CSA09-PROGRAMMING IN JAVA

INDEX

Introduction to Java

Java : Programming Environment

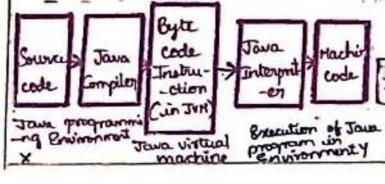
→ General - Purpose
→ Class - Based
→ Object- Oriented Programming language
→ Platform for Application development
→ Run time Environment
→ fast, secure & Reliable

Applications :-

* Laptops
* Data Centers
* Game Consoles
* Scientific Super Computers
* Cell phones.

Characteristics :-

* Object- Oriented
* Architecture neutral
* Portable
* Distributed
* Robust
* Secure
* High Performance
* Multi-threaded
* Dynamic
* Simple


Source code
Java Compiler
Byte code Instruction (in JVM)
Java Interpreter
Machine code
Java programming Environment
Java virtual machine
Execution of Java program in Environment
x

Keywords : predefined meaning in language

Eg:

Abstract	default	Package	this
boolean	do	Private	throw
break	double	Public	throws
byte	else	return	transient
Case	extends	short	try
Catch	final	static	void
char	finally	super	volatile
class	float	switch	while
Continue	for	Synchronized	Protected

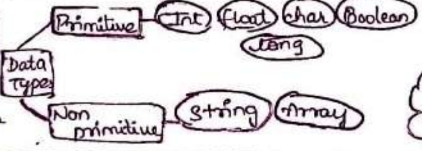
Data Types :

* Data types are different sizes
* values stored in the variable

Types of Data Types :

* primitive data types

	Data type	Default value	Default size
Boolean	boolean	false	1 - bit
Numeric	char	'\u000'	2- byte
	byte	0	1- byte
	short	0	2 - byte
	int	0	4- byte
	long	0L	8- byte
	float	0.0f	4- byte
	double	0.0d	8 - byte



(*) Non primitive data types
→ classes, Interfaces & Arrays

Variables :

* Basic unit of storage
* Reserved area allocated in memory
* Combination of an identifier
* optional initializer
* Visibility
* lifetime.

Declaring a Variable

Type identifier [= value] [identifier [= value] ...] ;

Types:

* Local variable
* Instance variable
* Static variable

1. Local Variable :

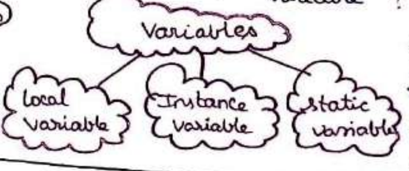
* Declaring inside the body of the method .
* Only with in the method
* Not Defined with "static" Keyword

2. Instance Variable :

* Declared inside the class but outside the body of the method
* Not declared as static
* Instance- specific
* Not shared among instances.

3. Static Variable :

* Cannot be local
* Single copy of the static variable
* Share it among all the instances
* class is loaded in memory



Arrays :

* Collection of similar type of elements
* Contiguous memory location
* Elements of a similar data type
* Stored in a Contiguous memory location.
* Store only a fixed set of elements

Advantages :

* Code optimization
* Random access

Types :

* Single dimensional Array
* Multi dimensional Array

* Single Dimensional Array :
* A special type of Variable
* Store multiple value single data type
* Int, float, double, char pointer, structures
* Stored in a Contiguous memory location

Initialisation of an Array :

```
arrayRef var = new datatype[size];
```

* Multi dimensional array :

→ data is stored - row & column
→ size index position of array
→ second index position of element.

Syntax :

```
int[][] arr = new int[3][3];
```

Application :-

* Scheduling concepts
* Sorting
* Searching

Operators, control, Statements

Operators:-

Special symbols Perform special operations:.

Types:-

- Arithmetic Operators
- Assignment operators
- comparison operators
- Logical operators
- Bitwise operators
- Increment/Decrement operators

Arithmetic Operators:-

Operator	Name	Example
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
%	Modules	x%y

Assignment operators:-

Operator	Example	
=	x = 5	x = 5
+=	x+ = 3	x = x+3
-=	x- = 3	x = x-3
*=	x* = 5	x = x*5
/=	x /= 5	x = x/5
%=	x% = 5	x = x%5
&=	x& = 3	x = x&3
!=	x! = 3	x = x!=3
^=	x ^ = 3	x = x^3
>>=	x >> = 5	x = x>>5
<<=	x << = 5	x = x<<5

Comparison operators:-

Operator	Name	Example
==	Equal to	x == y
!=	Not equal to	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than (or) equal to	x <= y

Logical operators:-

Operator	Name	Example
&&	Logical and	x<5 && x<10
||	Logical or	x<5 !! x<4
!	Logical not	!(x<5 && x<10)

Bitwise operators:-

Operator	Name	Example:
&	Bitwise AND	x & y
^	Bitwise exclusive OR	x ^ y
|	Bitwise inclusive OR	x | y
~	Compliment	~x
<<	Left shift	x << y
>>	Right shift	x >> y

Increment / Decrement operator

Operator	Name	Example
++	Post increment	x++
	Pre increment	++x
--	Post Decrement	x--
	Pre Decrement	--x

Control Statements:-

Executed according order

Smooth flow of Program

Types:-

Decision Making Statements

- * If Statements
- * Switch Statements

Looping Statements

- * do while
- * while
- * for loop

Jump statements

- * Break statement
- * continue statement

Decision making Statements:-

If Statements:-

Evaluate a condition

Diverted specific condition

condition either True (or) false

Types:-

- * Simple If Statement
- * If- else statement
- * If - else - If ladder
- * Nested If - statement

Simple If Statement:

Expression evaluates to true

Syntax:-

```
If (Condition)
{
  Statement ;
}
```

If-else Statement:-

```
If ( condition)
{ statement 1;
}
else
{ statement 2;
}
```

Nested If Statement

```
If (condition1) {
    Statement 1; }
If ( condition2) ; {
    statement 2 }
else { statement3;
} }
```

If - else - If ladder

```
If(condition1)
{
  Statement 1;
}
else If (condition2)
{
  Statement 2;
}
else
{
  Statement 3;
}
```

Switch Statement:-

Multiple blocks of code in a single case

```
switch ( expression)
{
   case value1:
    statement 1;
   } break ;
      :::
  case value n:
  statement n;
   break;
  default:
  } default statement ;
```

Looping Statement:-

Execute code repeatedly

Execution Instruction particular condition

Types:-

- * for loop
- * while loop
- * Do- while loop

for loop:-

```
for ( Initialization ; condition ; increment/Decrement)
  {
    block of statements
  }
```

while loop:-

```
while (condition) {
    Statement
  }
```

do-while

```
do { statement
}
while ( condition ;
```

Jump statement:-

Transfer - control specific statement

Execute other part of the Program

Types:-

- * Break :-> stop the current flow of the Program
- * Continue:-> Skips the specific part

Applications:-

- * Mathematical Calculation
- * Searching
- * Sorting
- * Handling Electronic and IoT Related operations

OOPs Concepts

OOPs:

Simplifies Software development

Realworld entity.

OOPs Concepts

* Object
* Class
* Encapsulation
* Inheritance
* polymorphism

Object
Class
OOPS
Encapsulation
polymorphism
Inheritance

Object

* State and behaviour
* Instance of a class
* Runtime entity
* Required memory space

Classname object = New Classname();

Class

* collection of objects
* Logical entity
* User defined datatype

```
class className
{
   DataType var1;
   Datatype var2;
       :
   Datatype varN;

   ReturnType methodName(par1, Par2)
   {
      method Implementation;
   }
}
```

'this' Keyword

used inside any method to refer current object.

Instance variable hiding

* When local variable and instance variables are same, Local variable hides instance variable.
* It resolves name collision.

Garbage Collection

Deallocates unused object spaces from memory

Inheritance

→ Properties and behaviours of parent class

→ code reusability

⇒ Acquire features

⇒ uses 'Extends' Keyword

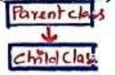
Types of Inheritance

* Single Inheritance
* Multilevel Inheritance
* Hierarchial Inheritance

Single Inheritance

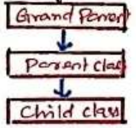
→ only one Super class with one subclass

→ Subclass inherits features from Super class



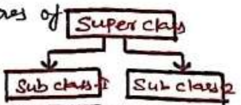
Multilevel Inheritance

* Chain of Inheritance



Hierarchical Inheritance

Morethan one subclass inherits features of super class



Encapsulation

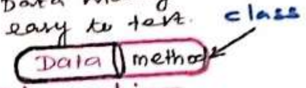
wrapping code and data in a single unit

Advantage:

control over the data

Data hiding

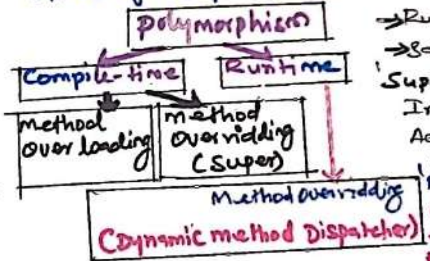
easy to test.



Polymorphism

many class related to other many forms perform different tasks.

Types of polymorphism



Example:

```
class Animal
{
   public void animalsound()
   {
     System.out.println("Animal sound");
   }
}
Class Dog extends Animal
{
   public void animalsound()
   {
   System.out.println("Dog sound");
   }
}
```

APPLICATIONS: Reusable plug-in component in s/w development processes.

method Overloading

→ Two(or) more methods shares same name.

→ Different parameter declaration

→ change the no of arguments

→ Change datatypes

Method Overridding

→ Same method signature defined in both super and sub class

→ Runtime polymorphism

→ Same name in parent class

'Super' Keyword

Invokes superclass constructor or Achieves method overridding during compile time polymorphism

'Final' keyword

* Constant variable
* Prevent inheritance
* prevent method overridding

'Abstract' class

Object can't instantiated

Constructor

* Special type of function holding class name
* has no returntype
* Invokes during object creation, automatically
* Initialize an object

Types

→ Default Constructor

→ Parameterized Constructor

Packages, interfaces and String Handling

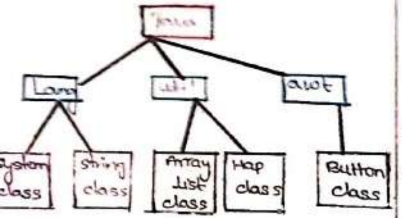
Packages :-

* Group of similar type of classes
* Interfaces and sub packages

Types :-

* Built in packages
* user - defined packages

Built in package :-



User defined packages :-

Categorised classes using customized Package.

Accessing a packages :-

Import package * ;
Import package class name ;
fully Qualified name

Advantages :-

easy Maintained * Access protection
Removes Naming collision.

Example :-

```
package my pack ;
public class simple {
public static void main (string args [ ]) {
system. out. print ln ("welcome pack");
}
 }
```

Access Modifiers :-

* No modifier * private
* protect * public.

Interfaces :-

* Blue print of a class
* provide static constants & abstract methods

Declare Interface :-

```
Interface < Interface - Name >
 {
 }
```

Example :-

```
Interface printable
{
 int min = 5 ;
 void print ( ) ;
 }
```

Implementing Interface :-

$$\begin{tikzcd}
\text{class} & \text{Interface} & \text{Interface} \\
\text{class} \arrow[u, "\text{extends}"] & \text{class} \arrow[u, "\text{Implements}"] & \text{Interface} \arrow[u, "\text{extends}"]
\end{tikzcd}$$

Example :-

```
Interface printable
{
void print ( ) }
class Ab Implements printable {
public void print ( ) {
system.out. print ln (" Hello "); }
public static void Main (string args[ ])
{
 Ab obj = new Ab ( )
 obj . print ( ) ;
 }
  }
```

String handling :-

String :- " sequence of " char " values.

```
char [ ] ch = {'a','e','i','o','u'};
string s = new string (ch) ;
```

Types of strings :-

* string - Immutable
* string buffer - mutable

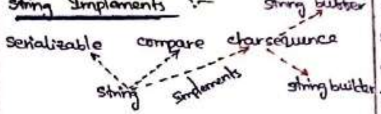
Operations of string :-

* Compare * concat * equals
* split * length * replace
* compare to ( ) * intern * substring.

Creating strings :-

```
String str = "abc"
char data [ ] = {'a','b','c'};
string str = new string (data) ;
```

String Implements :-



String Object Creation :-

* By string literal string s = " welcome"
* using key word

String s = new string ("welcome") ;

String Methods :-

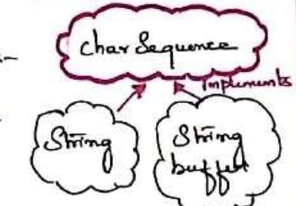
* char char At (int index)
* Int length ( )
* static string format (string format , objects ..... args)
* static string format (Locale I, string format , Objects..... args)
* string substring (int begin Index)
* boolean contains (char sequence less)
* static string join(char sequence delimiter , char sequence .... elements)
* Boolean equals (object another)
* Boolean is empty ( )
* string concat (string , str)
* string replace (char , old , char new)
* string split (string regex)
* string [ ] split (string regex , int limit)
* string Index of (string substring)
* string to lower case ( )
* string to upper case ( )
* string trim ( )
* static string value of (int value).

example :-

```
public class string example {
public static void main (string args [ ]) {
string s1 = " good "
char ch [ ] = new string {'s','t','r','i','n','g'};
string s2 = new string (" example ");
 system. out. println (s1) ;
 system. out. println (s2) ;
 system. out. print (s3) ;
 }}
```

Output :-

```
good
example
string
```



Application :-

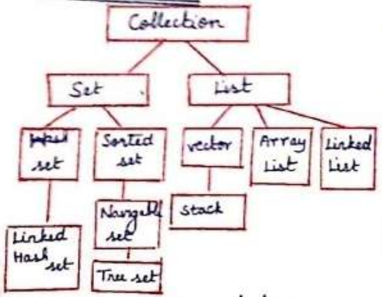
* Modularization of a applications.
* Searching keyword in Internet.

Java collection in Framework

Collection:

* single unit of object
* Store & Manipulate group of object
* Set of class & Interfaces

Collection overview:



Recent changes to collections

- Provides Stored object references
- Avoid Runtime type mismatch error

Set Interface:

* Cannot contain duplicate
* unsorted set
* Allow one null value

Implementation:

* Hash set
set < data - Type > S1 = new Hash set < data_Type>();
* Linked Hash set:
Set < data-Type > S2 = new Linked Hash set < data-Type>();
* Tree set:
Set<data_type> S3 = new Tree Set < data-Type> ();

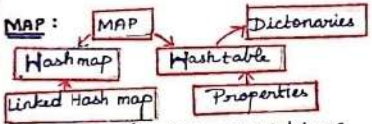
List Interface:

* store ordered collection
* Allows duplicate

Implementation:

* vector
List < data_Type > list 1 = new vector
* Stack
List<data-type> list 2 = new stack ()
* Array List
List<data-Type> List 3 = new Arraylist()
* Linked List
List<data-Type> List 4 = new linked list()

MAP:



* contains unique key
* Maintain no order
* key elements

→ search → updata → delete

Hash Table:

- array of list
- value based key
- unique element
- allow null key
- synchronized

Linked Hash Map:-

- Inherits Hash Map class
- Maintain insertion order.

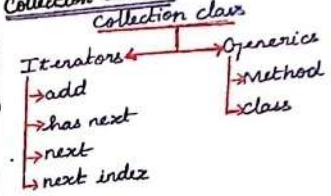
Methods:-

* void clear ()
* void rehash ()
* boolean contain & object values
* boolean contain key (object key)
* Buffer is empty ()
* object set (object key)
* object remove (object key)
* int size

Example:

```
Hash map <Integerstring >
hm = new Hash map <Integer string>
hm. Put ( 100, "good")
hm. Put ( 101, "better")
for (map-entry m = hm. entryset ())
```

Collection classes:-



Iterators:

- Obtaining or removing events
- Traverse only in forward direction

List Iterators:

* Bidirectional
* Modification of element

Method:

* boolean has next → Return True / False
* object next → Return next element
* Void element → Remove current element / → illegal state exception

Example:

```
Array< string> city names = new
        Arraylist<string> ()
city Names. add ("chennai")
city Names. add ("Delhi)
Iterator iterator = city Names. Iterator ();
system. out. Println ("citynames");
while (iterator. has next ())
system. out. Println (iterator. next () + " ");
system. out. Print ln ();
```

Generics:

Supports Generic Methods and Generic class.

Rules:

* Declare angle brackets
* Declare return Type
* Passed generic Method
* Represent any reference type

Types:

* Method
* Class

Methods:-

* single generic Method
* different types of arguments

Class:-

* Non-generic class declaration
* More type of Parameter.

```
class classname <T> {
    T Var;
    type method (T Var) {
        { ... }
}
```

Applications:-

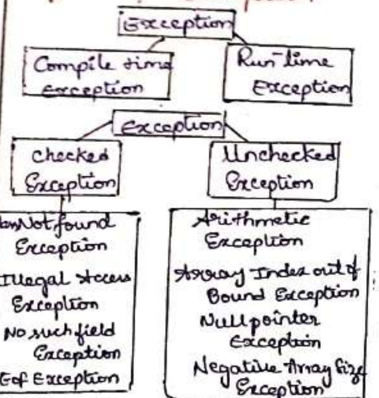
* Fetching and Manipulating data - database
* Bank transactions.
* Student database
* Pay roll System

Exception Handling and Multithreaded Programming

Exception
- Abnormal Condition
- Run time Error.

Exception Handling
* Handles runtime Error
* Maintains normal flow of Application

Types of Exception



Built in Type Exception
* predefined Exception
* All type of checked & unchecked Exception

Types of Error :-

Syntax Error
- Due to poor understanding of language

Logic Error
- poor understanding of problem.

five Keywords :- * try
* catch
* finally * Throws
* Throw

Try Catch finally paradigms

Syntax:

```
try
{ // statement Cause an
        Exception
}
catch
{ // Error handling code
}
finally
{ // code executed before
     try block
}
```

Examples :-

```
1. int a = 50/0 ; Arithmetic
                     Exception
2. String s = null;
   System.out.println(s.length());
   // null pointer Exception.
3. int a[] = new int[5];
   a[10] = 50 ; // Array Index
                  out of Bounds
```

Throw
Throw an exception Explicitly

Syntax Throw Throwable Instance

Throws
Declare an Exception

Syntax

type method() throws Exception name

Uncaught Exception
* occurs when an Exception not caught by program construct.

User Defined Exception
- create own Exception
- Throw an Exception.

Syntax

```
class MyException extends
          Exception
{ Public MyException(String s)
  { Super (s); //call parent
               Exception
  }
}
```

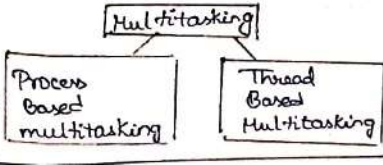
Multithreaded Programming
- Executing multiple threads simultaneously

Thread
- light weight Process
- Smallest unit of Processing

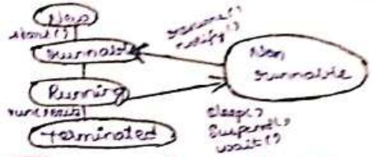
Advantages of Multithreading
- Threads are Independent
- Saves time

Applications of Multithreading
→ Games → Animations

Multitasking
* Executes more than one task simultaneously
* Two types.

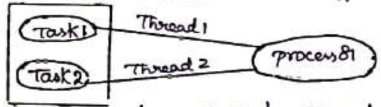


Life Cycle of Thread
1) New 2) Runnable 3) Running
4) Non runnable (Blocked) 5) Terminated



Two ways of Creating Thread
1) Implements the runnable interface
2) Extends the thread class

Creating multiple Thread
* perform multiple tasks
* Improves CPU performance
* Share same address space



Methods Supported by Thread

```
public void run()
public void start()
Public void sleep()
Public int get priority()
Public void set priority()
public void suspend()
public void resume()
public void stop()
```

Thread Priorities :

```
Public void setpriority (int level)
public int getPriority ()
```

APPlications :-

* Implementing network server and web server.
* Online transaction system.

Synchronization, Interthread communication and I/o classes

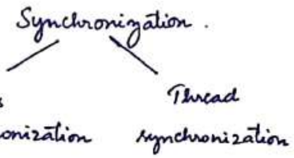
Synchronization

- control the accur of multiple threads.
- Access the shared resource.

Purpose.

- Prevent thread interference
- Prevent consistency problem.

Types.



Thread synchronization.

1. Mutual exclusive
2. co-operation (inter-Thread-communication)

Mutual Exclusive.

a) Synchronized Method
  - Lock and object
  - Any shared resource.

SYNTAX.

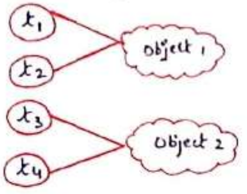
```
Synchronized type method()
{ ....
}
```

b) Synchronized Block.
- perform on specific resource

SYNTAX.

```
synchronized
{ ..
}
```

c) Static synchronization



Inter Thread communication.

- Thread paused running
- critical section
- Another Thread enter (or lock).

Methods.

* wait() * notify
* notifyAll

Java I/o classes.

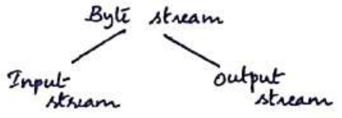
Stream

- Sequence of Data
- composed of bytes

System.out // standard output stream

System.in // standard input stream

System.err // standard error stream.



Byte stream classes.

Buffered Input stream
Buffered output stream
Data Input stream
Data Output stream
File Input stream
File Output stream
Input stream
output stream

Character stream classes

Buffered Reader
Buffered Writer
File Reader
File writer
Input stream Reader
output stream Reader
Print writer
Reader
Writer

File.

- creating File
- Rename a File
- copy the content
- Delete a file.

Methods.

Void write (byte [] arr)
void write (int b).
Void write (byte [] arr, int off, int len).
void close ().

Console class.

- Read Text and password

Methods.

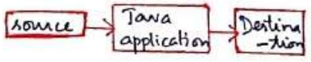
readline ()

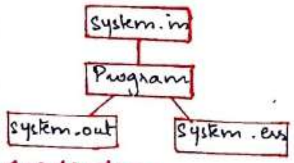
```
console c = system.console();
string n = c.read line();
```

readpassword ().

```
console c = system.console ();
Char [] ch = c.readpassword();
string pass = string.value of (ch);.
```

stream I/O





Serialization

- Writing state of object
- Convert to Byte stream
- Implement RMI

Deserialization.

- Restore serialized object

object Inputstream

Constructor

```
Public object Input stream(
        Input stream in)
    Throws IOException.
```

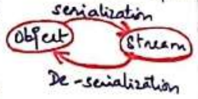
Object output stream

constructor

```
Public object output stream
(output stream out)
Throws IoException
```

Methods.

1. Public final void write object (object o) Throws IoException
2. Public void flush () Throws Io Exception
3. Public void close () Throws IoException



Stream Benefits.

- clean Abstraction
- Filtered stream class
- custom streaming interface.
- Java Input stream class
- Java output stream class.
- Reader class
- Writer class
- Network streams
- Socket streams.

APPlications:-

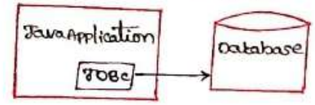
* Scheduling

* optimization problems.

Java Database connectivity

JDBC :-

- Java database connectivity
- Advancement for ODBC
- standard API specification
- Interbace or channel.



Why JDBC ?

- Open Database connectivity (platform dependent)
- Remove dependence (JDBC)

Simple JDBC Connectivity :-

$$\begin{tikzcd} \text{① Register Drive} \arrow[r] & \text{② net connection} \arrow[d] \\ \text{④ execute Query} \arrow[d] & \text{③ create statement} \arrow[l] \\ \text{⑤ close connection.} & \end{tikzcd}$$

Steps for connectivity between Java program and database :-

Loading the driver :-

class.forname ()

Load the driver class file

) Driver Manager . register Driver ( )

Register the backend driver

. Create the Connection :-

- Establishes the connection.

Connection con = Driver Manager . get connection (url, user, password)

3. Create a statement :-

- Interact with database

Statement st = con. create statement()

4. Exequte the Query :-

- Retreiving the data.
- updating / Inserting table.

(i) execute Query ( )

(ii) Execute update (sql.query)

5. Close the Connection :-

con. close ()

Two ways to load jar file

1. Paste mysql connector . Jar file in Jre / lib / ext folder.
2. set classpath

   a) temporary b) permanent.

Opensource databases :-

* MySQL * Microsoft SQL

* postgre SQL * Tetradata database

* MongoDB * SAP HANA, Express edition

* CouchDB * DynamoDB

JDBC statement types :-

* Statement * Prepared statement

* collable statement.

$$\begin{tikzcd} \boxed{\text{statement}} \arrow[d, "\text{extends}"'] & \text{- Provide method execute Queries} \\ \boxed{\text{Prepared statement}} \arrow[d, "\text{extends}"'] & \text{- Dynamically faster} \\ \boxed{\text{Collable statement}} \arrow[r] & \text{Dynamically faster store binary data} \end{tikzcd}$$

1. create statement :-

Statement st = Connection . create statement ( );

2. prepared statement :-

- Accept parameterized SQL Queries

string Query = " Insert into people name, age) values (?, ?)";

statement st = con, prepared statement (Query);

st. set string (1, Ayan);

st. set string (2, 25);

3. Collable statement :-

- stored procedure

collable statement st = con Prepare call (" { call procedure name (?, ?)")

JDBC Driver :-

- Implement the defined Interface.
- Interacting with database Server.

Types :-

Type 1 :- JDBC - ODBC bridge driver

Type 2 :- JDBC - Native API

Type 3 :- JDBC - Net Pure Java

Type 4 :- 100%. Pure Java.

Basic Insert, update & Delete :-

1. Insert, Into table (Attribute, Attributes) values (?, ?);

2. update Table set Field = value where Field, = value

3. Delete From Table where field = value

Example :-

```
// Loading Driver
class. forname ("com. mySQL. Jdbc driver");

// Get Connection
connection con = Driver Manager .
get connection ("jdbc : mySQL :
/// local host : 3306 / emp"."root"
POrt");

// create statement
statement st = con. create statement ();

// Execute the Query
a) string str = "create table EmpL
   eno, int not null , name varchar (10),
salary double (10, 2) , primary key(eno))
st. execute update (sql);

b) Result set rs = st. execute Query(
   "select * from emp")
while (rs . next ())
{
  sop (rs .get Int (1));
  sop (rs. get String (2));
  sop (rs. get Double (3));

// close the Connection
  con . close ();
```

Applications :-

* Social Media website

* email Systems.

* e-commerce websites.

Applete

Applet

* contained in Java. applet package
* Needs either browser (or) an applet viewer tool
* Runs on window, not by console based Java runtime interpreter.

Types of Applets:

1. Local Applet
2. Remote Applet

Methods:

1. Void destroy ()
2. Audioclip get Audio clip (URL url)
3. URL get CodeBase()
4. URL get Document Base()
5. Image get Image (URL url)
6. Void init ()
7. Void play (URL url)
8. Void resize (Dimension dim)
9. Void show status (String str)
10. Void start ()
11. Void stop ()

Basic of Applet.

* Apply execution not begin at main()
* drawstring() instead of system.out.println()

Applet Architecture:

1. Window Based program
2. Event driven
3. User initiates interaction.

Applet Skeleton

init ()
start()
stop ()
destroy ()
paint()

Applet Initialization

AWT calls the following methods

1. init ()
   First Method.
2. Start ()
   after init ()
3. paint ()
   applet's output.

Applet termination:

1. Stop()
   - leaves The HTML Document
   - restart using start()
2. Destroy ()
   - removed completely from memory

Simple Applet Display

Methods:

Void draw string
  (string message, int x, int y)

Background color and foreground color

Void set Background
  (Color new color)

Void set Foreground
  (color new color)

Color constants

1. Color. black
2. Color. dar Gray
3. color. light Gray.
4. Color. blue color. gray
5. Color. magenta
6. Color. cyan
7. Color. green
8. Color. organge
9. Color. pink
10. Color. red
11. Color. white
12. Color. yellow.

HTML Applet Tag

Applet Tag is used to start

Syntax:

```
<APPLET
[CODEBASE = codebase URL]
CODE = applet File
[ALT = alternate Text]
[Name = applet Instance Name]
WIDTH = Pixels  HEIGHT = Pixel
[ALIGN = alignment]
[VSPACE = Pixels]
[HSPACE = pixels]
>
[< PARAM NAME = Attribute NAME
   VALUE = Attribute Value>]
</APPLET>
```

CODEBASE: Base of URL
CODE: class file
ALT: Displayed in The Browser
NAME: Name of The applet
WIDTH & HEIGHT: Size of the applet
ALIGN: Alignment of the applet
VSPACE AND HSPACE:
  They are optinal.

Applicattons:-

* commerical website
* graphics and animation.
* games
* developing web pages

Event Handling and AWT

EVENT HANDLING

Delegation Event Model.

user interaction with GUI

Three important players.

1. Event source
2. Event Listener | handler.
3. Event object.

Event Source.

GUI component.

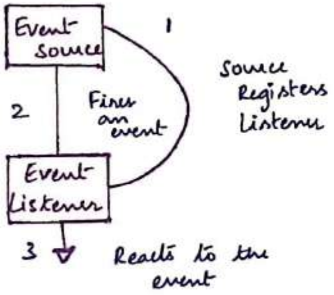
Event Listener | handler.

1. Receives events
2. Containing business logic

When an event occurs

1. An event object created
2. Event object fired.

Control flow.



Method of Event Source.

void add <Type> Listener (<Type> Listener listener obj)

Being unregistered

void remove <Type> Listener (<Type) Listener listener Obj)

Event classes.

Found in java.util package

1. Component Event.
2. Input Event
3. Action Event.
4. Item Event
5. Key Event
6. Mouse Event
7. Text Event.
8. Window Event.

Event Listeners.

1. Action Listener.
2. Mouse Listener.
3. Mouse Motion Listener.
4. Window Listener.

Action Listener Method.

Public void action performed. (Action Event e)

Mouse Listener Methods.

1. Public void mouse clicked (Mouse Events e).
2. Public void mouse Entered (Mouse events e)
3. Public void mouse Exited (mouse event e)
4. Public void mouse pressed (mouse event e)
5. Public void mouse released (mouse event e)

Window Listener Methods.

1. Public void window opened (windows Event e)
2. Public void window closing (window Event e)
3. Public void window closed (window Event e)
4. Public void window Activated (window Event e)
5. Public void window deactivated (window Event e).
6. Public void window Iconified (window Event e).
7. Public void window Deconified (window Event e).

AWT

- Abstract window Toolkit
- API to develop graphical user interface.

Components | controls.

- canvas	- button
- Scroll bar	- Checkbox
- Label	- List
- choice	

Text component.

- Text Area — Text Field

AWT Layout Managers.

- Border Layout	- Flow Layout
- Card Layout	- Grid Layout
- Grid Bag layout	

Menu Component

- Menu Item — Menu

Container

• Panel	- Dialog
• Window	- Frame

Dialog container

File Dialog.

Applications:-

* Animation
* window based Applicat
* gaming

ANALYTICAL QUESTIONS, ARRAY, LOOP

1 Write a Program to find the sum of the digits of N digit number (sum should be single digit)

Sample Input :

```
Enter N value : 3
Enter 3 digit number : 143
```

Pseudo :

```
User to enter the number
Get the modulus / remainder of number
Sum of the remainder of the number.
Divide the number by 10
```

2 Write a program to find the number of composite numbers in an array of elements.

Sample Input :

```
Array of Elements : {16,18,27,16,23,21,19}
```

Pseudo :

```
Create two for loop first one for input of the element and the second one for logic and condition to count composite no.
In for loop create if, and else if statement
    condition of for loop = (i=0;i<n;i++),
    condition of if = (a[i]=2) (to continue without two because it is prime no)
    condition of else if statement = (a[i] %2 = 0) (and increment of count++ to check next number after increment
After for loop create a if statement, and the condition is "count>2" to print the number which is divide by more than 2 numbers.
```

Sample Output :

```
Number of composite no's : 5
```

3 Write a program for matrix multiplication ?

Sample Input :

```
Mat1   1 2        Mat2   2 3
       5 3               4 1
```

Pseudo :

```
Enter the value of m and n(or) order of first and second matrix.
    Create a matrix of size a[m][n] & b[p][q]
    If the number of columns of first matrix is not equal to the number of rows of the second matrix, print matrix multiplication is not possible and exit. If not, proceed to next step.
    Create a third matrix, c of size m x q, to store the product.
    Set a loop from i=0 to i=m.
    Set an inner loop for the above loop j=0 to j=q.
    Initialize the value of the element (i,j) of the new matrix to 0.
    Set an inner loop inside the above loop from k=0 to k=p
    Using the add and asign operator (+=) store the value a[i][k] * b[k][j] in 3rd, c[i][j]
```

Sample Output :

```
Mat C :  10  5
         22 18
```

4. Write a program to find the Square root of a perfect square number (positive and negative values)

Sample Input : Enter the number 6561

Pseudo:

```
User to enter number is n.
start a loop from 1 to n/2.
During iteration, for every integer 'i', calculate x=i*i.
Now with this 'x' there are 3 possibilities
If x==n there n is perfect square, return true
If x>n then x has crossed the n, and n is not perfect square, return false.
If the above steps a or b are not true, continue.
```

Sample Output : Square Root : 81, -81

5. Program to find the frequency of each element in the array.

Sample Input : {1,2,8,3,2,2,2,5,1}

Pseudo:

```
user to enter the number of Elements
Get the element values and stored
compare stored values
Identify no.of times the value repeat
Print the element and Frequency.
```

Sample output :

Element	Frequency
1	2
2	4
8	1
3	1
4	1

6. Write a program for matrix addition.

Sample Input :

```
Mat1 =  1 2        Mat2 =  2 3
        5 3                4 1
```

Pseudo:

```
Enter the value of m and n(or) order of first matrix.
Enter the value of p and q (or) order of second matrix
    create matrix of size a[m][n] and b[p][q]
    create a new matrix to store the sum of two matrix
    Transverse each element of two matrices and add
    Store this Sum in the new matrix at corresponding
    Print the final new matrix.
```

Sample output :

```
Mat Sum :  3 4
           9 4
```

7. Find the Mean, Median, Mode of the array of numbers?

Sample Input :

```
Array of elements : {16,18,27,16,23,21,19}
```

To Find median : User to enter the number of array elements

Median :

```
Firstly, simply sort the array.
Then, check if the number of elements present in the array is even or odd.
If odd, then simply return the mid value of the array
Else, the median is the average of 2 middle values.
```

Mean :

```
At First find the sum of all the numbers present in the array.
Then simply divide the resulted sum by the size of the array.
```

Sample Output :

```
Mean : 20
Median : 19
Mode : 16.
```

8. Write the program to find whether the person is eligible to vote or not. Input : Get the age from console

Pseudo : If age is below 18, throw the exception with "Not eligible". You are eligible to vote after __ years.
If above 18 and equal to 18, print eligible to vote

Output : Enter your age : 7
Sample output : You are eligible to vote after 11 years.

ANALYTICAL QUESTIONS , STRINGS

1. Write a program to Print the number of vowels in the given statement ?

Sample input : Saveetha School of Engineering

Pseudo :

- user to enter the string
- Declare two variables (vcount for vowel counting and count for consonant counting) to calculate the vowels and consonants in the string and initialize it 0.
- use a for loop to iterate through each character of the string
- use an if condition to check whether any character matches with the vowels in the alphabets
- If any vowel encounters then increment the vcount
- Else if any consonant encounters then increment the count
- Display the values of both the count variables.

Sample output :-

```
Number of vowels = 12
```

2. Write a program to print consonants and vowels separately in given word

Sample input :-

Given word : Engineering

Pseudo :

- user to enter the song
- Declare two variables (vcount for vowel counting and count for consonant counting) to calculate the words vowels and consonants in the string and initialize it to 0.
- use a for loop to iterate through each letter character of the string.
- use an if condition to check whether any character matches with vowels in alphabets
- If any vowel encounters then increment the vcount
- Else if any consonant ecounters then increment the count.
- Display the values of both the count variables.

Sample output :

```
consonants : n g n i n g
vowels     : e i e e i
```

3) write a program that finds whether a given character is present in a string or not. Incase it is present it prints the index at which is present. Do not use built in find functions to search character

Sample input:

Enter the string : I am a programmer

Enter the character to be searched: P

Pseudo:-

1) Create a hash table with (key, value) tuple represented as (character, index) tuple.
2) store the first index of each character of str in the hash table.
3) now, for each character of patt check if it is present in the hashtable or not.
4) If present then get its index from the hash table and update min index
5) For no matching character print " No character present".

Sample output:-

```
P is found in string at index : 8
```

4) write a program to arrange the letters of the word alphabetically in reverse order.

Sample input:

Enter the word : MOSQUE.

Pseudo :

1) Take the first two words. identify the first different letter words. The letter in the first word will precede letter in second word.
2) If there exists no such letter, then the first string must be smaller in length than second string.
3) Assingn the second word to the first word and input the third word into second word.

Sample output:-

```
Order : USQOME
```

5) write a program that accepts a string from user and display same string after removing vowels from it.

Sample input;

Enter a string: we can play the game

Pseudo:-

1) Take string input from user and store is a variable is called as s.
2) After the take string variable s, with empty string 3) After the call replace All () on s subject 4) write regex on replace All() Method like this S1= S.replace All ("[aeiou]", " ");

Output:- w cn ply th gm

ANALYTICAL QUESTIONS , OOP'S EXCEPTION, FRAMEWORK

1. Create a HashTable to maintain a bank detail which includes Account number and the Customer name. let Account number be the key in the Hash Table. Write a Java program to implement the following operations in the Hash Table. Add 3 records
Display the size of Hash Table
Clear the Hash Table

Input : Get account no, customer name from console

Pseudo:

```
Create HashTable object with two wrapper class<Int, String> for having account number as key and customer name.
Add records using add( )
Display the size of HashTable
Display the records inserted in HashTable.
Clear the Hash Table
```

Output: Bank customer details such as account no and customer name (entered in input)

2. Using Iterator in Java to insert the following elements, append + symbol using List Iterator in the each existing element and print them in reverse order. { C, A, E, B, D, F }.

Input:
Set of elements : C, A, E, B, D, F

Pseudo:

```
Create Array List object.
Insert elements as in input.
Create Iterator object and perform hasNext( ) to fetch and display the elements
Create ListIterator object and perform hasNext( ) to fetch and modify the element using set method.
Perform hasNext( ) to fetch and display the modified elements
Perform hasPrevious( ) to fetch and display the elements in reverse order.
```

3. Create a simple generics class with two type parameters for swapping two values of different types.

Input :
two values of different types such as char, Integer, float, double, String.

Pseudo:

```
Create generic class with <T1,T2>
create and define generic method with <T1,T2>
for swapping two variables.
Call the generic method each type separately
```

output :
Two values of different types after got swapped.

4. Generate a Java Code to find the sum of N numbers using array and throw ArrayIndexOutofBounds Exception when the loop variable beyond the size N.

Input :
set of N numbers stored in an array
(1, 2, 3, 4, 5)

Pseudo:

```
Define the array variable with size = 5, and values {1, 2, 3, 4, 5}
Use the try block to contain statement to sum the element, throw the exception if loop variable beyond the size of the array.
Define the catch block ( ArrayIndexOutofBoundsException) to recieve the exception and display proper error message
```

output:
CASE 1 : Sum of elements 15 (or)
CASE 2 : ArrayIndexOutOfBound Exception caught with error message

5. Display the Multiplication Table for 5 and 10 using various stages of life cycle of the thread by generating a suitable code in Java.

Input:
5 10

Pseudo:

```
Create class Table
Define printable( ) with one parameter, have synchronized block in which, display multiplication Table values, use try and catch blocks appropriately.
create Mythread class extends Thread
create object for class for Table and define run( ) to invoke printable(int).
```

output:

```
Multiplication Table of 5 and 10.
5 x 1 = 5          10 x 1 = 10
5 x 2 = 10         10 x 2 = 20
5 x 3 = 15         10 x 3 = 30
5 x 4 = 20         10 x 4 = 40
5 x 5 = 25         10 x 5 = 50
5 x 6 = 30         10 x 6 = 60
5 x 7 = 35         10 x 7 = 70
5 x 8 = 40         10 x 8 = 80
5 x 9 = 45         10 x 9 = 90
5 x 10 = 50        10 x 10 = 100
```

create customer class with deposit() and withdraw() as synchronized Methods. Declare Account No, AccName and Balance as instance variables inside the class. from the Main class Input the amount for withdraw() operation and If requested amount is not available in existing balance amount, withdraw() Method should be temporarily suspended using wait() Method until deposit() Method receives the input for amount to be added in the existing Balance amount and then withdraw() could be completed in a successful Manner. Develop the above Scenario using synchronization and inter- Thread communication

Input: Input the amount for bank balance.

Pseudo: create customer class with amount as class member.

Define withdraw method as synchronized and utilize wait() if thread goes to critical section.

Define deposit Method as synchronized and utilize notify() when recovers.

use try and catch blocks appropriately inside two Methods.

create main method test IC Class

create customer class object and invoke start() and run() methods

2) Establish JDBC connection with MySql to create, insert, update and delete employee records in employee database which consist of following table structure where Eno should be made not null and primary key.

Eno	int
Ename	varchar(20)
Salary	Double

Input: Necessary data (as mentioned in table) from the backend

Pseudo:-

1) Follow the JDBC connectivity steps
2) Create table and store in database
3) Do update and delete operations.
4) Display the data.
5) Close the connection.

output:- Employee records displayed before and after doing update and delete operations.

3) Develop an applet program to compute addition of two numbers using following AWT components.

1) Create a labels as "enter first number", "enter second number" and "Addition of two numbers.

2) Create textboxes for the inputs given by the user and getting output.

Design button as "Result" and "Clear". If "Submit" button is clicked, it has to display the addition of two numbers and "Clear" is clicked then text ~~book~~ box can be cleared.

Input:- Design label box and text box to get two numbers.

Pseudo:- use command button to perform Addition operation by having proper definition for action performed Method

4) Creating student Bio-data form using appropriate AWT controls with JDBC connectivity.

Input: Get Necessary data such as Name, Fathername, DOB, sex, address, qualification, gender, education.

Pseudo: Incorporate AWT controls to get necessary data

Establish JDBC Connectivity steps to interact with database.

use command button to submit the data entered in input controls by having proper definition for action performed Method.

Output:

Data stored at backend for created structure of student bio data