Full Stack Development

Core Java :- Declarations and Access Control, Object Orientation, Operators, Flow Control, Exceptions, Gradle Fundamentals, TDD with Junit 5, Strings, IO, Formating & Parsing, Generics and Collections, Threads.

Full stack development is to development of both front end (client side) and back end (Server side) portions of web application.

Declaration:

* By String literal :- String s = "welcome";
* By new keyword :- String s = new String("welcome");

features of java

→ platform Independent
→ Robust
→ Secured
→ Object oriented
→ Multithreaded
→ high performance
→ Distributed

Access Control

2 - types of AC :-
→ Access Modifier
→ Non-access Modifier

Access Modifier :- Specifies the accessibility or scope of a field, method, Constructor, or class.

4 types:

1. Private :- The access level of a private modifier is only within the class. It can't be accessed from outside the class
2. Default :- The access level of a default modifier is only within the package

within the package and outside the package than Child class. If you do not make the child class, it can't be accessed from outside the package.

4. Public :- The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Prog :-

Private :-

```
class A {
private int data = 40;
private void msg() {
  Sop("Hello java");
}
public class Simple {
PSVM() {
A obj = new A();
Sop(obj.data);
obj.msg();
}}
```

Protected

```
package pack;
public class A {
protected void msg() {
Sop("Hello"); }}

package mypack;
import pack.*;
class B extends A {
  PSVM() {
B obj = new B();
 obj.msg();
}}
```

Public

```
package pack;
public class A {
public void msg() {
Sop("Hello"); }}

package mypack;
import pack.*;
Class B {
  PSVM() {
A obj = new A();
 obj.msg();
}
}
```

Object Orientation

Object is instance of class.
Class is blue print of creating object.

Inheritance:-

It is a mechanism in which one object acquires all the properties and behaviors of a parent object. We can create new classes that are built upon existing class. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover you can add new methods & fields in your current class also.

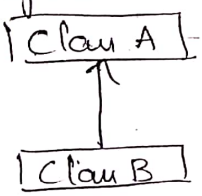
Syntax:- class Subclass-name extends Superclass-name
{
method & fields
}

ex:-

```
class Employee {
float Salary = 40000;
}
class programmer extends Employee {
int bonus = 10000;
 psvm(){
 programmer p = new programmer();
sop("programmer salary is:" + p.salary);
sop("Bonus of programmer is" + p.bonus);
```

Types.

1) Single



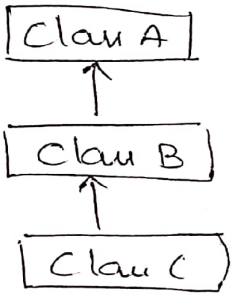
When a class inherits another class, it is known as a single inheritance.

```
Class Animal {
 void eat(){
 sop("eating...");
```

```
class Dog extends Animal {
void bark() {
sop("barking...");}}
class TestInheritance{
psvm(){
Dog d = new Dog();
d.bark();
d.eat();
}}
```

1) Multilevel Inheritance

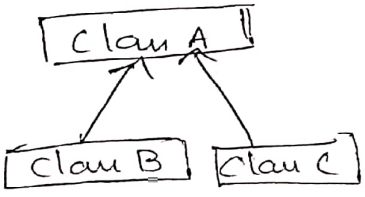
There is chain of inheritance, it is MI.




```
Class Animal {
void eat() { sop("eating---");}}
Class Dog extends Animal{
void bark() { System.out.println("barking...");}}
Class BabyDog extends Dog {
void weep(){ sop("weeping...");}}
Class TestInheritance2 {
psvm() {
BabyDog d = new BabyDog();
BabyDog d = new BabyDog();
d.weep();
d.bark();
d.eat();
}}
```

3) Hierarchical Inheritance.

When two or more classes inherits a single class.




```
Class Animal {
Void eat() { sop("eating...");}
}
```

```
class Dog extends Animal {
void bark() { sop("barking "); }
}
class Cat extends Animal {
void meow() { Sop("meowing "); }
}
class TestInheritance3 {
public static void main(String args[]){
 Cat c = new Cat();
  c.meow();
  c.eat();
 }}
```

Multiple Inheritance

Consider a Scenario where A, B and C are three classes. The C class inherits A and B classes. If A & B classes have the same method and you call it from child class object, there will be ambiguity to call the method of A or B class.

```
class A {
void msg() { Sop ("Hello"); }}
class B {
void msg() { Sop ("welcome"); }
}
class C extends A,B {
public static void main (String args[]){
 C obj = new C();
  obj.msg();
 }}
```

Polymorphism

having many forms.

Method Overloading :-

Multiple methods having same name but different in parameters.

```
class Adder {
static int add(int a, int b) {
  return a+b;
}}
class TestOverloading2 {
public static void main(String[] args){
  Sop(Adder.add(11,11));
  Sop(Adder.add(12.3,12.6));
}}
```

Method overriding

If Subclass has the same method as declared in the parent class.

```
class Vehicle {
void run() {
Sop("vehicle is running"); }
  }
class Bike extends Vehicle{
  psvm(String args[]){
Bike obj = new Bike();
obj.run();
}
}
```

Abstraction.

is a process of hiding the implementation details and showing only functionality to the user.

Abstract class :-

A class is declared as abstract is known as an abstract class.

Rules

→ An abstract class must be declared with an abstract keyword
→ It can have abstract and non-abstract methods
→ It cannot be instantiated
→ It can have final methods
→ It can have Constructors and static method also

```
abstract class Bank {
abstract int getRateOfInterest();
}
class SBI extends Bank {
int getRateOfInterest(){ return 7; }
}
class PNB extends Bank {
int getRateOfInterest(){ return 8; }
}
class TestBank {
PSVM(){
Bank b;
b = new SBI();
Sop("Rate of interest is:"+b.getRateOfInte:
b = new PNB();
Sop("Rate of interest is:"+b.getRateOfInte:
}}
```

Interface in Java

An Interface in Java is a blue print of a class. It has Static Constants and abstract methods.
It is used to achieve abstraction & multiple Inheritance in Java.

Uses:-

1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

Syntax:

```
interface <interface_name> { }
```

Implementation

```
class Classname [extends Superclass] [implements
interface [, Interface--] { }
```

ex:-

```
interface printable {
void print();
}
Class A6 implements printable {
public void print(){ Sop("Hello");
}
PSVM() {
A6 obj = new A6();
```

Encapsulation

Binding or wrapping code and data together into a single unit are known as Encap---

Package

Package is a group of similar types of classes, interfaces and Sub-packages.

Packages are 2 form

→ built-in- package → java, lang, awt, javax, swing, util etc

→ User defined package.

Advantage

It is used to categorize the classes and interfaces so that they can be easily maintained.

Java package provides access protection

Java package removes naming collision

eg:-

```
package mypack;
public class Simple{
  psvm(){
    sop("welcome to package");
}}
```

Package
↓
import
↓
Class

Static keyword

It is used for memory management mainly.

→ variable, method, Block, Nested class

eg:-

```
class Calculate{
static int cube (int x){
return x*x*x;
}
psvm (){
int result = Calculate.Cube(5);
sop(result);
}}
```

this Keyword

It reference Variable that refers to the current obj.

Uses :-

* This can be used to refer current class instance variable.
* This can be to invoke current class method & class Constructor.
* this can be passed as an argument in method call, Constructor call.
* It return the current class instance from the method.

eg :-

```
Class Student {
int rollno;
String name;
float fee;
Student(int rollno, String name, float fee){
this.rollno = rollno;
this.name = name;
this.fee = fee;
}
void display() {
Sop(rollno+" "+name+" "+fee);
}}
Class Test{
PSVM(){
Student S1 = new Student(111, "ankit", 5000f);
Student S2 = new Student(112, "Ajay", 6000f);
S1.display();
S2.display();
}}
```

Operators

- Unary Operator ex- i++, i--, ++i, --i, ~, !
- Arithmetic Operator - *, /, %, +, -
- Shift Operator - << >> >>>
- Relational Operator - <, >, <=, >=, ==, !=
- Bitwise - &, ^, |
- Logical - &&, ||
- Ternary = ? :
- Assignment - =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=

```
public class Operator {
PSVMC){
int a= 10;
int b= 10;
int c= 5;
Sop(a++);
Sop(b++);
Sop(a+b);
Sop(a-b);
Sop(a<<b);
Sop(a<b&&a<c);
Sop(a<b||a<c);
Sop(a&b||a<c); Sop(a+=4);
}}
```

Ternary

```
public class operator {
PSVMC){
int a=2;
int b=5;
int min= (a<b)?a:b;
Sop(min);
}}
```

Flow Control / Control Statements

1. Decision making stat
   - if statements
   - switch statements
2. Loop statements.
   - do while
   - while loop
   - for loop
   - for-each loop.
3. Jump
   - break
   - Continue.

if Statement
- Simple if
- if-else
- if else if ladder
- Nested if

* if :- It evaluate boolean & enable prg to enter a block of co

```
if (condition){
   Statement1;
}
```

eg:-
```
public class Student{
psvm(){
int x=10;
int y=12;
if (x+y>20){
sop("x+y is greater than 20"); }}}
```

* if-else:- which another block of code i.e else block. the else block is executed if the condn. of the if-block is evaluated as false.

Syntax:
```
if (condition){
 statement1;
}
else{
  statement2;
}
```

eg:-
```
public class Student{
psvm(){
int x = 10;
int y = 12;
if (x+y<10){
 sop("x+y is less than 10");
}else{
  sop("x+y is greater than 20");
}}}
```

* if-else-if ladder

It contains the if-statement followed by multiple else-if statements. It is chain of if-else statements. to create a decision tree where the prg may enter in the block of code where condition is true.

Synt:- if (condition 1) {
Statement 1;
}
else if (condition 2) {
Statement 2;
}
else {
Statement 2;
}

eg:-
```
public class Student {
 PSVMC() {
 String city = "Delhi";
 if (city == "Meerut") {
  sop("city is meerut");
 } else if (city == "noida") {
 sop("city is noida");
 } else if (city == "Agra") {
  sop("city is agra");
  } else {
   sop (city);
 }}}
```

* Nested - if

The if state. can contain a if or if-else statement inside another if or else-if statement.

Synta:-
```
if (condition 1) {
 statement 1;
 if (condition 2) {
  statement 2;
 }
 else {
  statement 2;
 }}
```

eg:-
```
publicclass Student {
 PSVMC() {
 String addres = "Delhi, India";
 if (addres.endswith("India")) {
  if (addres.contains("Meerut")) {
  sop("your city is meerut");
  } else if
```

Switc
Switc
The Sr
4 Sir
whic
Synta
Sr
C

eg:-
puk
ps
int
Sr
ca
so
b
ca
Sc
b
cl
S

```
op("your city is noida");
}else
sop(address.split(",")[0]);
}
}else{
sop("you are not living in india");
}
}
```

Switch Statements

Switch statement are similar to if-else-if statements. The switch contains multiple blocks of code called cases & single case is executed based on the variable which is being switched.

Syntax :-

```
Switch(expression){
 case value1:
  statement1;
  break;
  .
  .
  .
 case valuexl:
   statement;
 break;
 default;
   default statement;
}
```

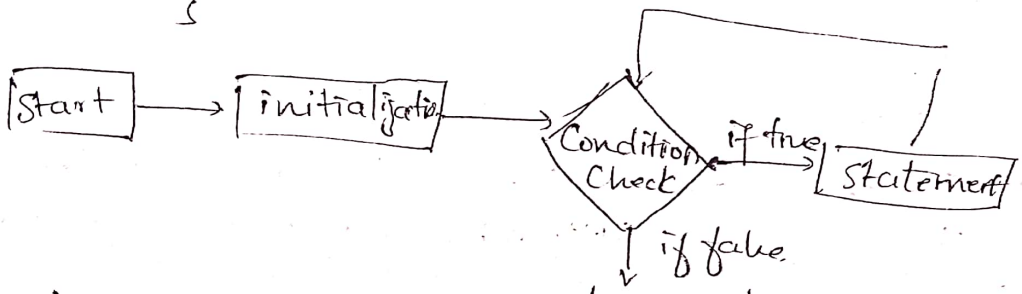
eg :-

```
public class Student implements Cloneable{
 psvm() {
 int num=2;
 switch(num){
 case 0:
 sop("number is 0");
 break;
 case1;
 sop("number is 1");
 break;
 default;
 sop(num);
 }}}
```

Loop statements :- used to execute the set of instruction in a repeated order.

1. for loop :- It enables us to initialize the loop variable, check the condition, & increment/decrement in a single line of code. we use for loop only when we exactly know the no of times, we want to execute block of code.

Syn :- for (initialization, condition, increment/decrement) {
}



eg :-

```
public class Calculation {
 psvm() {
  int sum = 0;
  for (int j=1; j<=10; j++) {
   sum = sum+j;
  }
  sop("The sum of first 10 nos is "+sum);
 }
}
```

for-each loop :- used to traverse the DS like array & collection.

Syn :- for (data-type : array-name/collection_name) { }

Eg :-

```
public class Calculation {
 psvm() {
  String[] names = {"java", "", "c++"};
  sop("printing the content of the array");
  for (String name : names) {
   sop(name);
  }
 }
}
```

3. while loop :- used to iterate over the no of statements multiple time.

Syn :- while (Condition) {
}

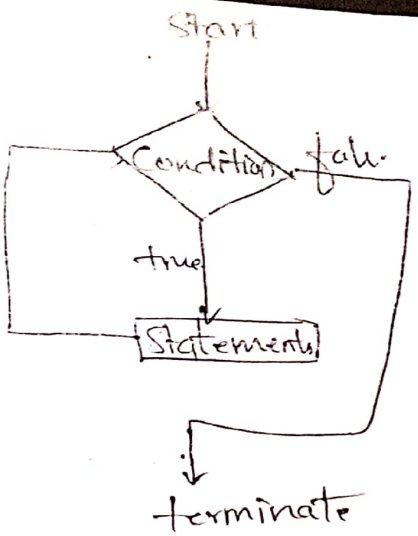
do-w

It chec
the loop
& we ha

Syn :-

eg :-

eg :-

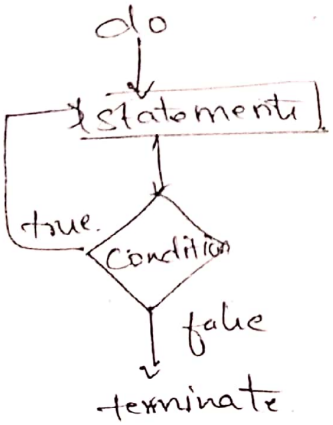


eg:- public class Calculation{
psvm(){
int i = 0;
sop("printing the list of first 10 even no")
while(i<=10){
sop(i);
i=i+2;
}}}

do-while loop

It check the condition at the end of the loop after executing the loop statements. when the no of iteration is not known & we have to execute the loop at least once.

Syntax: do{ }while(condition);



eg:- public class Cal{
psvm(){
int i = 0;
sop("printing the list of first 10 even numbers");
do{
sop(i);
i=i+2;
}while(i<=10);
}}

Jar Dog extends Animals

Motto of TDD
Red → fail
Green → pass
Refactor – removes duplicate code.

TDD with Junits

TDD is process in which testcase are written before the code that validate those cases.
TDD is a technique in which automated unit tests are used to drive the design & free decoupling & dependencies.
⇒ Testing framework used for testing process (unit testing)

Installation of gradle

1. Check if java is installed in your local system. ~~to~~
~~Check if~~
2. Download Gradle.
3. Set Environment Variables
4. Verify gradle Environment.

String is a Sequence of character

```
Char[] ch = {'j','a','v','a'};
String s = new String(ch);
```

Java String class provides a lot of method to perform operation on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring().

~~for 5th sem.~~

I/O:- is used to process the input and produce the output.

~~Exception handling~~

Output Stream:- Java application uses an output stream to write data to a destination. It may be a file, an array, peripheral device or socket

Input Stream:
Java application uses an input stream to read data from a source. it may be a file, an array, peripheral device or socket

Parser:
smaller
A pars
and pr

Generics
The ja
deal wit
by det
In ger
objects

Advant
* Type
* Type
* Con
so pr

ex:-
impor
Class
psv
Array
list.
List.
Stri
Sop
Iter
wh
so
}
}
}

Simple date formate

```
public class date {
 psvm(){
  Date date = new Date();
 SimpleDateFormate s= new SimpleDateFormate("dd|mm|yyyy")
 String strDate = s.formate(date);
   sop(strDate);
 }}
```

Parser: is compiler that is used to break the data into smaller elements coming from lexical analysis phase.
A parser take input in the form of sequence of tokens and produce output in the form of parse tree.

Generics

The java generics programming is introduced in J2SE to deal with type - safe objects. It make the code stable by detecting the bugs at compile time.
In generics java programmer to store a specific type of objects.

Advantages

* Type-safety :- only single type of objects in generic can hold.
* Type casting is not required :- There is no need to typecast the obj.
* Compile time checking :- It is checked at compile time so problem will not occur at run time.

ex:-

```
import java.util.*;
class TestGeneric1{
  psvm(){
 ArrayList<String> list = new ArrayList<String>();
  list.add("rahul");
  list.add("jai");
  String s= list.get(1);
  sop("element is:"+s);
 Iterator<String> itr=list.iterator();
  while(itr.hasNext()){
   sop(itr.next());
  }
 }
}
```

Green → Pass
Refactor.

Concurrent Pattern in Java. Ex:

* Executor → interface that executes tasks
* ExecutorServices → extends Executor, execute() met
* ScheduledExecutorServices → executes tasks periodicall
* Future → respresents the result of asynchrono tasks
* CountDownLatch → utility class blocks threads until some tasks are completed (counter)
* CyclicBarrier → Similar to countDownLatch & reuse the threads, await()
* Semaphore → Set of permits, critical section (Pattern)
* ThreadFactory → Creation of thread on demand
* ~~Blocking~~ blocking Queue → dequeing, Enqueuing, (to perform) & flow control.

asynchronous → no of threads performs simultaneously

* Delay Queue → Priority Queue based on delay time
* class of type Delayed
* Lock → lock(), unlock() → not allow thread to access Some logical part of code.
* Phaser → reusable barrier → dynamic threads needs to wait before execute or continues, phases are coordinated → revising the instance of phases.

Concurrent collections in java.

without concurrent collection
- non synchronized
- → no thread safety
- → Synchronized → low performance, needs wait/re
- → Runtime Exception

Concurrent collections are synchronized
→ high Performance
No return exception.

```
import java.util.concurrent.CopyOnWriteArrayList;
import java.util.*;
class CurrentDemo extends Threads {
  static CopyOnWriteArrayList l = new ArrayList();
public void run()
{
  try {
      Thread.sleep(2000);
      }
    catch (Interrupted Exception e) {
SOP("Child threads" + " adds element");
child thread ← l.add("D");
}

public static void main(String [] args)
  throws Interrupted exception
```

(Reflection, Stream, Collection)

main thread
```
{ l.add("A");
  l.add("B");
  l.add("C");
Concurrent Demo t = new
t.start();
Iterator itr = l.iterator();
while(itr.hasNext()){
 String s = (String) itr.next();
 SOP(s);
Thread.sleep(6000);
}
 S.O.P(l);}
```

String Operation

Ex:
```
List<String> show = names.stream().sorted().collect
        (collectors.toList());
Syso(show);
List<Integers> numbers = Arrays.asList(2,3,4,5,2);
Set<Integer> SquareSet = numbers.stream().map(x->x*
.collect(collectors.toSet());
Syso(Square Set);
number.stream().map(x->x*e).forEach(y->Sysdy
even = numbers.stream().filter(x->x%2==0).reduce
(0, (ans,i)) -> ans+i);
Syso(even);
```

JDBC.

✓ Import
* Load
* Register
* Establish

Collection
Group of obj.

SHIVAS
Date
Page

* Lamda expression
* Stream API - Ex.

R
E
F
E

* Create statement
* Execute Query
* Close connection.

Design patterns in Java
Creational
Structural
Behavioural.

Lamda expression → Syntax, Expression, Arrow mark.

```
import java.util.ArrayList;
public class Main {
public static void main (String[] args){
ArrayList<Integer> numbers = new ArrayList<Integers>()
numbers.add(5);
numbers.add(9);
numbers.add(8);
numbers.add(1);
number.forEach((n)-> {Sys o(n); });  } }
```

Lambda ⇒ Stored in variable type interface.
Ex: Consumer

Lambda expression can be stored in variable if the variables type is an interface which has only one method.

Consumer → Interface is used found in java.util. package to store a lambda expression in a variable.

```
import java.util.ArrayList;
import java.util.function.Consumer;
public class main {
public static void main (String[] args) {
ArrayList<Integer> number = new ArrayList<Integer>();
```

```
number.add(5);
number.add(9);
number.add(8);
number.add(1);
Consumer<Integer> method = n -> { Syso(n); };
number.forEach(method); } }
```

If the lambda expression needs to return a value then the code block should have a return statement

JDBC.

Ex:

```
import java.sql.*;
class OracleCon {
PSVM() {
try {
Class.forName("Oracle.jdbc.driver.OracleDriver
Connection con = DriveManager.getConnection "jdbc:
oracle:thin@localhost:1521:xe", "System", "Oracle");
Statement stml = con.createStatement()
Result set rs = stml.executeQuery ("Select * from em
while (rs.next()).
SOP(rs.getInt(1)+" "+rs.getString(2)+" "+rs.get
        String(3));
con.close();
}
catch (Exception e)
{
Syso(e); } } }
```