can't be changed.

→ Final keyword is used to create immutable object.

→ Rules : Class name must be declared as final.
- Data member in the class must be declared as final.
- Parameterized constructor are used.

```
- final class Employee {
  public final int emp-id ;
  public final String emp-name ;
  public Employee (int id, String name) {
    emp-id = id ;
    emp-name = name ; }
  public void display-date () {
  System.out.println (" Emp ID : " + emp-id ) ;
  System.out.println (" Emp Name : " + emp-name) ; }
- class EmployeeInfo {
  public static void main (String args[ ]) {
    Employee obj = new Employee (101, "AAA") ;
    obj.display data ( ) ;
    obj.emp-id = 100 ; // This line cause error
  }
}
```

→ As the class and class members are made final, we can not change its content using object.

(5) Explain "this" Refrence.

→ The this keyword is used by a method to refer the object which invoked that method.

→ This keyword is used inside the method definition to refer the current object. That means this is a reference to the object which is used to invoke it.

```
public class Demo {
  int a, b;
- public void Sum(int a, int b){
    this.a = a;
    this.b = b; }
- public void display() {
    int c = a+b;
    System.out.println("a = " +a);
    System.out.println("b = " +b);
    System.out.println("Sum = " +c);
    System.out.println(); }
- public static void main(String args[]){
    Demo obj1 = new Demo();
    Demo obj2 = new Demo();
      obj1.Sum(10, 20);
      obj1.display();
      obj2.Sum(40, 50);
      obj2.display();
  } }
```

//new line

a = 10	a = 40
b = 20	b = 50
Sum = 30	Sum = 90

Ch-5

Object Oriented Thinking

(1) Explain class abstraction and Encapsulation.

→ Abstraction : Hiding internal detail and showing functionality is called abstraction.

→ Encapsulation : Binding code and data into single unit is called encapsulation.

→ Each class is a collection of data and the function that manipulate the data.

→ The data components of class are called data fields and function components are called member function or method.

→ Object is an instance of class. The objects represent the real world entity. The object are used to provide a practical bases for real world.

class name
Data field
Method

→ Following is a list of object.

- For class vehicle various objects are - Alto, ZEN, SANTRO and so on.
- For windows OS System - the windows, icons, menues.

→ Encapsulation means the detailed implementation of component which can be hidden from rest of the system.

→ The data inside the class is accessible by the function in the same class. It is normally not accessible from outside component.

(2) Advantage of OOP.

-> Using inheritance the reudant code can be eliminated and existing classes can be used.

-> The standard working modules can be created using OOP.

-> Due to data hiding property, important data kept away from unauthorized access.

-> It is possible to create multiple object for given class.

-> For upgrading a system from small scale to large scale is possible due to OOP feature.

-> Due to data centered nature of OOP most of the details of application model can be captured.

-> Message passing technique in OOP allows the object to communicate external system.

-> partitioning the code for simplicity, understanding and debugging is possible due to object oriented approach.

(3) Explain Primitive data types and wrapper class types.

→ Wrapper classes are those classes that primitive data type can be accessed as object.

→ The wrapper class is one kind of wrapper around a primitive data type.
The wrapper class represent primitve data types in its intance of the class.

Primitive data	wrapper class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short
void	Void

* Method :

→ Integer val = new Integer (int-var) → An object is created for the integer variable int-var.

→ Float val = new Float (f-var) → An object is created for the float variable f-var.

→ Double val = new Double (d-var) → An object is created for double variable d-var.

→ Long val = new Long (Long-var) → An object is created for Long variable long-var.

→ Suppose an object for holding and integer value is created then we can retrieve integer value from it using typvalue() method For instance the object obj contains an integer value then we can obtain the integer value from [obj.].

```
int num = obj.intValue();
```

Similarly we can use for float, double, long.

→ We can convert integer value to string using toString() method.

```
Str = Integer.toString(int-value);
```

→ For converting String to numerical value the parseInt and parseLong method can be used.

→ The wrapper class do not contain constructor.

→ Methods of the wrapper class are static.

→ After assigning value to wrapper class we can't change them.

```
class Wrapper {
public static void main (String args[])
  {
  System.out.println ("Creating an object");
  Integer i = new Integer (10);
  System.out.println ("Obtaining integer value
                       from object:" +i.intValue());
  String str = "100";
  System.out.println ("Obtaining integer value
                  from string:" +Integer.parseInt(str));
  }
}
```

```
Creating an object
Obtaining Integer value from object : 10
Obtaining Integer value from string : 100
```

→ Uses :

- Used to convert string value into numeric value.
- Used to convert numeric value into string.
- It is a medium to store primitive data type into object.

(4) Explain String Builder and String Buffer class.

→ * String Builder

→ It provide us way to use mutable string but here we lack thread saftey. it can't be used by multiple threads simultaneously. class is not applying this so String buffer becomes faster than it.

Syntax : StringBuilder a = new StringBuilder ("Hey");

* String Buffer

→ It provides us a way to use mutable string in Java. These strings are safe to be used by multiple thread simultaneously. It consume less time.

Syntax : StringBuffer b = new StringBuffer ("Hello");

* Constructor :

→ StringBuilder ( ) → Create empty String with initial
StringBuffer ( ) capacity of 16.

→ StringBulider (String str) → Create an StringBuilder
String Buffer (String str) with specified String

→ StringBuilder (int length) → create an empty string with
StringBuffer (int length) capacity as mentioned

* Method

-> append (String str) -> Appends the String to the buffer.

-> capacity () -> It returns capacity of String.

-> charAt (int index) -> It returns specified character at given index.

-> delete (int start, int end) -> It deletes string specified by given index.

-> insert (int offset, char ch) -> It insert character at the position specified by offset.

-> length () -> It returns length of String.

-> setcharAt (int index, char ch) -> Set the character ch at given index.

-> SetLength (int new_len) -> It sets length of String.

-> toString () -> It convert string Representing data into String.

-> replace (int start, int end, String str) -> It replace the character specified by index.

-> reverse() -> The character sequence is reversed.

String Buffer	String Builder
Created in heap memory.	Created in heap memory
Provides us mutable strings	Provides us mutable String
default Capacity is 16 char.	Defaut capacity is 16 char.
Safe to be used by multiple thread.	Unsafe to be used by multiple thread.
Slower than String Bulider class.	Faster than String Buffer class.
Introduced in JDK 1.0	Introduced in JDK 1.5
It is synchronized	It is not synchronized.

```
class Demo {
public static void main (String args[]) {
 String Builder str = new StringBuilder("Autopilot");
System.out.println("Before Replace:" +str);
 str.replace(4,9,"mobile");
System.out.println("After Replace:" +str);
} }
```

```
Before replace : Autopilot
After replace : Automobile
```

(5) Explain Inheritance with types.

→ Inheritance is a mechnism in Java by which derived class can borrow the properties of base class and at the same time derived class may have some additional properties.

→ Advantage :

- Reusability : Parant class code can be used by derived class without any need to rewrite the code.
- Extensibility : Parent class logic can be extend in the derive class.
- Data hidding : Parent class can decide to keep some data private. So it can't used by derived class.
- Overriding : With inheritance we will able to override the method of parent class. so that meaningful implementation of base class method can be designed in the derived classes.
- With the help of inheritance we can minimize the ammount of duplicate code.

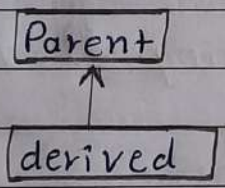
→ Parent class is also known as Super class.

→ Derived class is also known as Sub class.

```
class A                          ← Super class
{
  ..... }
class B extends A {              ← Sub class
  ..... }
```

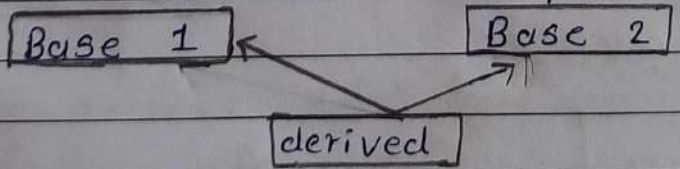
* Types :

(i) Single Inheritance : In this there is one parant per derived class. This is most common form of Inheritance.

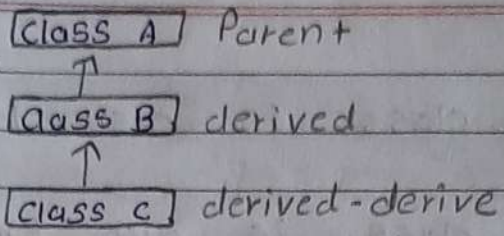


(ii) Multiple Inheritance : In this derived class is derive from more than one parent class.

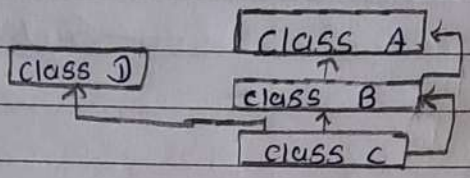
Java does not implement multiple Inheritance directly it can use concept of Interface.



(iii) Multilevel Inheritance : When class is derived from parent class which it self is derived a class then it is called multilevel Inheritance.



(iv) Hybrid Inheritance : When two or more types of Inheritance combined it is called hybrid Inheritance.




```
class Animal {
void eat() {
  System.out.println("eating...");}
class Dog extends Animal {
void bark() {
System.out.println("barking...");}
}
class Inheritance {
public static void main(String args[]){
   Dog d = new Dog
    d.bark();
    d.eat()  }
  }
```

```
barking...
eating...
```

(6) Polymorphism in Java.

→ It is a concept by which we can perform a single action in different ways.

→ It is derived from two Greek words: poly and morphs.
Poly means many and morphs means forms.

→ There are two type of polymorphism in Java:
(i) Compile-time polymorphism
(ii) Runtime polymorphism

→ We can perform polymorphism in Java by method of overloading and overriding.

→ If you overload static method in Java, it is the example of compile-time polymorphism.

* Run-time polymorphism

→ Run-time polymorphism and dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time.

→ In this process an overriden method is called through the reference variable of superclass.

→ If the refrence variable of parent class refers to the object of child class it is known as upcasting.

Ex.

```
class A { }
class B extends A { }

A a = new B ( ) ; // upcasting
```

```
class Bike {
  void run() {
   System.out.println ("running ....") ;
  }

class splendor extends Bike {
  void run() {
   System.out.println ("running safely") ;
  }

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

running safely

(7) Explain FINAL keyword. (final)

→ The final keyword in Java is used to restrict the user. The Java final keyword can be used in many context.
Final can be : variable, method, class

→ The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable.
It can be initialized in constructor only.

→ The blank variable can be static also which can be initialized in static block only.

(i) Java final Variable

→ If you make any variable as final, you can't change the value of final variable.

```
class Bike{
 final int speed = 50;
   void run(){
     speed = 80; }
public static void main(String args[]){
 Bike obj = new Bike();
 obj.run();
}
}
```

Compile Time error

(ii) Java final method

-> If you make any method as private you can't override it.

```
class Bike {
 final void run() {
   System.out.println("running");}
class Honda extends Bike {
   void run() {
    System.out.println("Walking");}
public static void main(String args[]){
    Honda h = new Honda();
    h.run();
   }}
```

compile-time error

(iii) Java final class

-> If you make any class final you can't extend it.

```
final class Bike {}
class Maruti extends Bike {
  void run() {
    System.out.println("running");}
public static void main(String args[]){
  Maruti M = new Maruti();
   M.run();
 }}
```

Compile-time error

(8) Method Overloading vs Method overriding.

Overloading	Overriding
→ Occurs at compile time.	Occurs at run time.
→ In this different number of parameter can be passed to function.	In this same number of parameter can be passed to the function.
→ It may have different return types.	It will have same return type.
→ It is performed within a class.	It is performed between two classes

Ch - 6

Exception Handling, I/o

Abstract classes and Interfaces

(1) Explain Exception.

→ In Java exception is an indication of some unusual event. Usually it indicates the error. Exception handled is using five keyword. try, catch, throws, throw, finally.

→ The Java code that you may think produce exception is placed within try block.

→ Let's see example of try and catch in order to handle the exceptio divided by zero.

```
class Exception {
  public static void main (String args[]){
  try{
    int a, b;
     a = 5;
     b = a/0; }
  catch (ArithmeticException e){
  System.out.println (" Divided by zero");
    }
}
```

Divided by zero

-> Inside try block the statment b = a/0 executed and then raised arithmetic exception, this exception is caught by catch block.

-> There must be try-catch pair and catch block should be immediate follower by try statment.

* Benifits :

- Using this the main application logic can be seprated out from the code which may cause some unusal condition.
- When calling method encounters some error then the exception can be thrown. This avoid crashing of app.
- The working code and error handling code can be seprated out due to exception handling.
- Using this various types of error in the source code can be grouped together.

(Q) Explain Error.

→ The exception heirarchy is derived from base class throwable. The throwable class is further divided into two classes exception and errors.

→ When any kind of serious problem occurs which could ~~not be~~ not be handled easily like OutOfMemoryError then an error is thrown.

- There are two types of error
  (i) Compile-time error
  (ii) Run-time error

(i) Compile-time error : The error detected by the Java ~~that~~ compiler during the compile time is called.

- When compiler issuses the errors it will not genrate the class file.

- Ex. : Missing semicolons.
→ Wrong spelling of keyword.
→ Missing brackets of class and methods.
→ Missing double quotes in String.
→ Use of undeclared variable
→ Use of = insted of ==
→ Illegal refrence to the object.

(ii) Run-time error : Sometimes program is free from compile time errors and created a class. But it does not show the result as per our expectation it is called Run-time error.

→ The run time error are basically the logical errors that caused due to wrong logic.

- Ex. : An expression divided by zero.

→ Accessing the array element which is out of index.

→ Trying store incompatible type value into Array.

→ Passing the parameter that is not in valid range.

→ Converting invalid string to number.

→ Trying to illegally change the State of thread.

→ Trying to access character which is out of bounds of a string.

(3) Explain try - catch Block.

-> The statment are likely to cause an ~~error~~ exception are enclosed within try block. For this statment the exception is thrown.

-> There is another block defined by catch which is responsible for handling the exception thrown by try block.

-> As soon as exception occurs it is handled by catch block.

-> The catch block is added immediatley after the try block.

-> If any statment in try block genrate exception then remaining ~~expression~~ statment skipped and control transfer to catch block.

* Syntax.

```
try
{
  // exception gets genrated here
}
catch (Type-of-exception e)
{
  // excetion is handled here
}
```

-> Programming example same as Que. (1).

(4) Explain throws, throw and finally.

--> * throws

-> When a method wants to thrown an exception then keyword throws is used.

```
class Exception {
static void fun (int a, int b) throws
                    ArithmeticException {
  int c;
  try {
   c = a/b ; }
  catch (ArithmeticException e){
  System.out.println ("Caught Exception"); }
}
public static void main (String args[]){
  int a = 5;
  fun (a,0); }
}
```

Caught Exception

-> This method should be static type. Also this method is responsible for handling the exception the try-catch block should be within fun.

* throw

-> for explicity throwing the expression exception, the keyword throw is used. The keyword throw is normally used within the method.
We can not throw multiple exception using throw.

```
class Exception {
  static void fun (int a, int b)
    {
      int c
    if (b==0)
        throw new ArithmeticException("Caught")
    else
      c = a/b;
    }
  public static void main (String args[]){
      int a=5;
      fun(a,0);
    }
}
```

Caught

* finally

-> It is used to execute important code such as closing the connection etc.

-> It is always executes wheather an exception is handled or not.

-> It contains all necessary statment that need to be printed when exceptions occurs or not.

-> finally block follows try-catch block.

```
class finally {
public static void main (String args[]){
  try {
  int a = 2 ;
  System.out.println(a); }
  catch (NullPointerException e){
  System.out.println(e); }
  finally {
  System.out.println ("Running finally"); }
  }
```

```
2
Running finally
```

Throw	Throws
It is used for explicity throwing the exception	It is used for declaring exception
Throw is followed by instance.	Throws is followed by exception class.
Throw is used within the method.	Throw is used with method signature.
We can't throw multiple exception	We can declare multiple exception.

(5) Defining Custom Exception classes.

→ We can throw our own exception using throw keyword.
→ Syntax : throw new MyOwnException();
→ class should be extend to exception.

```
class MyOwnException extends Exception {
  MyOwnException (String msg) {
     super(msg); }
  }
class Demo {
  public static void main (String args[]){
    int age = 15;
    try {
    if (age<21)
    throw new MyOwnException ("age less than
                              the condition"); }
    catch (MyOwnException e){
    System.out.println (e.getMessage());  }
   finally {
    System.out.println ("Finally block"); }
  }
}
```

```
age is less than the condition
```

(6) Explain Abstract classes.

-> A class which is declared with the keyword abstract is kown as abstract class.
-> It can have abstract and non-abstract method.
-> It needs to extended that its method implemented.

-> An abstract class must be declare with abstract keyword.
-> It can have abstract and non-abstract method.
-> It can't be instantiated.
-> It can have constructor and static method.

```
abstract class Bike{
  abstract void run();}
class Honda extends Bike{
  void run(){
  System.out.println("Running...");}
public static void main(String args[])
  {
  Bike obj = new Honda();
    obj.run();
  }}
```

Running.

(7) Explain Interfaces in Java.

→ An Interface in Java is a blueprint of class. It has static constants and abstract method.

→ The Interface in Java is a mechanism to achieve abstract and multiple Inheritance.

→ There can be only abstract method in Java, not method body.

→ Java Interface represent the IS-A relationship.

→ It can't be instantied just like abstract cla

→ By Interface, we can support the functionality of multiple Inheritance.

→ It can be used to achieve loose coupling.

```
interface printable{ void print(); }
class A6 implements printable{
pubic void print() {
 System.out.println ("Hello"); }
public static void main (String args[]){
  A6 obj = new A6();
  obj.print();
} }
```

Hello

(8) Difference between Class and Interface.

Class	Interface
→ Class can only inherit only one abstract class.	This class can Implement more than one Interface
→ Can have access modifier such as public, private, protected.	~~Can~~ It is public by default.
→ Method in abstract class may or may not have implementation.	Method in ~~Inherit~~ Interface have no implementation. Only declaration of method given
→ Java abstract classes are comparitively efficient.	The interface comparitively slow and implies extra level of ~~implements~~ indirection.
→ Java abstract class can be extend using keyword extends.	Java Interface can be implement using keyword implements.
→ Variable of abstract class are non final.	Variable of Interface are final default.

Chapter : 7 and 8

(1) JavaFx Layout.

→ Layouts are top level container classes that define UI style for scene graph object.

→ Layout can be seen as the parent node to all the other nodes.
JavaFx provides different layout panes that support different style and layout.

→ We have serval built-In layout panes in JavaFx that are HBox, VBox, StackPane, FlowBox, etc. Each built in layout can be represented with seprate cluss.

→ All these Java classes belong to javafx.scene.layout package.

→ javafx.scene.layout.pane is a base class for all built-in layout classes In Javafx.

→ BorderPane : Organizes nodes in top, left, right, center and bottom of the screen.

```
BorderPane root = new BorderPane();
```

Method : setBottom, setCenter, setLeft, setRight, setTop

→ FlowPane : Orgnizes nodes in the horizontal rows according to the avalaible horizontal spaces.
Wrap the nodes to the next line if horizontal space is less.

```
FlowPane root = new FlowPane ();
```

→ GridPane : Organizes node in the form of rows and colums.

```
GridPane root = new GridPane ();
```

→ HBox : Organizes the nodes in single row.

```
HBox row = new HBox ();
```

→ StackPane : Organizes nodes in the form of a stack.

```
StackPane abc = new StackPane ();
```

→ VBox : Organizes node in verticle column.

```
VBox anew = new VBox ();
```

* Step to create layout

(i) Instantiate the respective layout class,

```
HBox root = new HBox ();
```

(ii) Setting the properties for layout.

```
root.setSpacing (20);
```

(iii) Adding nodes to the layout object.

```
root.getChildren ().addAll (<NodeObjects>);
```

(2) JavaFx UI Controls.

→ The graphical user interface of every desktop application mainly consider UI elements, layout and behaviour.

→ The UI elements are the one which are actually shown to the user for interaction or information exchange.

→ Layout defines the organization of the UI element on the screen.
Behaviour is the reaction of UI element when some event is occurred on it.

→ The package javafx.scene.control provides all the nesessary classes for the UI components like Button, Lable, etc.

→ Lable : It is used to define a simple text on the screen. Typically it is placed with node. it describes

```
Lable L = new Lable ("Hello");
```

→ Button : It is used to control the function of application. Button class is used to create a lable button.

```
Button btn = new Button ("Click here");
```

→ RadioButton : It is used to provide various option to the user. The user can only choose one option among all. A RadioButton is either selected or deselected.

```
RadioButton rb1 = new RadioButton ("red");
```

→ CheckBox : CheckBox is used to get the kind of information from the user which contains various choices. User marked checkBox either on (true) or off (false).

```
CheckBox ch = new CheckBox ("Second");
```

→ TextField : TextField is basically used to get the input from user in the form of text. javafx.scene.control.TextField represent TextField.

```
TextField tf = new TextField ("Enter name");
```

→ PasswordField : PasswordField is used to get user's password. Whatever type in password it is not shown on the screen to anyone.

→ Hyperlink : It is used to refer any of the webpage through your application.

```
Hyperlink hp = new Hyperlink ("...");
```

→ Slider : It is used to provide pane of option to the user in graphical form where the user needs to move a slider over the range of values select one of them.

```
Slider sl = new Slider (1, 10, 20);
```

→ ProgressBar : It is used to show the work progress to the user. It is represented by class javafx.scene.control.progressBar

```
ProgressBar pb = new ProgressBar();
```

→ ScrollBar : It is used to provide scrollBar to the user so that user can scroll down the application pages.

```
ScrollBar s = new ScrollBar();
```

(3) Explain mouse and key event.

-> Mouse event is fired when the mouse button is pressed, released, clicked, moved or ragged on the node.

-> Mouse Pressed : setOnMousePressed (EventHandler <MouseEvent>)

-> Mouse Released : setOnMouseReleased (EventHandler <MouseEvent>)

-> Mouse clicked : setOnMouseClicked (EventHandler <MouseEvent>)

-> Mouse Moved : setOnMouseMoved (EventHandler <MouseEvent>)

-> Mouse Dragged : setOnMouseDragged (EventHandler <MouseEvent>)

-> Mouse Entered : setOnMouseEntered (EventHandler <MouseEvent>)

-> Mouse Exited : setOnMouseExited (EventHandler <MouseEvent>)

* Keyboard :

→ When any key on the keyboard is pressed, released, or typed on a node then the keyboard event occure.

→ Key Pressed : setOnKeyPressed (EventHandler <KeyEvent>)

→ Key Released : setOnKeyReleased (EventHandler <KeyEvent>)

→ Key typed : setOnKeyTyped (EventHandler <KeyEvent>)

→ For Recognized keyboard event we use KeyEvent class.

Constant	Associated key
HOME	The home key
END	The end key
CONTROL	The control key
SHIFT	The Shift key
ESCAPE	The escape key
ENTER	The enter key
TAB	The tab key
0 to 9	The key from 0 to 9
A to Z	The letter from A to Z

(4) Explain color class and font class and Image class.

→ * Font class

→ Font class is used to describe name of the font, its size and weight.

→ The javafx.scene.text.Font class is used to create fonts.

Font(double size, String name, String famaily)

where,
size : represent size of the font
name : name of font.
famaily : Represent font famaily.

→ Commonly used method :

- font (name, FontWeight, size) is used to create a font of specific name, weight and size.
- getFamilies() : It returns name of font famaily
- getFontNames() : It returns full font name including famaily and weight.

* Color class

→ For use color class we must import the pacakage javafx.scene.paint.color.

→ In Java class we can color the object using setFill() method.

→ The color class makes use of the colors made from combination of RGB.
Ex. (255, 0, 0) represent red color.
(0, 0, 0) represent black color.
(255, 255, 255) represent white color.

→ There is a rgb() method of color class which accepts this red, green, blue values. It also accept one more parameter called alpha.

→ alpha represent opacity of color.

```
Color (double red, double green, double blue,
                    double opacity )
```

→ There are some standard color BLACK, BLUE, BROWN, CYAN, GOLD, GRAY

```
Rect.setFill(Color.RED);
```

→ brighter() → larger red, green, blue
darker() → smaller red, green, blue

* Image class

→ It is used to display images in JavaFx application. for this we need to impor. javafx.scence.image.ImageView classes.

→ For using Image we need to load an image using InputStream. Then instance of InputStream class can be passed to Image class an argument.

```
FileInputStream ab = new FileInputStream
                    ("J:\\image3.jpg");

Image img = new Image(ab);
ImageView imgv = new ImageView(img);
```

↳ After loading an image we can view the image by instantiating the ImageView class and passing the image to its constructor.

(5) Explain Inner class.

→ Inner classes are the nested classes. That means the classes that are defined inside other classes.

```
class Java-Outer-class{
    ....
class Java-Inner-class{
    ....
}
}
```

→ The outer class can Inherit as many member of inner class object wants.

→ If outer and Inner both are public then any other classes can create an instance of inner class.

→ The inner class object do not get instantiated with outer class object.

→ Outer class can call private method inside of inner class.

→ If Outer and Inner both class have same variable that it can be accessed like this,

Outerclassname.this.variable-name.

→ There are four types of Inner class.

(i) Static Member classes: It is defined as the Static member variable of another class.

- Static member of the outer class are visible to static inner class.
- Non-Static member of outer class are not avalaible in inner class.

(ii) Member Inner classes: This type of inner classes is non-static member of outer class.

(iii) Local Inner classes: It is defined within a Java code within just like local variable.

- local classes are never declared with an access specifier.
- local classes are completley hidden from outside world.

(iv) Anonymous Inner classes: It is local class without any name.

- It is one-shot class created exactly where needed.
- It can extend the class, it can implement interface or it can be declare in method argument
- It is created for following situation:
  - When class has very short body
  - Only one instance of class is needed.
  - class is used immediatley after defining it.

(6) Compare AWT and Swing.

→

AWT	Swing
→ It is an API to devlope GUI app	It is a part of Java foundation classes. Used to create various app.
→ The componenat of this are heavey weighted.	The components of this are light weighted.
→ Less functionality as compare to swing.	More functionality as compare to AWT
→ Execution time is more than swings.	Execution Time is less then AWT.
→ These are platform dependent.	These are platform Independent.
→ MVC pattern is not supported.	MVC pattern is Supported.
→ Provides less powerful componants.	Provides more powerful components.

(7) Compare List and Set Interface.

	List	Set
→	It is an indexed sequence.	It is an not-Index. Sequence.
→	It allows duplicate element.	It can not allow duplicate element.
→	Element by there position can be accessed.	Element acess by there position are not allowed.
→	Multiple null element can be Store.	Null element can be Stored only once.
→	Ex. Array List, Linked List.	Ex. Hash Set, Linked hash set.

Chapter : 9

Binary I/o, Recursion and Generics

(1) Explain Text and Binary I/o.

-> When the data which get stored in binary form then those files are called binary files.

-> The text I/o requires encoding and decoding for storing them to the file whereas the binary I/o does not require encoding and decoding.

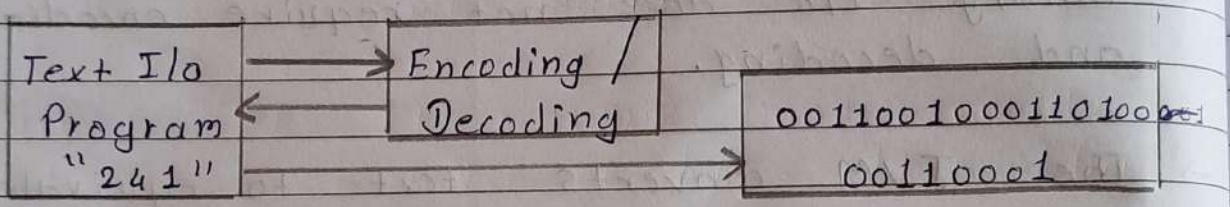
-> The JVM converts text to equivalent unicode. Then this unicode is written in binary form to the file.

-> While reading data from the file the unicode written in binary form is decoded into corresponding character. These character together give textual information.

Ex.

(i) Text Ilo Program : Consider that we want to store string "241" using text Ilo to a file.

- Only character type data is allowed.
- ASCII for 2 is 50. Unicode of 50 is 0x32. Binary form of 0x32 is 00110010.
- ASCII for 4 is 52. Unicode of 52 is 0x34. Binary form of 0x34 is 00110100.
- ASCII for 1 is 49. Unicode of 49 is 0x31. Binary form of 0x31 is 00110001.



(ii) Binary Ilo Program : Consider that we want to store string "241" Using Binary Ilo to file.

There is no need to decode or encode data.

- The 241 equals F1 in hex form. The binary form of F1 will be 11110001.

- Binary Ilo is more sufficient than text Ilo because it does not require encoding decoding. It can be execute on any machine.
- Various type of data is allow such as text, Image, Sound.

(2) Explain Binary I/o class.

* Input Stream is an abstract class for streaming the byte input. Various methods defined by this class as follows.

→ int avalaible() : It returns total numbers of byte input that are avalaible currently for reading.

→ void close() : Input source is closed by this method.

→ Void mark (int n) : This method places a mark in the input stream at the current point.

→ boolean markSupported() : It return true if mark() or reset() are supported by the invoking stream.

→ Void reset() : This method reset the Input pointer to the previously set mark.

→ int read() : It returns next available number of bytes to read from input stream.

* Output Stream as an abstract class for streaming the byte output. all this method are void type.

→ Void close() : It closes the output stream and if we write further after using this method it throws IO exception.

→ void flush() : This method clear ouput buffer.

→ void writen (int val) : This method allows to write a single byte to an output stream.

(3) Explain Problem solving using Recursion.

→ Recursion is a programming technique in which the method call it self repeatedly.
→ The parameter passed to a particular method allow to call itself. Such a method is called recursive method.
→ A most common example of recursion is use of factorial, which is usally used in mathamatics.

Ex. If the 5 factorial has to be calculated then, it will be = 5·4·3·2·1 = 120

→ Recursive definition of the factorial function is used to evalute 5!

Step 1:	5! = 5*4!
Step 2:	4! = 4*3!
step 3:	3! = 3*2!
step 4:	2! = 2*1!
step 5:	1! = 1*0!
step 6:	0! = 1

* Properties

→ The method is implemented using if-else or a switch statment that lead to different cases.
→ One or more base cases are used to stop recursion.
→ Every recursive call reduce the original problem, bringing it increasing closer to a base case until it becomes that case.

```
public class method {
  Static int factorial (int n) {
      if (n == 1) {
          return 1; }
      else {
      return (n*factorial(n-1)); }

  public static void main (String args[]){
System.out.println ("Factorial of 5 is:"
                    +factorial(5));
    }
}
```

Factorial of 5 is : 120

(4) Explain Tail Recursion.

→ Tail reursion is a kind of recursion in which there is no pending operation after returning from recursive function.

→ This is a special type of recursive approach which helps improving the space and time efficiency of recursive method.

→ Advantage : Optimizes the task for compiler for executing recursive function.
- Improves the space and time efficiency of recursive code.

```
Function A (){
.....
.....
call method A() recursively
}
```

Tail Recursion

```
Function B(){
.....
.....
call method B() recursively
.....
}
```

Non-Tail Recursion

(5) Explain Generics.

→ The Java Generics programming is introduce for deal with type-safe objects.

→ It makes code stable by detecting bugs at compile time.

→ Before generics we can store any type of objects in the collection.
Now generics force programmer to store specific type of object.

* Advantages:

(i) Type-safety: We can hold only a single type of object in generics. It doesn't allow to store other object.

(ii) Type Casting is not required: There is no need to typecast object.
Before generics we need to typecast.

(iii) Compile-time checking: It is checked at compile time so problem will not occure at runtime. Good programming Strategy say it is far better to handle problem at compile time than run time.

(6) Differentiate Iteration and Recursion.

Iteration	Recursion
-> Executing certain set of instruction repeatdly, without calling self function	Executing certain set of instruction repeatedly, ~~with~~ by calling self function.
-> These are implemented with the help of for, while, do-while	These are implemented with the help of calling the same function again and again.
-> These are more efficient because better execution speed.	These are less efficient
-> Memory utilization is less.	Memory utilization is more.
-> It is simple to implement.	It is complex to implement.

(7) What is FileInputStream and FileOutputStream.

→ * FileInputStream :

→ This class reads the data from a specific file. It is usually used to read the content of a file with raw bytes such as image.

→ First of all you need to instantiate this class by passing a string variable or a File object, representing path of the file to be read.

```
File file = new File ("D:\\ image.jpg");
FileInputStream inputStream = new FileInputStream(file);
```

* FileOutputStream :

→ This write data into specific file or file descriptor. It is usually used to write the content of a file with raw bytes.

→ First of all you need to instantiated this class by passing a string variable or a file object, representing path of the file to be read.

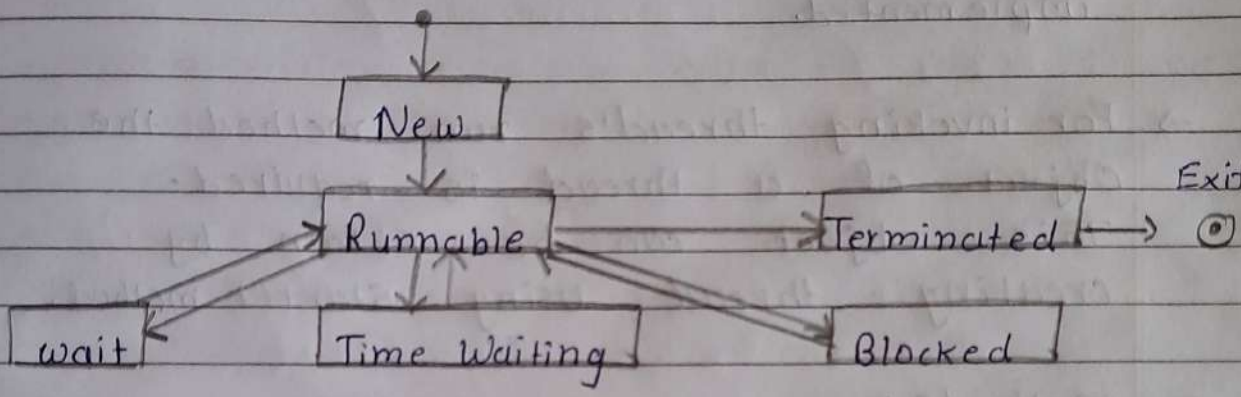
```
File file = new File ("D:\\ image.jpg")
FileOutputStream OutputStream = new
                                FileOutputStream (file);
```

Chapter: 12

Concurrency

(1) Explain Thread state and Life cycle.

→ Thread is a life cycle specifies how thread gets processed in Java program. by executing various method



→ New : When thread start life cycle it enters in the new state or create state.

→ Runnable : This is a state in which thread start executing.
- Waiting state.
- Sometimes one thread has to undergo in waiting state because another thread running.

→ Block : When a particular thread issues an Input/output request then OS sends it in block state until the Ilo operation gets completed. After Ilo completion the thread is sent back to the runnable state.

-> Terminated : After succesfull completion of the thread in runnable state enters terminated state.

-> The run() method is most important method in any program. By using this method thread's behaviour can be implemented.

-> For invoking thread's run method the object of a thread is required. This object can be obtain by creating thread using start() method.

* Methods :

-> start() : The thread can be start or invoke the run method.

-> run() : Once thread is started it executes in run method.

-> setName() : We can give name to thread using this method.

-> getName() : We can obtain thread name using this method.

-> join() : This methods waits for thread to end.

(2) Explain Thread Synchronization.

→ When two or more threads need to be access shared memory, then there is some way to ensure that the access to the resource will be by only on thread at a time.
The process of ensuring one access at a time by one thread is called synchronization.

→ Synchronization is a concept which is based on monitor. Monitor is used as mutually exclusive lock or mutex. When thread own this monitor at a time then the other threads can't access the resources. Other threads have to be there in waiting state.

→ In Java every object has implicit monitor associated with it. For entering in Objects monitor, the method associated with keyword synchronized.

→ There are two way to achieve synchronization
(i) Using synchronized method
(ii) Using synchronized block (statment).

(i) Using Synchronized method.

```
class table {
synchronized void display (int n){
 for (int i=1 ; i<=5 ; i++) {
  System.out.print (n*i) ;
  try {
   Thread.sleep (400) ;}
  catch (Exception e) {
   System.out.println (e) ; }
}}}

public class Test {
 public static void main (String args[]){
  final Table obj = new Table ;
  Thread t1 = new Thread () {
   public void run () {
    obj.display (5) ; }
  } ;
  Thread t2 = new Thread () {
   public void run () {
    obj.display (100) ; }
  } ;
   t1.start () ;
   t2.start () ;
}}
```

5	100
10	200
15	300
20	400
25	500

(ii) Using Synchronized block method.

```
Class Table {
void print(int n){
  synchronized (this) {
    for (int i=1 ; i<=5 ; i++) {
      System.out.println (n*i);
  try {
  Thread.sleep(400);
 }catch (Exception e){
   System.out.println(e); }
 } } } }
class Thread1 extends Thread{
  Table t;
    Thread1(Table t){
      this.t = t; }
  public void run(){
    t.print(5); } }
class Thred2 extends Thread {
  Table t;
  Thread2(Table t){
    this.t = t; }
  public void run(){
    t.print(100); } }
public class Block {
public static void main(String args[]){
  Table obj = new Table();
  Thread1 t1 = new Thread1(obj);
  Thread2 t2 = new Thread2(obj);
  t1.start();
  t2.start(); } }
```

→ Same output:

* Remember

→ Only methods can be synchronized but variables and classes can't synchronized.

→ Each object has one lock.

→ A class contain serval methods and all methods need not be synchronized.

→ If two threads in a class need to execute synchronized method and both the method are using the same instance of class then only one thread can access the method at a time.

→ We can not synchronized the constructor.

→ A Thread can aquire more than one lock.