**Java Full Stack Training Notes**

**Day 1**

**11/01/2020**

**Database :**

Program :

Input : Data

Process : Business rules

Output :Data

1 File based system

2 Database System

Limitation of File base system

1. Data Redundancy (Storing same records or duplicate records)
2. Data inconsistency (format of the file)
3. Data Security (read or write mode)

.txt

Employee.txt

Id,name,Salary

1,Ravi,12000

2,Ramesh,1400

Employee.txt

Id name Salary

1 Ravi 12000

2 Ramesh 1400

.doc

.pdf

**Database :**

Data : row fact

Information : Processed data or meaningful data.

Database : Storing data and information in proper format like table.

DBMS :Database Management System. It is a software which help to store the data in a table format.

Excel sheet is a small DBMS.

Employee – Table

Columns

Id Name Salary Row

1 Ravi 12000

2 Ramesh 14000

Database Model

Hierarchical Model

Network Model

RDBMS

Relational model

TrainerStudentRecords

TId TName Tech Sid SName Age

1 Raj Java 100 Seeta 21

1 Raj Java 101 Meeta 22

1 Raj Java 102 Leeta 23

2 Ravi Python 103 Keeta 24

2 Ravi Python 104 Veeta 25

Trainer

PK

Tid TName Tech

1 Raj Java

2 Ravi Python

Student

PK FK

Sid SName Age TSId

100 Seeta 21 1

101 Meeta 22 1

102 Veeta 23 2

103 Leeta 24 null

104 Teeta 25 3 Error

RDBMS : Table is known as Relation

Column is known as Attribute

Row is known as Tuple or records

1970

Dr EF Codd’s Rules

12 Rules start from 0 to 11

RDBMS

RDBMS Company

Oracle Oracle

MySQL Sun Microsystem (Oracle)

SQL Server

2020/19 Microsoft

Db2 IBM

SQL : Structure Query Language

5 types

1. DRL or DQL (Data Query Language or Data Retrieval Language)
   1. Select Clause
2. DDL : Data Definition Language
   1. Create, Drop, Alter , Truncate , rename
3. DML : Data Manipulation Language
   1. Insert , Delete and Update
4. TCL : Transaction Control Language
   1. Commit, rollback and savepoint
5. DCL : Data Control Language
   1. Grant and revoke

Oracle :

MySQL :

90% SQL Query

**Oracle :**

In Oracle Database Login itself is database.

**scott : username**

**By default oracle database username is scott/admin/oracle etc**

**Scott username**

**Tiger password**

**select \* from tab;** tab is pre-defined table provided by oracle database to check all pre-defined as well as user-defined tables.

MySQL/db2/sql server

**create database databaseName;**

**show databases;**

**use databaseName;**

**show tables;**

**DRL or DQL**

**use test**

**show tables**

**select \* from tableName;**

select \* from employees;

select \* from departments;

select \* from locations;

to check the table structure

**desc tableName;**

desc employees;

display specific columns

select columnname,columname from tableName;

**columnName alias**

select employee\_id as emp\_id, first\_name as f\_name, last\_name as l\_name from employees;

select first\_name,last\_name,concat(first\_name,last\_name) as Full\_name ,salary,salary+1000 as GrossSalary from employees;

Employee\_Id, FullName, Basic\_Salary, HRA, DA, PF, GrossSalary

**1st Assignment**

Full\_name --- First\_name and Last\_name

Salary --- Basic\_Salary

HRA –10%

DA – 5%

PF – 7%

GrossSalary – Salary +HRA + DA – PF

**Filter the Data**

Where clause

Select \* from tableName;

Select columnName,columName from tableName 🡪 filter the columns

1. Relational Operators

Select \* from tableName where columnName RO Value

>

>=

<

<=

= (is a equal Operator in SQL)

!= or <>

select \* from employees where salary > 10000;

select \* from employees where salary <> 17000;

1. Between operator (range of records)

Select \* from tableName where columnName between min and max

Select first\_name from employees where salary between 5000 and 10000

MySQL default date format is yyyy-mm-dd

Oracle default date format is dd-mon-yy

Display all employee first\_name and salary where hire date between 1990 to 1995.

1. In operator (specific more than one values)

Select \* from tableName where columName in (v1,v2,v3);

select first\_name,salary,hire\_date from employees where job\_id in('IT\_PROG','ST\_MAN');

select first\_name,salary,hire\_date from employees where employee\_id in(100,120,10000);

1. Like operator

Select \* from employees where first\_name like ‘Steven’;

Select \* from employees where first\_name = ‘Steven’;

Start with **S** Character ‘

Select \* from employees where first\_name like ‘S%’;

End with **a** character

Select \* from employees where first\_name like ‘%a’;

Contains **e** character

Select \* from employees where first\_name like ‘%e%’;

% zero or infinity

\_ any single character

Start with **L** as first character, second may be anything and third character must be **u**

select first\_name from employees where first\_name like 'L\_u%';

1. is null

select first\_name,salary,commission\_pct from employees where commission\_pct is null;

select first\_name,salary,commission\_pct from employees where commission\_pct is not null;

**Logical operators**

1. and
2. or
3. not

**Not**

select \* from employee where salary between 10000 and 20000

select \* from employee where salary not between 10000 and 20000

select \* from employee where job\_id in(‘IT\_PROG’,’ST\_MAN’)

select \* from employee where job\_id not in(‘IT\_PROG’,’ST\_MAN’)

select \* from employee where first\_name like ‘Steven’;

select \* from employee where first\_name not like ‘Steven’;

**and**

select first\_name from employees where salary > 12000 and department\_id=90;

**or**

select first\_name from employees where salary > 12000 or department\_id=90;

**Order by clause**

This clause mainly use to sort the records ascending or descending order.

Select \* from tableName order by columnName asc/desc

By default asc consider

select first\_name,salary from employees order by salary;

select first\_name,salary from employees order by salary asc;

select first\_name,salary from employees order by salary desc;

select first\_name,salary from employees where employee\_id between 110 and 150 order by salary desc;

Multi sort

select first\_name,department\_id,salary from employees order by department\_id desc,salary asc;

**Joins :** Joining is use to retrieve more than one columns from more than one table with or without conditions.

Create table table1(srno int, name varchar(10), salary float);

Insert into table1 values(1,’Ravi’,12000);

Insert into table1 values(2,’Ram’,14000);

Create table table2(accno int, name varchar(10), amount float);

Insert into table2 values(100,’Ravi’,500);

Insert into table2 values(101,’Ramesh’,1000);

1. **Cartesian product :**

Select columnname1,columnname2…… from table1, table2…. Tablen

M\*N

select srno,salary,accno,amount from table1,table2;

**Table Name with common columns**

select table1.srno,table1.salary,table1.name,table2.accno,table2.amount from table1,table2;

**Table alias**

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1,table2 t2;

**Equi Join using where clause**

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1,table2 t2 where t1.name = t2.name;

Common records

Left Outer Join : Left table remaining records + common records

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1,table2 t2 where t1.name = (+)t2.name; **In oracle database**

Right Outer Join : right Table remaining records + common records

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1,table2 t2 where t1.name(+) = t2.name; **In oracle database**

**Inner Join using on clause**

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1 inner join table2 t2 on t1.name = t2.name;

**Left Outer join**

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1 left outer join table2 t2 on t1.name = t2.name;

**Right Outer join**

select t1.srno,t1.salary,t1.name,t2.accno,t2.amount from table1 t1 right outer join table2 t2 on t1.name = t2.name;

**FK = PK**

**De-Normalization**

Employee

Id Name

10

Project 5 employee – 5 project

Pid ProjectName

10

**Normalization**

First\_name,Salary,Department\_name

select emp.employee\_id,emp.first\_name,emp.salary,dept.department\_name from employees emp, departments dept where emp.department\_id = dept.department\_id;

First\_name, Department\_name, City

Where clause

On clause

**Self Join** : Joining the same table itself is known as Self join

select emp1.employee\_id,emp1.first\_name,emp1.job\_id,emp2.first\_name,emp2.job\_id from employees emp1, employees emp2 where emp1.manager\_id = emp2.employee\_id;

**MySQL functions**

Function : It contains set of instruction to perform a specific task.

Database Function : Function take one or more parameter and return output or results.

2 types of

1. Pre-defined function
2. User-defined function (PL SQL).

Pre-defined functions

1. Single row function : The output or result apply for every records independently.
2. **String functions**

select **upper**(first\_name),**lower**(first\_name),**concat**(first\_name,' ',last\_name),**length**(first\_name) from employees;

nested function

select **length(concat**(first\_name,' ',last\_name)) from employees;

select **substring**(first\_name,2,2) from employees;

1. **Date functions**

select upper('raj deep'),curdate(),sysdate();

select datediff('2021-01-11','2021-01-05');

select datediff(curdate(),'2021-01-05');

select date\_format(curdate(),'%d-%m-%y');

1. **Maths functions**

select ceil(10.1),ceil(10.9),floor(10.1),floor(10.9);

select round(2356.4578,2);

select truncate(2356.4578,2)

1. Multi row function : The output or result apply for all records depending upon the group.

**Aggregate Functions**

sum(), count(), max(), min(), avg() etc

select sum(salary) as TotalSalary from employees;

select avg(salary) as AvgSalary,max(salary) as MaxSalary,min(salary) as MinSalary from employees;

select count(commission\_pct),count(manager\_id),count(employee\_id) from employees;

select count(\*), count(employee\_id) from employees;

**group by clause**

select aggregatefunction from tableName group by columnName(FK)

select sum(salary) from employees group by department\_id;

select department\_id,sum(salary) from employees group by department\_id;

**Where clause**

It is before group by

Condition apply for every individual records

Where clause we can use without group by

**Having clause**

After group by clause

Conditions apply for group by records.

But having clause we have to use with group by clause

select department\_id,sum(salary) from employees where department\_id is not null group by department\_id having sum(salary) > 50000 order by department\_id desc;

**Git : Sub Version Control System**

**Centralized repository**

Github :

Code commit : AWS

Azure

Google cloud

Oracle Cloud

DevOps tools

**Day 2**

**12/01/2020**

**DDL :**

**Create table**

**Syntax**

**Create table tableName(columnName dataType1,**

**columnName dataType2,**

**columnName datatTypen)**

**Emp**

**Id Name Salary**

**int varchar(10) float**

**Oracle : number(10), number(10,2)**

**create table emp(id int,name varchar(10), salary float);**

**DML :**

Insert into tableName(columnName1,columnName2,columnName3) values(v1,v2,v3);

insert into emp(id,name,salary) values(1,'Ravi',12000);

insert into emp values(2,'Ramesh',14000);

insert into emp(id,name) values(3,'Raju');

insert into emp values(4,'Ajay',null);

insert into emp(id) values(5);

insert into emp values(6,null,null);

insert into emp(name,id,salary) values('Dinesh',7,22000);

**Delete Query**

delete from tableName; : all records

**delete from emp;**

Delete with where clause

Delete from tableName where clause

**delete from emp where id=1;**

**delete from emp where name like ‘Ravi’;**

**delete from emp where salary between 2000 and 5000;**

**update Query**

syntax

update tableName set columnName = value;

update emp set salary = 20000; update salary for all employee

update with where clause

update tableName set columnName = value where clause

update emp set salary = 20000 where id=2;

update emp set name=’Ravi Kumar’ where name like ‘Ravi’;

update emp set salary = 22000 where salary = 18000;

DDL

Create

Drop

Drop table tableName ;

drop table sample;

**drop table sample;**

it will remove all records as well as table structure.

Truncate

Truncate table tableName;

It remove all records from table but maintains the table structure.

**truncate table sample**

delete

1. It is a part of DML
2. Using delete we can use where clause
3. Without where clause delete all records but maintains table structure.
4. We can use TCL commands.

truncate

1. It is part of DDL
2. We can’t use where clause
3. Delete all records but maintains table structure.
4. We can’t use TCL commands.

drop

1. It is part of DDL
2. Remove all records as well as table structure.

Alter command

1. Add column

alter table emp add desg varchar(2);

1. Modify column

alter table emp modify desg varchar(10);

1. Rename the column

alter table emp rename column desg to designation;

1. Change the tableName

alter table emp rename to empdetails;

1. Drop column

alter table empdetails drop column designation;

DDL

DML

Online shopping

Online Examination

custId,custName,age,phnumber,accnumber,typeofaccount,amount,pid,productName,price,mgrId,managerName,phnumber etc

**Single table**

**Super keys**

**Any one**

custId PK

accNumber, unique

phNumber, unique

mgrid unique

pid unique Candidate keys

CustId,CustName : super key but not candidate key

CustName not super key not a candidate key

CustId : is super key as well as candidate key

cusId,mgrId,

custId,Pid

custId,Pdi,mgrId

custId,phNumber

custid,phNumber,pid,

etc

**Candidate keys**

custid,phNumber,pid

phNumber,Pid

Pid CK

PhNumber CK

CustId,CustName :Super key

CustName :not CK

CustId :CK

MySQL

auto\_increment : generate the numbers one by one

Oracle

sequence: using this they generate auto increment number.

**🡨-PK-🡪(ProductId,customerId):**

**ProductId, CustomerId Price**

A X 10

A Y 20

B X 20

B Y 40

A X 50 Error

Composite primary key

**Constraints :** It is use to restrict the user to insert the invalid data in table.

1. Not null
2. Unique
3. Default
4. enum
5. Primary key
6. Foreign key
7. Check constraints : it doesn’t support.

EmpDetails

PK not null male/female unique IBM

EmpId FName Age gender phnumber Company

create table empdetails(

empid int primary key,

fname varchar(10) not null,

age int,

gender enum('male','female'),

phnumber int unique,

company varchar(10) default 'IBM');

**Relationship : Using Primary key and foreign key**

4 types

One means primary key and many foreign key

1. **One – to – many : Trainer -- Student**
2. **Many – to – one : Employee -- Project**
3. One – to one : Person -- Passport
4. Many – to – Many : Students -- Technologies

ER Diagram : Entity Relationship Diagram

One – to – Many

Trainer

PK

TId TName Tech

Column Level

**create table trainer(tid int primary key,tname varchar(10) not null,tech varchar(10) not null);**

Table Level

**create table trainer(tid int, tname varchar(10) not null, tech varchar(10) not null, constraint t\_pk primary key(tid));**

Student

PK FK

Sid SName Age TSId

**create table student(sid int primary key,sname varchar(10) not null,age int,tsid int references trainer(tid)); Not consider as FK in mysql but consider as In Oracle.**

Table Level

**create table student(sid int,**

**sname varchar(10) not null,**

**age int, tsid int,**

**constraint s\_pk primary key(sid),**

**constraint ts\_fk foreign key(tsid) references trainer(tid));**

**create table trainer(tid int,**

**tname varchar(10) not null,**

**tech varchar(10) not null,**

**constraint t\_pk primary key(tid));**

**Drop Foreign key**

alter table student drop foreign key ts\_fk;

**Add foreign key on student table after table created with records**

alter table student add constraint ts\_fk foreign key(tsid) references trainer(tid);

**Drop the primary key**

alter table student drop primary key;

**Add the primary key after table created with records (records must be unique)**

alter table student add constraint s\_pk primary key(sid);

**Keys**

Constraints

**One to One**

Person

create table person(pid int, pname varchar(10), primary key(pid));

1 Raj

Passport

1

create table passport(ppid int, typeof varchar(10),primary key(ppid),foreign key(ppid) references person(pid));

One – to – One (primary key and foreign key)

Many to – Many

Students SkillSets

Sid PK SSId PK

SName SkillName

create table students(sid int,sname varchar(10), primary key(sid));

create table skillset(ssid int,skillname varchar(10), primary key(ssid));

One – to – many bidirectional is known as many – to – many relationship.

Third table

Maintain the relationship

create table students(sid int,sname varchar(10), primary key(sid));

1. Ravi
2. Ramesh

create table skillset(ssid int,skillname varchar(10), primary key(ssid));

100 Java

101 Python

102 Angular

create table students\_skillset(

sss\_id int primary key auto\_increment, 1,2,3,4,

s\_ss\_id int, FK

ss\_s\_id int, FK

foreign key(s\_ss\_id) references skillset(ssid),

foreign key(ss\_s\_id) references students(sid));

1, 100, 1

2,101, 1

3, 102, 1

4, 100, 2

5,101, 2

**On delete cascade**

alter table student add constraint ts\_fk foreign key(tsid) references trainer(tid) on delete cascade;

**On Update cascade**

alter table student add constraint ts\_fk foreign key(tsid) references trainer(tid) on update cascade;

**On delete set null**

**On update set null**

Check constraints syntax

**create table sample(id int primary key,amount float check(amount>500));**

**Sub Query :**

Query within another query is known as sub query

Syntax

Outer query (Inner Query)

Outer Query (Inner Query (Inner Inner Query))

Sub Query must be return only columns

1. Single row sub query
2. Multi row sub query

Single row sub query

Select \* from tableName where columName RO (select columnName from tableName);

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

Multi row sub query

Select \* from tableName where columnName in | RO any | RO all (select columnName from tableName)

Find the person name whose salary is greater than average salary of all employee working in a department 60 or Finance.

**Without sub query**

select avg(salary),count(\*) from employees where department\_id=60;

select first\_name from employees where salary > 5760;

**With sub query**

select first\_name from employees where salary > (select avg(salary) from employees where department\_id=60);

Multi row sub query

In

RO any

RO all

Find person name and job\_id whose min salary must be > 2000 and max salary must be < 10000.

select first\_name from employees where job\_id in (select job\_id from jobs where min\_salary > 2000 and max\_salary <10000);

In with Join

Display FirstName, Job\_Title whose salary between 5000 and 10000

RO any

RO all

Select salary from employees where department\_id=30;

Min 🡪 2500

Max🡪11000

>any ( > min salary of inner query)

select first\_name,salary from employees where salary >any (select salary from employees where department\_id=30);

select first\_name,salary from employees where salary > (select min(salary) from employees where department\_id=30);

>all (> max salary of inner query)

select first\_name,salary from employees where salary >all (select salary from employees where department\_id=30);

select first\_name,salary from employees where salary > (select max(salary) from employees where department\_id=30);

**exists**

select first\_name from employees where exists (select \* from employees where department\_id=100);

**not exists**

select first\_name from employees where not exists (select \* from employees where department\_id=100);

Regular Expression : search the contains base upon the pattern.

Start with S character

select first\_name from employees where first\_name regexp '^S';

End with a character

select first\_name from employees where first\_name regexp 'a$';

Range

Start with any character

select first\_name from employees where first\_name regexp '^[A-D]'

\* 1 to many

+ 0 to many

? optional (0 to 1)

git add .

git commit –m “Message”

git status

git pull

Project

Pid PName

create table project(pid int, pname varchar(10) not null);

Primary key

**alter table project add constraint p\_pk primary key(pid);**

Employee

Empid EmpName ProjectId;

create table employee(empid int, empname varchar(10), pid int);

**alter table employee add constraint emp\_pk primary key(empid);**

**alter table employee add constraint emp\_proj\_fk foreign key(pid) references project(pid);**

SQL

NoSQL

MongoDB

JSON

Document

Id,name,age

Id,phnumber

Id,image

Info

**Day 3**

**13/01/2020**

**Java : Java is pure object oriented and platform independent programming language**

**Till Java 7**

**Java is combination on procedure and object oriented programming language**

**From Java 8**

**Java is combination of procedure, object, and functional programming language(Lambda).**

**1991 Oak. Initial Name of java is Oak. In Nov 1995 rename to Java.**

**It was part of sun microsystem**

**It is a part of Oracle.**

**James Gosling and Team**

**Version 1.0 7, 8, 9, 14 version.**

**Java 7.**

**Java 8.**

**Features**

1. **Simple : C/C++. Pointer, Memory Management, Operator Overloading, Friend, Virtual keyword, delete etc.**
2. **Pure object oriented programming language.**
3. **Platform independent programming language : Write only run anywhere.**
4. **Compiler and interpreter :**
5. **Portable :**

**Portable : machine must be different.**

**Platform Independent : OS must different**

1. **Exception Handling**
2. **Multi threading**
3. **Distributed application : RMI**

**Basic Programming :**

**class**

**syntax**

**class className {**

**fields/variable**

**function /methods;**

**}**

**className follow pascal naming rules**

1. **If class contains one word first letter must be upper case.**
2. **If class contains more than one word each word first letter upper case.**

**Demo**

**Employee**

**EmployeeDetails**

**ManagerInfo**

**etc**

**open command prompt**

**java –version**

**javac**

**java**

**Simple Java program**

class Demo {

public static void main(String args[]) {

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

}

}

To compile

javac Demo.java

To run

java Demo

**Another Example**

class Demo {

public static void main(String args[]) {

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

System.out.print("Welcome to Java......");

System.out.printf("Welcome to Java......");

}

}

**Data Types :** It is a type of data which tell what type of data it can hold.

Data types

2 types

1. Primitive data type : It is use to store only value.
   1. byte 1 byte
   2. short 2
   3. int 4
   4. long 8 Number without decimal point
   5. float 4
   6. double 8 with decimal
   7. char 2 any single character
   8. boolean 1 bit true or false.
2. Non primitive data type or reference data type : it is use to store value as well as reference of another data type.

**Data Types example**

class Demo {

public static void main(String args[]) {

int a=10;

double b = 10.20;

char c ='%';

boolean res = true;

System.out.println(a);

System.out.println("The value of a "+a);

System.out.println(b);

System.out.println(c);

System.out.println(res);

}

}

**Type casting :** Converting from one data type to another data type is known as type casting.

2 types

1. Implicit type casting : By default internally taken care by JVM
2. Explicit type casting : We have to convert.

int family

-----------------------------🡪 Implicit --------------------------🡪

byte short int long

🡨------------------------Explicit ----------------------------------------

**Type casting example**

class Demo {

public static void main(String args[]) {

byte a=10; // byte range -128 to 127

short b=a; //implicit

System.out.println(a);

System.out.println(b);

short c = 129;

byte d = (byte)c; //Explicit (type)variableName;

System.out.println(c);

System.out.println(d);

}

}

Byte class

byte : primitive

int :primitive

Integer : class

int to float

----🡪 implicit ---------------🡪

int float

🡨-------explicit -----------------------

**Example**

class Demo {

public static void main(String args[]) {

int a=10;

float b=a; //Implicit

System.out.println(a);

System.out.println(b);

//float c = 10.0f;

float c = (float)10.10;

int d = (int)c;

System.out.println(c);

System.out.println(d);

}

}

**Operators**

1. Arithmetic Operator : +, -, \*, /, % (remainder):
2. Logical Operator : &&, ||, !
3. Conditional Operator : >, >=, <, <=, ==, !=
4. Assignment operator : =
5. Increment and decrement : ++, -- increment means increment the value by one. Decrement means decrement the value by 1
6. Ternary operator
7. instanceOf

+ve or –ver number is known as true

Zero known as false

class Demo {

public static void main(String args[]) {

int a=10;

System.out.println(a); //10

++a; // pre-incremenet

System.out.println(a); //11

a++; //post increment

System.out.println(a); //12

}

}

Another example

class Demo {

public static void main(String args[]) {

int a=10;

int b;

b =a;

System.out.println(b); //10

b = a++; // pre-incremenet increment and assign

System.out.println(b); //10

b = ++a; //post increment assign and increment

System.out.println(b); //12

}

}

**Ternary operator**

class Demo {

public static void main(String args[]) {

int a=10;

int b=50;

int res = a>b?a:b;

System.out.println(res);

}

}

**Another Ternary Operator**

class Demo {

public static void main(String args[]) {

int a=10;

int b=20;

int c=30;

int d =50;

int e = 60;

int f = 70;

int res = a>b?( c>d?c:d ): (e>f?e:f);

System.out.println(res);

}

}

**If statement**

1. simple if

if(condition) {

}

1. if else

if(condition){

}else {

}

1. nested if : if within another if

if(condition) {

if(condition) {

}else {

}

} else {

if(condition) {

}

}

Example

class Demo {

public static void main(String args[]) {

// largest of three number

int a=3;

int b=1;

int c=2;

if(a>b) {

if(a>c) {

System.out.println("a is largest");

}else {

System.out.println("c is largest");

}

}else {

if(b>c) {

System.out.println("b is largest");

}else {

System.out.println("c is largest");

}

}

}

}

1. if ladder or if else if

if(condition) {

}else if(condition) {

}else if(condition) {

}else {

}

**switch statement**

**syntax**

switch(variableName) {

case value: block1;

break;

case value: block2;

break;

case value :block3;

break;

default : wrongblock

break;

}

switch, case, break and default are keywords.

variablename must be type of int, char or String.

**Looping :** it use to execute set of statement again and again till the condition become false.

1. While loop
2. Do while loop
3. For loop
4. For each loop or enhanced loop

While loop

Initialization : start end

while(condition) {

body of the loop

increment or decrement

}

While loop example : Entry loop

class Demo {

public static void main(String args[]) {

int i=1;

int n=10;

while(i<=n) {

System.out.println(i);

i++;

}

System.out.println("Finish");

}

}

Do while loop example : Exit loop

class Demo {

public static void main(String args[]) {

int i=1;

int n=10;

do {

System.out.println(i);

i++;

}while(i<=n);

System.out.println("Finish");

}

}

**For loop**

class Demo {

public static void main(String args[]) {

for(int i=0;i<=10;i++) {

System.out.println(i);

}

System.out.println("Finish");

}

}

class Demo {

public static void main(String args[]) {

for(int i=0,j=10;i<=j;i++,j--) {

System.out.println(i +", "+j);

}

System.out.println("Finish");

}

}

**Taking the value through keywords**

1. Using Scanner class
2. Using DataInputStream
3. Using BufferedReader
4. Using command line arguments

**Scanner :** Scanner is a pre-defined class part of util package. Which provide set of methods which help to scan the value through keyboards.

Syntax to create the object of Scanner

**Scanner obj = new Scanner(System.in);**

**obj.nextByte();**

**obj.nextShort();**

**obj.nextInt();**

**obj.nextLong();**

**obj.nextFoat();**

**obj.nextDouble();**

**obj.nextBoolean()**

**obj.nextChar() no methods**

**String name = obj.next(); one more than one character.**

**char name[]=name.toChar();**

**char val = name[0];**

**obj.next(); terminator is space**

**Raj Deep**

**obj.nextLine() terminator is enter key**

**Taking employee details through keyboards**

import java.util.Scanner;

class Demo {

public static void main(String args[]) {

Scanner obj = new Scanner(System.in);

System.out.println("Enter the id");

int id = obj.nextInt(); //scan int value

obj.nextLine(); // hold the enter key

System.out.println("Enter the name");

String name = obj.nextLine();

System.out.println("Enter the salary");

float salary = obj.nextFloat();

System.out.println("The id is "+id);

System.out.println("The name is "+name);

System.out.println("The salary is "+salary);

}

}

**Assignment 1**

do{

Online Examination

1:English , 2 : Math 3 : GK

switch() {

case 1

3 Q

case 2

3 Q

case 3

3 Q

}

Do want to continue ?

}while();

**Result g\_total > 70**

**Result + 10**

**Result>=90 selected else try next time.**

System.out.prinntln();

**array :** array is user defined data type which is use to store same type of values.

Syntax

datatype arrayName[];

int abc[]; //Java

int abc[10]; //C or C++

**for each loop**

for(datatype variableName: arrayName) {

}

**Array with for and for each loop**

class Demo {

public static void main(String args[]) {

int []abc={10};

int []xyz={10,20,30,40,50,60,150,12,34,56,34,32,45,67,78,99};

System.out.println(xyz[0]);

System.out.println(xyz[5]);

System.out.println("Size of array is "+xyz.length);

System.out.println("Using for loop");

for(int i=2;i<8;i=i+2) {

System.out.println(xyz[i]);

}

System.out.println("Using for each loop");

for(int n : abc) {

System.out.println(n);

}

}

}

**Syntax to create the memory size for the array**

datatype arrayName[] =new datatype[size];

int []abc = new int[10];

String []names=new String[10];

char name[]=new char[20];

**Assignment 2**

Take n number records through keyboards as Id,Name,Salary,Desg

(array id,name,salary,Desg).

Salary = salary + hra + da – pf;

Hra is 10% salary

Da is 7 % salary

Pf 5 % salary

If desg is manager desg.equals(“Developer”)

15% bonus

If developer 10% bonus

Else

5 % bonus

Id, name, salary( grossSalary +bonus ) and desg

**OOPs**

**object :**

**object is any real world entity**

**state or properties –have --- fields/variable**

**Person**

**Behaviour---do/does --- function / methods**

**Bank**

**Animal**

**Car**

**Object is a concept.**

**class : Blue print of object**

**Template of object**

**In java method as well as variable must follow camel Naming rules**

1. **One word first then it must be lower case**
2. **More than one word from second word onward each word first letter upper case.**

**Syntax**

**ClassName objectRefereName = new ClassName();**

**class and object example**

class Car {

int wheel;

String color;

float price;

void start() {

System.out.println("Car Started...");

}

void appliedGear() {

System.out.println("Gear Applied successfully ");

}

void moving() {

System.out.println("Car is moving");

}

void stop() {

System.out.println("Car Stopped...");

}

}

class App {

public static void main(String args[]) {

System.out.println("Main method");

Car innova = new Car(); //heap memory

innova.start();

innova.appliedGear();

innova.moving();

innova.stop();

Car santro = new Car();

santro.start();

santro.appliedGear();

santro.moving();

santro.stop();

}

}

**Type of fields / variables**

3 types

1. Instance variable
   1. The variable which declare inside a class but outside a method including main method is known as instance variable.
   2. Instance variable hold default value according to their data types like int family 0, float family 0.0, char space, boolean false and String null.
   3. Instance variable we can use directly inside a method but the method must be part of same class as well as it must non static method.
2. Local variable
   1. The variable which declare inside a method including main method is known as local variable.
   2. Local variable doesn’t hold default value we have to initialize.
   3. Scope of local variable within that block where it declare.
3. Static variable or class variable

**Example of instance variable and local variable**

class Car {

int wheel;

String color;

float price;

void carInfo(String name) {

String msg=name+", Car Details ";

System.out.println(msg);

System.out.println("Wheel "+wheel);

System.out.println("Color "+color);

System.out.println("Price "+price);

}

}

class App {

public static void main(String args[]) {

System.out.println("Main method");

Car innova = new Car(); //heap memory

Car santro = new Car(); //heap memory

innova.wheel =4;

innova.color="Gray";

innova.price = 1400000;

santro.wheel = 4;

santro.price = 850000;

santro.color = "Black";

innova.carInfo("Innova");

santro.carInfo("Santro");

}

}

**Constructor** : Constructor is a type of special method which help to create the object.

Pts

1. Constructor have same name as class itself.
2. Constructor doesn’t contains return type not even void also.
3. Constructor no need to call explicitly it will call automatically when you create the object.

Simple constructor example

class Car {

Car() {

System.out.println("Object created...");

}

void carInfo() {

System.out.println("Car Info Method");

}

}

class App {

public static void main(String args[]) {

Car obj1 = new Car();

obj1.carInfo();

}

}

**Difference between constructor Vs Methods**

In the life of the object if we want to perform any task only one time. That type of task we have to write inside a constructor (empty or parametrized).

In the life the object if we want to execute any task more than one time that type of task we have to write inside a methods.

Constructor and Method example

class Cal {

int a,b,sum;

Cal() {

a=1;

b=2;

}

Cal(int x, int y) {

a=x;

b=y;

}

void setValue(int x, int y) {

a =x;

b = y;

}

void add() {

sum = a+b;

}

void display() {

System.out.println("Sum "+sum);

}

}

class App {

public static void main(String args[]) {

Cal c1 = new Cal(); c1.add(); c1.display();

Cal c2 = new Cal(); c2.add(); c2.display();

Cal c3 = new Cal(); c3.display();

Cal c4 = new Cal(100,200); c4.add(); c4.display();

Cal c5 = new Cal(10,20); c5.add(); c5.display();

Cal c6 = new Cal(); c6.setValue(11,22); c6.setValue(111,222); c6.add(); c6.setValue(1111,2222); c6.display();

}

}

**Assignment 3**

Create EmployeeDetails class with 4 instance array variables.

EmployeeDetails() : memory size for array id,name,salary, desg must assign in constructor at run time.

read()

read all employee id,name,salary,desg

calSalary()

hra, da, pf local variables.

calculate salary

bonus()

apply bonus

display()

display details

EmployeeTest :

Main methods

Object creation

And calling all methods

Take n number records through keyboards as Id,Name,Salary,Desg

(array id,name,salary,Desg).

Salary = salary + hra + da – pf;

Hra is 10% salary

Da is 7 % salary

Pf 5 % salary

If desg is manager

15% bonus

If developer 10% bonus

Else

5 % bonus

Id, name, salary( grossSalary +bonus ) and desg

**Create the Folder**

**Then create simple text file with message.**

**Right click in folder and click on git bash (terminal open for you)**

**git init**

**git status**

**git add filename.txt**

**or**

**git add .**

**git status**

**git commit –m “File added”**

**git status**

git config [user.email akash3000383@gmail.com](mailto:user.email%20akash3000383@gmail.com)

git config [user.name Akash](mailto:user.email=akash3000383@gmail.com)

git remote add origin https://github.com/Kaleakash/fullstackassignment.git

git push -u origin main

git push –u origin master

First time

Create folder

add file or assignment file

open git bash terminal

git init (first time)

git add filename

git add .

git commit –m “First time added file”

git remote add origin https://github.com/Kaleakash/fullstackassignment.git

git push –u origin main/master

again and again whenever you add new file

open git bash terminal

git status --🡪 red

**git add . (add all file in staging area)**

**or**

**git add filename.txt (specific file)**

**git status**

**git commit –m “1st day assignment done”**

**git push –u origin master/main**

[**akash300383@gmail.com**](mailto:akash300383@gmail.com)

**share you gitURL**

**Day 4**

**15/01/2020**

**this :** this is a keyword which refer to current object.

1. When local variable or parameter variable and instance variable have same name. The local variable or parameter variable hide the visibility of instance variable.

this.variableName = variableName; it may be in constructor or methods.

**Example of this keywords**

class Employee {

int id;

String name;

float salary;

Employee() {

id =123;

name = "Unknown";

salary = 8000;

}

Employee(int id, String name, float salary) {

this.id = id; //local = local

this.name = name;

this.salary = salary;

}

void setValue(int id, String name, float salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

void display() {

System.out.println("Id is "+id);

System.out.println("Name is "+name);

System.out.println("Salary is "+salary);

}

}

class Demo {

public static void main(String args[]) {

Employee emp1 = new Employee(); emp1.display();

Employee emp2 = new Employee(); emp2.display();

Employee emp3 = new Employee(100,"Ravi",12000); emp3.display();

}

}

**Encapsulation :** Binding or Wrapper data(variables/fields) and code(methods/functions) in a single unit is known as Encapsulation.

Security

class :

**Example**

class Employee {

private int id;

private String name;

private float salary;

void setValue(int id, String name, float salary) { //helper methods

this.id = id;

this.name = name;

//this.salary = salary;

if(salary<0) {

this.salary = 8000;

}else {

this.salary = salary;

}

}

void display() {

System.out.println("Id is "+id);

System.out.println("Name is "+name);

System.out.println("Salary is "+salary);

}

}

class Demo {

public static void main(String args[]) {

Employee emp = new Employee();

//emp.id = 123;

//emp.name="Ravi";

//emp.salary = -12000;

emp.setValue(123,"Ravi",-12000);

emp.display();

}

}

**Inheritance :** Inheritance is use to inherits or acquire properties and behaviour of old class to new class.

class OldClass { super class, base class or parent class.

properties

behaviour

}

class NewClass extends OldClass{ sub class, derived class, child class.

properties

behaviour

}

**Simple Inheritance example**

class A {

void dis1() {

System.out.println("A class method");

}

}

class B extends A {

void dis2() {

System.out.println("B class method");

}

}

class Demo {

public static void main(String args[]) {

A obj1 = new A();

obj1.dis1();

B obj2 = new B();

obj2.dis2();

obj2.dis1();

}

}

**Type of Inheritance**

1. Single Inheritance :

One super class and one sub class

class A {}

class B extends A {}

1. Multilevel Inheritance

One super class and n number of sub class extends one by one

class A {}

class B extends A {}

class C extends B {}

class D extends C {}

1. Hierarchical Inheritance

One super class and n number of sub classes extends directly to super class.

class A {}

class B extends A {}

class C extends A {}

class D extends A {}

1. Multiple Inheritance

More than one super class and one sub class

class A {}

class B {}

class C extends A,B {} But Java Doesn’t support. This type of inheritance java support indirectly using interface but not through class.

**OOPs Relationship**

1. Is a relationship
2. Has a relationship

**Assignment 4**

Manager / Programmer is a Employee

Employee has a Address

class Employee { super class must be generic

id,name,salary

Scanner obj = new Scanner(System.in);

Address add = new Address();

methods

read() id,name,salary

, calSalary() : hra, da and pf

dislay()

}

class Manager extends Employee { sub must be specific

numberOfEmp : numbers

readMgr() numberOfEmp

add.readAdd();

disMgr();

numberOfEmp

add.disAdd();

}

class Programmer extends Employee{ sub must be specific

projectName: string

readPrg()

projectName

add.readAdd();

disPrg();

}

class Address {

city, state, pinCode

Scanner obj = new Scanner();

readAdd()

read city,state and pincode

disAdd();

city, state and pincode

}

EmployeeTest

Main Don’t create the Employee class object.

S.O.P(ManagerDetails);

Manager mgr

mgr.read() 3 details

mgr.readMgr() 1 own details, 3 address details

S.O.P(ProgramerDetails)

Programmer prg

prg.read()

prg.readPrg()

mgr.calSalary();

prg.calSalary()

Display Manager and Programmer details.

**Polymorphism:**

One name many forms or many implementation

2 types

**Compile time polymorphism Run time polymorphism**

Static binding Dynamic binding

Early binding Late binding

Example Example

Method Overloading Method Overriding

**Method overloading :** The method have same name but different parameter list (type of parameter list or number parameter list) but return type must same.

Example

class Abc {

void add(int x, int y) {

System.out.println(x+y);

}

void add(int x, int y, int z) {

System.out.println(x+y+z);

}

void add(float x, float y) {

System.out.println(x+y);

}

void add(String x, String y) {

System.out.println(x+y);

}

}

class Demo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.add(1,2); //compile time binding

obj1.add("1","2");

obj1.add(10.10f,20.20f);

obj1.add(1,2,3);

}

}

**JDBC**

URL,

URL,UserName

URL,username,password

DriverManager.getConnection(url);

DriverManager.getConnection(url,username);

DriverManager.getConnection(url,username,password);

**Runtime Polymorphism :** Method Overriding :

The method have same name and same method signature (number of parameter list, type of parameter list as well as return type must be same).

To achieve the method overriding the class must in Inheritance.

Super class and sub class.

**Method Overriding Example**

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

void speed() {

System.out.println("90km/hr");

}

}

class Honda extends Bike {

void color() {

System.out.println("Gray");

}

}

class Demo {

public static void main(String args[]) {

Pulsar pu = new Pulsar();

pu.color(); pu.speed();

Honda hh = new Honda();

hh.color(); hh.speed();

}

}

**Annotation :** Annotation is meta-data. Meta-data means data about data.

int a;

Java provide lot of pre-defined annotation as well as we can create user defined annotation. Some annotation we can use on class level or methods level or constructor level or property level (primitive or complex property).

All annotation start with @ followed by annotation name.

**@Override :** it is pre-defined annotation we can use on method level on sub class method if sub class method overriding super class method then we will not get any error else it will through error at the compile time.

**Method overriding, reusability, merge method example**

class Bike {

void speed() {

System.out.println("60km/hr");

}

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

@Override

void speed() {

System.out.println("90km/hr");

}

}

class Honda extends Bike {

void color() {

System.out.println("Gray");

}

}

class Tvs extends Bike {

void color() {

System.out.println("White");

}

@Override

void speed() {

super.speed(); //callng super class speed meth

System.out.println("20km/hr");

}

}

class Demo {

public static void main(String args[]) {

Pulsar pu = new Pulsar(); pu.color(); pu.speed();

Honda hh = new Honda(); hh.color(); hh.speed();

Tvs tv = new Tvs(); tv.color(); tv.speed();

}

}

**abstract**

abstract is a keyword or non access specifiers. Which we can use with method and class but not with variable.

1. abstract method: the method without body or without curly braces or incomplete method is known as abstract methods.

Syntax

abstract returnType methodName(parameterList);

1. abstract class : if class contains abstract method then we have to declare the class as abstract class.

Syntax

abstract class className {

}

1. which ever class extends abstract class that class must be provide the body for all abstract methods mandatory. That class can ignore only if that class itself is a abstract class.
2. Abstract class we can’t create the object.
3. Abstract class can contains normal methods as well as abstract methods.

It can contains zero or 1 or all abstract methods.

1. Some time class is abstract but no abstract method. (If we don’t want to create the object of that class directly then we can declare the class as abstract class without abstract methods).

If we want to give instruction to not allow to create the object of that class.

1. Abstract class can contains default constructor we well as parameterized constructor because abstract class contains instance variable (inside constructor only default initialization happen).

A abstract 1 or more

B extends A abstract 1 or more :

C extends B normal must be provide body for A and B class methods.

**Abstract class example**

abstract class Bike {

int wheel;

Bike() {

wheel = 4;

}

abstract void speed();

}

class Pulsar extends Bike {

void color() {

System.out.println("Black");

}

void speed() {

int abc=20;

System.out.println("Wheel "+wheel);

System.out.println("90km/hr");

}

}

class Demo {

public static void main(String args[]) {

Pulsar pu = new Pulsar(); pu.color(); pu.speed();

}

}

**Final keyword :** Final is a keyword or non access specifiers we can use with variable, method and class.

1. Final variable : To declare constant variable in java we use final keyword with variable

final int A=10; : we have to initializes and we can’t change the value final variable. Final variable must be upper case.

1. Final method: if the method final we can’t override that method but we can call or use that method.
2. Final class : if class is final we can’t inherits or extends final class.

**Final example**

final class A {

final void dis1() {

System.out.println("A class method");

}

}

class B extends A {

/\*void dis1() {

System.out.println("A class method override by B class");

}\*/

}

class Demo {

public static void main(String args[]) {

final int A=10;

System.out.println(A);

//A=20;

System.out.println(A);

B obj1 = new B();

obj1.dis1();

}

}

**static :** static is a keyword or non access specifiers we can use with variable and method but not with class (Inner class. If class is inner we can use static keyword but not for outer class).

1. If variable is static we can assign the value for that variable using class name.

className.staticVariablename = value;

1. If method is static we can call that method with the help of className.

className.staticmethodName();

1. Even though we can assign the value for static variable with the help of object also as well as we can call static method with help of object also.
2. Inside a non static method we can access non static as well as static variable directly of same class.
3. Inside static method we can access only static variable directly. Non static we can’t access directly.

**Static example**

class Abc {

int a;

static int b;

void dis1() {

System.out.println("Non static method");

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

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

}

static void dis2() {

System.out.println("Static method");

Abc obj = new Abc();

System.out.println("a "+obj.a);

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

}

}

class Demo {

public static void main(String args[]) {

Abc obj1 = new Abc();

obj1.dis1();

obj1.a=100;

Abc.b=200;

obj1.dis1();

Abc.dis2();

obj1.b=300;

obj1.dis2();

}

}

Every class contains only one static memory. Every class contains n number of heap memory or instance number base upon the number of object created.



**Static and heap memory example**

class Abc {

int a;

static int b;

void dis1() {

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

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

}

/\*static void dis2() {

System.out.println("Static Method");

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

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

}\*/

}

class Demo {

public static void main(String args[]) {

//Abc.dis2();

Abc obj1 = new Abc(); //heap memory

Abc obj2 = new Abc();

obj1.a=10;

obj1.b=20;

Abc.b=30;

obj2.a=40;

obj2.b=50;

Abc.b=60;

obj1.dis1(); //a=10 , b=60

obj2.dis1(); //a=40 ,b=60

}

}

**Initialization and static block:**

Example

class A {

A() {

System.out.println("A class constructor");

}

void dis1() {

System.out.println("A class method ");

}

{

System.out.println("Initialization block 1");

}

static {

System.out.println("static block");

}

{

System.out.println("Initialization block 2");

}

}

class Demo {

public static void main(String args[]) {

A obj1 =new A();

obj1.dis1();

obj1.dis1();

A obj2 = new A();

obj2.dis1();

obj2.dis1();

}

}

**Day 5**

**16/01/2020**

By default git consider as master as a default branch.

Master branch is replace by main branch

**Branch : It is like a pointer which hold more than one commit details.**

My Remote repository default branch is : main / master

My Local repository default branch is : main / master

Two options to create local repository

1. **git init** : The folder become local repository with empty files.
2. **git clone** :

**Day 6**

**18/01/2020**

**Interface :** Interface is a type of reference data type or also known as 100% pure abstract class (till Java 7 Version).

Syntax

interface interfaceName {

fields;

methods;

}

By default all fields in interface are public static and final.

By default all methods are public and abstract.

interface Abc {

public static final int A=10;

static final int B=20;

final int C=30;

int D=40;

public abstract void dis1();

abstract void dis2();

public void dis3();

void dis4();

}

interface Abc {

int A=10;

void dis1();

}

interface Mno {

int B=20;

void dis2();

}

interface Xyz extends Abc,Mno {

int C=30;

void dis3();

}

class Sample implements Abc,Mno {

}

Like a class interface can extends another interface. Interface can extends more than one interface.

Class always implements interface. Class can implement more than one interface.

Which ever class implements interface that class must be provide the body for all abstract method belong to that interface. That class can ignore only if that class is a type of abstract class.

**Access specifiers for method overriding**

Super (class/interface) Sub class

public public

protected public

protected

default public

protected

default

private We can’t override

Interface Example

interface Abc {

int A=10;

void dis1();

}

interface Mno {

int B=20;

void dis2();

}

interface Xyz extends Abc,Mno {

int C=30;

void dis3();

}

class Sample implements Abc,Mno {

public void dis1() {

System.out.println("Abc interface Method");

}

public void dis2() {

System.out.println("Mno interface Method");

}

}

class Demo {

public static void main(String args[]) {

Sample s = new Sample();

s.dis1();

s.dis2();

}

}

Points

1. class extends class (only one )
2. interface extends interface (more than one)
3. class implements interface (more than one)
4. interface doesn’t extends or implements to class.

**Array object :**

Syntax

class Employee{

id,name,salary

}

Employee emp = new Employee();

className refereceName[]=new className[size];

Employee employees[]=new Employee[100];

int abc[]=new int[10];

employees :

0

1

100

99

**Assignment 5:**

Create Student class which contains sId,SName,Age,Marks[](PCMB) ,Grade(char)

Scanner obj

read()

take

id, name,age

for(i=0;iM<marks.length;i++) {

marks[i]=obj.next()

}

calculateGrade()

local variable total,avg;

avg > 90 A+ avg >80 A

avg >70 B avg >55 C

else

D

display

id,name,age,Grade

StudentTest

Create Student array object

How many student details do you want to store.

Student std[]=new Students[n];

for(int i=0;i<n;i++) {

std[i]=new Student();

std[i].read();

}

for(int i=0;i<i++){

std[i].calculateGrade();

}

for(int i=0;i<i++){

std[i].display();

}

Difference between Abstract class and interface.

Interface

1. Interface contains only final variable.
2. Interface contains only abstract method.
3. Interfaces doesn’t contains default constructor as well as we can’t write parameterized constructor.
4. Normal or abstract class can implements more than one interface.
5. Using interface we can achieve full abstraction.

Abstract class

1. Abstract class contains final as well as normal variables.
2. Abstract class contain normal as well as abstract methods.
3. Abstract class can contains default constructor as well as we can write parameterized constructor.
4. Abstract class or normal class can extends only one abstract class.
5. Using abstract class we can achieve partial abstraction.

Common we can’t create the object of interface as well abstract class.

Whichever class extends or implements abstract class or interface that class must be provide the body for all abstract method belong to that class or interface.

**Assignment 6:**

**Assignment 6: Update the Code**

**First Change**

**Create user-defined exception**

1. **MinimumAccountBalance**
2. **InValidAccountNumber (Transfer, Withdraw, deposit)**

**Second Changes : you have to use package concept.**

Account 🡪 bean package

AccountInitialization, 🡪 service package

Bank 🡪 service package

MyBank 🡪 service package

BankTest 🡪 main / test package

**Small Mini Projects**

**Menu Driven Application**

Banking Application

1. Create Account
   1. Default Details : 1010,1011,1012 Unknown, 500
   2. Name and Amount : 1013,Raj, 600 amount >= 500
2. Check Account Balance
3. Withdraw the amount
4. Deposit
5. Transfer

abstract class AccountInitialization {

static int accountCount=0;

Account accounts[]=new Account[10];

Start with 1010 (account number )

void accountCreate() {

account Number must be unique

name =”Unknown”

amount =500;

Account ac = new Account(accno,Unknown,500);

accounts[acountCount]

accountCount++;

}

void accountCreate(name,amount) {

accno must be unique

amount > 500 then crate account

assign name and amount

Account acc = new Account(uniqueNumber);

Call helper to set name and amount;

accounts[acountCount]

accountCount++;

}

abstract void transfer(fromaccNo,toAccNo,amount);

}

interface Bank

void withdraw : accno, amount

void deposit : accno, amount

void checkBalance : accno

all abstract methods

class Account

private Accno

private Name

private Amount it must be private

write constructor empty or parameterized.

helper method set the value.

setAccno,setName,setAmount

getAccno, getName,getAmount

class MyBank extends AccountInitialization implements Bank {

void transfer(fromaccNo,toAccNo,amount) {

fromAccNum : 1010

toAccoNumber : 1011

amount must be +ve

fromAccoNumber may not present

toAccoNumber may not present

Amount is not available

Maintain min 500 in fromAccount Number

}

Withdraw {

Accno

AccountNumber may be not present

Amount

Amount is not available

Maintain min 500 in fromAccount Number

}

Deposit {

Accno

AccountNumber may be not present

Amount

Amount is not available

Can’t deposit more than 50,000/-

}

Checkbalance {

Using account Number

}

BankTest App

Main method

Do {

Switch() {

1. Create Account
   1. Default details
   2. Name and Amount pass;
2. Check Account Balance
3. Withdraw the amount
4. Deposit
5. Transfer
6. Exit the Application

}

Do want to continue.

}while(condition)

Welcome, thank

**this, this(), super, super()**

this is use to refer the current object.

When local variable or parameter variable and instance variable have same name then local variable hide the visibility of instance variable. To refer the instance variable we use

this.variableName = variableName;

When super class variable and sub class variable have same name sub class variable hide the visibility of super class variable. To super superclass variable we use

super.variableName;

**this and super keywords example**

class A {

int a=10;

}

class B extends A {

int a=20;

void dis() {

int a=30;

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

System.out.println("a - instance variable "+this.a);

System.out.println("a - super class variable "+super.a);

}

}

class Demo {

public static void main(String args[]) {

B obj = new B();

obj.dis();

}

}

this(): It is use to invoke more than one constructor of same class or also known as constructor chaining of same class without creating fresh memory.

this() or this(parameter) it must be inside a constructor and it must be first statement inside a constructor.

**this() example**

class Employee {

Employee() {

System.out.println("Employee()");

}

Employee(int id) {

this(); //calling constructor

System.out.println("Employee(int)");

}

}

class Demo {

public static void main(String args[]) {

//Employee emp1 = new Employee();

Employee emp2 = new Employee(10);

}

}

**super()**

In Inheritance sub class can inherits only instance variable and non static method. It can’t inherits constructor as well as static property/methods.

By default every sub class constructor **super()** is available. It help to invoke sub class constructor to super class constructor (constructor chaining from sub to super class).

It must be first statement inside a constructor.

class Employee {

Employee() {

System.out.println("Employee()");

}

Employee(int id) {

System.out.println("Employee(int)");

}

}

class Manager extends Employee {

Manager() {

super(100); //by defaut

System.out.println("Manager()");

}

}

class Demo {

public static void main(String args[]) {

Manager mgr = new Manager();

}

}

/\*

Manager()

Manager(), Employee()

Employee(), Manager()

Employee();

\*/

**this() and super() example**

class Employee {

private int id;

private String name;

private float salary;

Employee() {

this.id = 123;

this.name = "Unknown";

this.salary = 8000;

}

Employee(int id) {

this();

this.id = id;

}

Employee(int id, String name) {

//this.id = id;

this(id);

this.name = name;

}

Employee(int id, String name, float salary) {

//this.id = id;

//this.name = name;

this(id,name);

this.salary = salary;

}

void disEmp() {

System.out.println("id is "+id+"Name is "+name+" Salary is "+salary);

System.out.println("Number of rec "+Employee.rec);

}

static int rec;

void setRecordsDetails(int rec) {

Employee.rec = rec;

}

}

class Manager extends Employee {

int numberOfEmp;

Manager(int id, String name, float salary, int numberOfEmp) {

super(id,name,salary);

this.numberOfEmp = numberOfEmp;

}

void disMgr() {

System.out.println("Number of emp is "+numberOfEmp);

}

}

class Demo {

public static void main(String args[]) {

int record=10;

Manager mgr = new Manager(1,"Ravi",45000,5);

mgr.setRecordsDetails(record);

mgr.disEmp();

mgr.disMgr();

}

}

**Run time polymorphism using object creation**

class A {

void dis1() {

System.out.println("A class dis1() method");

}

}

class B extends A {

void dis1() {

System.out.println("A class dis() method override by B class");

}

void dis2() {

System.out.println("B class dis2() method");

}

}

class Demo {

public static void main(String args[]) {

A obj1 = new A(); //1 Creating super class object.

obj1.dis1();

B obj2 = new B(); //2 Creating sub class object

obj2.dis1(); obj2.dis2();

//B obj3 = new A(); //3 Super class object and Sub class reference not possible

A obj4 = new B(); //4 Sub class object and super class reference with help of super class references we can

// call only those methods which is a part of super class and overrided methods. This is also known as

//Run time polymorphism.

obj4.dis1(); //5

//obj4.dis2(); //6

B obj5 = (B)obj4; //down level type casting..

obj5.dis1();

obj5.dis2();

}

}

/\*

Valid Object creation in Java.

A. line 5 error

B line 6 error

C no error

D line 5 and 6 error

A. all four

B. 1,2,3

C. 1,2,4

D,1,2

\*/

Another Example using Abstract class

abstract class A {

abstract void dis1();

}

class B extends A {

void dis1() {

System.out.println("A class dis() method override by B class");

}

void dis2() {

System.out.println("B class dis2() method");

}

}

class Demo {

public static void main(String args[]) {

A obj4 = new B(); // sub class object and super class reference may super class normal class or abstract class.

obj4.dis1();

}

}

Interface reference

interface A {

void dis1();

}

class B implements A {

public void dis1() {

System.out.println("A interface dis() method override by B class");

}

void dis2() {

System.out.println("B class dis2() method");

}

}

class Demo {

public static void main(String args[]) {

A obj4 = new B(); // sub class object and interface reference. Very Imp

obj4.dis1();

}

}

**Abstraction** : Hiding the internal implementation without knowing background details.

**100% pure abstraction example**

interface A {

int add(int x, int y);

}

interface B {

int sub(int x, int y);

}

class Server implements A,B{

public int add(int x, int y) {

return x+y;

}

public int sub(int x, int y) {

return x-y;

}

public void ownMethod() {

System.out.println("Own method");

}

}

class Demo {

public static void main(String args[]) {

Server s = new Server();

System.out.println(s.add(10,20));

System.out.println(s.sub(10,20));

s.ownMethod();

A obj1 = new Server(); Run time polymorphism

System.out.println(obj1.add(10,20));

//System.out.println(obj1.sub(10,20));

B obj2 = new Server(); Run time polymorphism

//System.out.println(obj2.add(10,20));

System.out.println(obj2.sub(10,20));

}

}

**packages :**

package is a collection of classes and interfaces.

2 types

1. User-defined package
2. Pre-defined package

Education

School College Pg

Attendance Attendance Attendance

Package is like a directory or folder which more than one class/interface have same but different purpose.

Syntax

package packgename;

package com;

class Demo {

public static void main(String args[]) {

System.out.println("Welcome to User defined package");

}

}

javac Demo.java

create directory with packageName and paste .class file

and run the command as

java com.Demo

or

javac –d . Demo.java

java com.Demo

IDE :

Netbean

Eclipse

MyEclipse

RAD

Etc

**Eclipse for JEE**

**Access Specifiers :** It use to give the visibility or accessibility of class, variable, method etc.

**private :**

**Using with :**

Instance variable, static variable, non static method, static method, constructor but not with local variable and class.

**Scope :**

Within a same class. (other class we can’t access directly as well as through object).

**default (nothing):**

**Using with :**

We can use with all.

**Scope :**

With a same package. May be non class is non sub class or sub class we can access through object.

**protected :**

**Using with :**

Instance variable, static variable, non static method, static method, constructor but not with local variable and class.

**Scope :**

Within a same package. May be non sub class or sub class other package if it is sub class.

**public :**

**Using with :**

Instance variable, static variable, non static method, static method, constructor, class but not with local variable.

**Scope :**

Within a same package as well as other package.

**Pre-defined package or built in package**

java javax-🡪 root package

lang sql

io swing

util net

sql rmi

net servlet

awt ejb

rmi jms

etc etc

Every package contains lot of classes, interface as well as other sub package.

By default every java program import lang package.

By default every java program extends Object class.

Class part of lang ( Name of the class itself is **Class)**

**class Class {**

**}**

This class is also part of lang package.

**Exception Handling**

**Exception** : Exception is object (memory) which occurs when unexpected or abnormal condition during the execution of a program.

**Java Program**

javac command java run

compile time error run time error

syntax error Error Exception

or

typo error

**Command Line Arguments**

class ExpDemo {

public static void main(String args[]) {

System.out.println("Hi");

System.out.println(args[0]);

for(String str:args) {

System.out.println(str);

}

}

public static void main(int args[]) {

System.out.println("Hello");

}

}

Run time error

Error Exception

**Error** : it is a type of error which generate at the run time which we can’t handle it. Ex : JVM Crash, Software issue or hardware issue, Out of memory.

**Exception :** it is a type of error which generated at the run time which we can handle it. Ex : divided zero, array index, null pointer etc

Error and Exception both are pre-defined classes part of lang package.

Object

Throwable

**Error** **Exception**

**Exception**

Group Group

**Checked Exception Unchecked Exception**

**RuntimeException** : pre-defined class

SQLException ArithmeticException

IOException ArrayIndexOutOfBoundsException

InterruptedException NumberFormatException

FileNotFoundException ClassNotFoundException

NullPointerException

InputMismatchException

Etc etc

All checked exception directly or indirectly connected / extends to **Exception** class.

All unchecked exception directly or indirectly connected / extends to Exception class through **RuntimeException**

All unchecked exception sub class RuntimeException class.

To handle both the type of exception java provide five keywords.

1. try
2. catch
3. finally
4. throw
5. throws

try catch block

try {

The code generate exception 1 or more than one line

}catch(Exception e) {

}

try {

1

2

3

4

5

6

7

8

9

10

}catch(Exception e) {

}

try {

1

2

3

4

5

}catch(Exception e) {

}

try {

5

6

7

8

9

10

}catch(Exception e) {

}

Try with single catch : Generic exception and generic solution.

**try {**

**}catch(Exception e){**

**}**

Try with multiple catch block : Specific exception specific treatment.

**try {**

**}catch(ArithmeticExeption e) {**

**}catch(ArrayIndexOutOfBoundsException e) {**

**}catch(NumberFormatException e){**

**}**

**Day 7**

**19/01/2020**

**Finally block :** This block will execute 100% sure if any exception generate or not.

try {

}catch(Exception e){

}finally {

}

Finally block example

class ExpDemo {

public static void main(String args[]) {

System.out.println("Hi");

try {

int res = 10/1;

System.out.println("No Exception");

}catch(Exception e){

System.out.println("Catch block");

}finally {

System.out.println("finally block");

}

System.out.println("Normal Statement");

}

}

Try catch and finally

1. try – catch
2. try – catch –catch –catch
3. try – catch – finally
4. try – catch – catch – catch – finally
5. try – finally

**Finally example**

class ExpDemo {

public static void main(String args[]) {

System.out.println("Hi");

try {

int res = 10/0;

System.out.println("No Exception");

}finally {

System.out.println("finally block");

}

System.out.println("Normal Statement");

}

}

try {

open file / open database connection

read or write or store or retrieve

}catch(Exception e){

}finally {

close file or close database connection to close resources

}

**throw :** this keyword is use to raise or generate pre-defined or user-defined exception depending upon the conditions.

Syntax

**throw new Exception();**

or

**throw new ExceptionSubClass();**

throw Example

class MyException extends Exception {

MyException() {

super();

}

MyException(String msg) {

super(msg);

//System.out.println("Pameterized constructor");

}

}

class ExpDemo {

public static void main(String args[]) {

int a=10;

int b=5;

try {

//int res = 10/0;

if(a>b) {

//throw new Exception(); // generating exception;

//throw new ArithmeticException();

//throw new ArithmeticException("a>b");

//throw new MyException();

throw new MyException("a is > than b ");

}else {

}

}catch(Exception e) {

System.out.println("Catch Block");

System.out.println(e.toString());

}

System.out.println("Normal Statement");

}

}

**throws :** throws keyword is use to throw exception to caller methods.

Syntax

returnType methodName(parameterList) throws Exception {

}

void dis() throws Exception,ExceptionSubClass {

}

**throws Exception example**

class ExpDemo {

static void dis1() throws ArithmeticException {

try{

int a=10/0;

}catch(ArrayIndexOutOfBoundsException e){}

System.out.println("dis1 method");

}

static void dis2() throws Exception{

//try{

dis1();

//}catch(Exception e){}

System.out.println("dis2 method");

}

public static void main(String args[]) throws Exception{

//try{

dis2();

//}catch(Exception e){}

System.out.println("Main method");

}

}

**Checked Exception**

InterruptedException it is type of checked exception we have handle it or throw it using try catch or throws.

Checked exception check twice compile time or run time.

class ExpDemo {

public static void main(String args[]) throws Exception{

System.out.println("Hi");

//try{

Thread.sleep(3000);

//}catch(Exception e){}

System.out.println("Take Tea Break..");

}

}

**Lang package**

By default every java program import lang package.

By default every java class extends Object class.

Imp class and interfaces from lang package

1. String
2. StringBuffer
3. StringBuilder
4. Math
   1. Exception and Type of Exception classes
5. Object
6. Thread
7. Runnable interface
8. Cloneable interface.
9. Wrapper classes

String : In Java String is a reference data type or pre-defined class part of lang package.

In Java every string not end with null character (\n).

Syntax

String str1 = “Welcome to Java Training”; // using string literal

String str2 = new String(“Welcome to Java Training”); //using new keyword.

Assignment 7 : Take n number of names through keywords and display those names in ascending or descending order. (Ignore case sensitive).

String names[]={“Raj”,”ravi”,”Akash”,”ajay”};

Akash, ajay, Raj, ravi 🡪Asc

ravi, Raj, ajay, Akash 🡪Desc

Hint : compareTo() methods or any other ways.

String class Example

**package** com;

**public** **class** StringDemo {

**public** **static** **void** main(String[] args) {

String str1 = "Welcome to Java Training";

String str2 = **new** String("Welcome to Java Training");

System.***out***.println(str1);

System.***out***.println(str2);

System.***out***.println(str1.charAt(1));

System.***out***.println(str1.toLowerCase());

System.***out***.println(str1.toUpperCase());

System.***out***.println(str1.length());

System.***out***.println(str1.substring(5));

System.***out***.println(str1.substring(5, 10)); //start 0 index, end 1 position,

System.***out***.println(str1.compareTo(str2));

}

}

compareTo() : This method check both string ASCI code.



== : It always check value as well as reference code of that memory.

equals() : it is use to check the value of two reference. It doesn’t matter same memory or different memory.

String class equals methods example

**package** com;

**public** **class** StringEquals {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

String str1 = "Raj"; // string literal

String str2 = "Raj"; // refer to same memory.

String str3 = **new** String("Raj"); // new memory

String str4 = **new** String("Raj"); //new memory

**if**(str3.equals(str4)) {

System.***out***.println("Equal");

}**else** {

System.***out***.println("Not Equal");

}

}

}

String class is known as immutable class. Immutable means can’t change.

String class is immutable string class.

We can create the object of string using string literal or using new keyword.

StringBuffer is mutable string class.

Using new keyword.

**StringBuffer Example**

**package** com;

**public** **class** StringImmutable {

**public** **static** **void** main(String[] args) {

String name="Ravi";

System.***out***.println(name); //Ravi

name = name+" Kumar";

System.***out***.println(name); //Ravi or Ravi Kumar

System.***out***.println(name.toUpperCase());

System.***out***.println(name);

System.***out***.println("Using String Buffer");

StringBuffer sb = **new** StringBuffer("Ravi");

System.***out***.println(sb);

System.***out***.println(sb.append(" Kumar"));

sb.insert(4, " Patil");

sb.delete(2, 4);

sb.reverse();

System.***out***.println(sb);

}

}

StringBuilder : It is a type of String mutable class.

StringBuffer class maximum methods are synchronized they are thread safe but slow is performance. StringBuilder methods are not synchronized they are not thread safe but very fast.

**Math:** Math class also known as static class. (When all variable and methods are static that type of class is know static class).

This class use to do some mathematical operation.

**Math.PI**

**Math.sqrt(9);**

**Object class Example**

By default every class extends Object class.

This command is to display all method part of that specific class or interface.

javap java.packageName.className/interfaceName

getClass(); : The return type of methods method is Class class reference.

hashCode();

equals(); Collection Framework

clone();

toString();

wait();

notify();

notifyAll() : Multithreading

finalize()

**JavaBean :**

1. Class must be public
2. Variable must be private.
3. For each variable we have to provide setter and getter methods.
4. Setter method to set the value and getter method to get the value.
5. Insider a setter method we can set the value with terms and conditions.

class Employee {

private int id;

private String name;

private float salary;

public void setId(int id) {

this.id = id;

}

public int getId() {

return id;

}

}

**Normal class :**

1. Class may be public or may not.
2. Instance variable may be private or may not.
3. If variable are private then we have to provide helper method to set the value. The methods name may be anything.

toString() : When we display any user defined class reference in println it internally call toString() method of Object class. That class toString() method return object in String format.

finalize() : This method get call automatically before GC. (Pre-Destructor methods).

**package** com;

**public** **class** Manager {

**void** dis() {

System.out.println("dis method");

}

@Override

**protected** **void** finalize() **throws** Throwable {

// **TODO** Auto-generated method stub

System.out.println("Before GC");

}

}

In main method

Manager mgr = **new** Manager();

mgr.dis();

mgr = **null**; //GC

System.*gc*(); // WE can request to JVM for GC

System.***out***.println("Bye..");

System.***out***.println("Bye...");

**clone :** This method is use to create the clone of the object or duplicate object.

If we are planning to create the clone.

We have to make that class must be implements Cloneable interface part of lang package.

Cloneable is a marker interface.

Marker interfaces doesn’t contains any methods or Zero methods.

clone() : this method is part of object class which help to create to clean. Cloneable interface which support to create the clone.

**Day 8**

**20/01/2020**

**Util Package**

Utility :

Collection Framework (Data Structure) :

int a=10;

array

int abc[];

class Employee {

id,name,salary

}

Employee emp =new Employee();

Array object

Employee emps[]=new Employee[100];

Array memory size fixed.

Emps array can hold only same type of objects.

Primitive array or array object doesn’t provide any pre-defined methods which help to add, remove, search, iterate etc

Collection Framework : It contains set of classes and interfaces which provides set methods with help of those methods we can add any objects or elements (any primitive type (Wrapper classes)) as well as user-defined objects. We can add, remove, search, iterated etc. Memory size no issue.

Collection Framework Hierarchy

javap java.util.List

Iterable -🡪 lang package

extends

Collection -🡪 interface -🡪 util package

extends extends extends doesn’t

List Set Queue Map 🡪 interfaces

Hold collection Collection Collection Key-value pairs

Of elements of elements of where key is Unique

It can allow It doesn’t elements value may be duplicate.

Duplicate elements allow duplicate. FIFO.

Maintain order Some API

Using index position maintains

Order, UnOrder

Sorted.

Deque (interface)

ArrayList HashSet PriorityQueue HashMap

LinkedList LinkedHashSet LinkedHashMap

Vector TreeSet TreeMap

Stack Hashtable

ArrayList :

Normal array :

Normal array is use to store same type of value.

Fixed memory size.

Adding and removing element from array complexity.

ArrayList :

By default ArrayList help use to store same as well as different type of values.

Memory size get increase dynamically.

We can add or remove elements very easily.

**ArrayList Example**

import java.util.\*;

class A {}

class B {}

public class CollectionExpDemo {

public static void main(String args[]) {

ArrayList al = new ArrayList();

al.add(10);

al.add(10.10);

al.add("Ravi");

al.add(true);

A obj1 = new A();

al.add(obj1);

B obj2 = new B();

al.add(obj2);

System.out.println(al);

}

}

Collection of classes is use to store collection of element or objects.

Primitive Data types Wrapper classes

byte Byte

short Short

int Integer

long Long

float Float

double Double

char Character

boolean Boolean

Wrappers classes is use to do type casting.

With help of wrapper classes we can convert primitive to object and vice-versa.

Wrapper class Example

**int a=10;**

**Integer b = new Integer(a); // converting primitive to objects.**

**System.out.println(a);**

**System.out.println(b);**

**int c = b.intValue(); //type casting**

**float d = b.floatValue(); //type casting**

**Float e = b.floatVallue(); // auto-boxing**

**System.out.println(c);**

**System.out.println(d);**

**Auto – boxing**

import java.util.\*;

class A {}

class B {}

public class CollectionExpDemo {

public static void main(String args[]) {

ArrayList al = new ArrayList();

int a=10;

Integer b = new Integer(a);

al.add(b);

al.add(new Integer(100));

al.add(20); // auto-boxing : converting primitive to objects.

al.add(10.20); // auto-boxing : converting primitive to objects.

Integer c =30; //auto-boxing : converting primitive to Integer

System.out.println(c);

}

}

**Primitive Data types**

int a=10;

add, sub, mul, div, mod, >, <.

a is primitive

Wrapper classes

Integer b = new Integer(a);

b is a Integer object

**ArrayList methods**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

ArrayList al = new ArrayList();

al.add(100);

al.add(200);

al.add(300);

al.add(400);

al.add(500);

System.out.println(al);

System.out.println("First value "+al.get(0));

System.out.println("Second value "+al.get(1));

al.add(1,10);

System.out.println(al);

System.out.println("First value "+al.get(0));

System.out.println("Second value "+al.get(1));

al.remove(1); // remove using index position

al.remove(new Integer(400)); // remove using value.

System.out.println(al);

al.set(2,30);

System.out.println(al);

}

}

**LinkedList :** It is a type of List and Queue implementation class. In Java LinkedList Internally provide double linked list features.

Node ---🡪

Ref value

Ref value

Previous value Next

Single Linked List

Double Linked List

Circular Linked list

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

LinkedList al = new LinkedList();

al.add(10);

al.add(20);

al.add(30);

al.addFirst(100);

al.addLast(200);

al.add(1,123);

System.out.println(al);

System.out.println("Get "+al.get(0));

}

}

**Vector :** By default all methods in Vector are synchronized.

Synchronized : Slow in performance and single threaded (thread safe).

Work is safe.

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

Vector vv = new Vector();

vv.add(10);

vv.addElement(20);

System.out.println(vv);

}

}

**Stack :** Stack First In Last Out

Push() : add on top

Pop : retrieve top element

Search() : present or not

Peek() : top select check.

**Stack Example**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

Stack ss = new Stack();

ss.push(100);

ss.push(200);

ss.push(300);

ss.push(400);

System.out.println("Size "+ss.size());

System.out.println(ss);

System.out.println("Pop "+ss.pop());

System.out.println(ss);

System.out.println("Search "+ss.search(300));

System.out.println("Search "+ss.search(3000));

System.out.println("Peek "+ss.peek());

System.out.println(ss);

}

}

**Set :** It doesn’t allow duplicate.

HashSet

LinkedHashSet

TreeSet

Set doesn’t index concept. We have to remove or search element using value.

**HashSet Example**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

HashSet hs = new HashSet();

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

hs.add(3);

hs.add(5);

hs.add("Ravi");

hs.add(1);

hs.add(true);

hs.add(4);

hs.add(5);

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

System.out.println(hs);

hs.remove(5);

System.out.println(hs);

System.out.println("Search "+hs.contains(4));

System.out.println("Search "+hs.contains(10));

hs.clear();

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

System.out.println(hs);

}

}

**LinkedHashSet :** This class internally extends HashSet and maintains the orders.

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

LinkedHashSet hs = new LinkedHashSet();

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

hs.add(3);

hs.add(5);

hs.add("Ravi");

hs.add(1);

hs.add(true);

hs.add(4);

hs.add(5);

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

System.out.println(hs);

hs.remove(5);

System.out.println(hs);

System.out.println("Search "+hs.contains(4));

System.out.println("Search "+hs.contains(10));

hs.clear();

System.out.println("Size is "+hs.size());

System.out.println("Empty "+hs.isEmpty());

System.out.println(hs);

}

}

TreeSet : TreeSet internally implements SortedSet interfaces and SortedSet interface extends Set interface.

TreeSet display the elements in sorting (ascending order by default).

In TreeSet we have to store same types of values otherwise it will throw ClassCastException (type unchecked exception).

TreeSet provided some extra methods

Like subset, headset, tailset etc

**TreeSet**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

TreeSet hs = new TreeSet();

hs.add(3);

hs.add(5);

hs.add(2);

hs.add(1);

hs.add(7);

hs.add(4);

hs.add(5);

hs.add(null);

System.out.println(hs);

System.out.println(hs.tailSet(3));

}

}

**Queue : First In First Out**

PriorityQueue : **First In First Out depending upon the priority (lower priority).**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

PriorityQueue pq = new PriorityQueue();

pq.add(4);

pq.add(2);

pq.add(3);

pq.add(1);

pq.add(2);

pq.add(5);

System.out.println(pq);

System.out.println(pq.poll());

System.out.println(pq);

System.out.println(pq.poll());

System.out.println(pq);

}

}

**Map :** It is use to store key value pairs where key is unique and value may be duplicate.

HashMap

LinkedHashMap

TreeMap

Hashtable

**HashMap Example**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

HashMap hm = new HashMap();

hm.put(2,"Ramesh");

hm.put(1,"Rajesh");

hm.put(4,"Lokesh");

hm.put("abc","Ajay");

System.out.println(hm);

hm.put(1,"Balaji");

hm.put(5,"Ramesh");

System.out.println(hm);

if(!hm.containsKey(4)) {

hm.put(4,"Seeta");

System.out.println("Stored...");

}

System.out.println(hm);

System.out.println("name is "+hm.get(4));

hm.remove("abc");

System.out.println(hm);

}

}

LinkedHashMap : maintains the order.

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

LinkedHashMap hm = new LinkedHashMap();

hm.put(2,"Ramesh");

hm.put(1,"Rajesh");

hm.put(4,"Lokesh");

hm.put("abc","Ajay");

System.out.println(hm);

}

}

**TreeMap : sorted as a key. Because it internally implements sortedMap interface.**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

TreeMap hm = new TreeMap();

hm.put(2,"Ramesh");

hm.put(1,"Rajesh");

hm.put(4,"Lokesh");

hm.put(3,"Ajay");

System.out.println(hm);

}

}

**Hashtable** : By default all methods are synchronized.

Hashtable doesn’t allow key as null as well as value as null.

Where HashMap allow one key as null and value may be more than one null value.

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

HashMap hm = new HashMap();

//Hashtable hm = new Hashtable();

hm.put(2,"Ramesh");

hm.put(1,"Rajesh");

hm.put(4,"Lokesh");

hm.put(3,"Ajay");

hm.put(null,"Balaji");

hm.put(6,null);

System.out.println(hm);

}

}

Retrieving elements from collection of classes

1. For each loop : type of loop
2. Iterator
3. ListIterator
4. Enumeration : They are interfaces.

**Retrieve Records from Set API**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

Set ss = new HashSet();

ss.add(10); // auto-boxing

ss.add(20);

ss.add(30);

ss.add(40);

ss.add(50);

System.out.println(ss);

System.out.println("Using for each loop");

for(Object a:ss) {

System.out.println(a);

}

System.out.println("Using Iterator");

Iterator ii = ss.iterator();

while(ii.hasNext()) {

Object obj = ii.next();

System.out.println(obj);

}

}

}

Retrieve records from List

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

List ll = new ArrayList();

ll.add(10);

ll.add(20);

ll.add(30);

ll.add(40);

ll.add(50);

System.out.println(ll);

System.out.println("Using for each loop");

for(Object obj: ll) {

System.out.println(obj);

}

ListIterator li = ll.listIterator();

System.out.println("Forward direction");

while(li.hasNext()) {

System.out.println(li.next());

}

System.out.println("Forward direction");

while(li.hasPrevious()) {

System.out.println(li.previous());

}

}

}

**Enumeration Using Vector**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

Vector vv = new Vector();

vv.add(10);

vv.add(20);

vv.add(30);

vv.add(40);

Enumeration en = vv.elements();

while(en.hasMoreElements()) {

Object obj = en.nextElement();

System.out.println(obj);

}

}

}

**Map Example**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

Map mm = new HashMap();

mm.put(1,"RAvi");

mm.put(2,"Rajesh");

mm.put(3,"Lokesh");

/\*for(Objet obj :mm) {

System.out.println(mm);

}\*/

//Iterator ii = mm.iterator();

Set ss = mm.entrySet(); // converting map to set

Iterator ii = ss.iterator();

while(ii.hasNext()) {

//System.out.println(ii.next());

Map.Entry mp = (Map.Entry)ii.next(); //converting set to Map.Entry interface reference

System.out.println("Key is "+mp.getKey()+" , Values "+mp.getValue());

}

}

}

**Collection Framework with different type of values with Generics**

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

List ll = new ArrayList();

ll.add(10); // auto-boxing

ll.add(10.10);

ll.add("Ravi");

ll.add(20);

Object obj = ll.get(1);

if(obj instanceof Integer) {

Integer i = (Integer)obj;

int n = i.intValue();

System.out.println(n);

}

if(obj instanceof Double) {

Double i = (Double)obj;

double n = i.doubleValue();

System.out.println(n);

}

}

}

**Generics**

**Collection Framework with Generics**

CollectionClassName <Type>referenceName = new CollectionClass<Type>();

Set<Type> std = new HashSet<Type>();

Type may be all wrapper classes or user defined class object or string class.

Set<Integer> ss = new HashSet<Integer>();

import java.util.\*;

public class CollectionExpDemo {

public static void main(String args[]) {

List<Integer> ll = new ArrayList<Integer>();

ll.add(100); // auto-boxing : converting primitive to object

//ll.add("Ravi");

//ll.add(30.40);

ll.add(200);

ll.add(300);

int n = ll.get(0); // auto - unboxing : converting object to primitive

System.out.println(n);

}

}

**Collection Framework with Complex Object**

**Employee.java**

**package** bean;

**public** **class** Employee {

**private** **int** id;

**private** String name;

**private** **float** salary;

**public** **int** getId() {

**return** id;

}

**public** **void** setId(**int** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **float** getSalary() {

**return** salary;

}

**public** **void** setSalary(**float** salary) {

**this**.salary = salary;

}

**public** Employee(**int** id, String name, **float** salary) {

**super**();

**this**.id = id;

**this**.name = name;

**this**.salary = salary;

}

**public** Employee() {

**super**();

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", salary=" + salary + "]";

}

}

**DemoTest.java**

package main;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import bean.Employee;

public class DemoTest {

public static void main(String[] args) {

List<Employee> listOfEmp = new ArrayList<Employee>();

Employee emp1 = new Employee();

emp1.setId(100);

emp1.setName("Ravi");

emp1.setSalary(14000);

Employee emp2 = new Employee(101, "Ramesh", 16000);

System.out.println("Number of Records are "+listOfEmp.size());

listOfEmp.add(emp1);

listOfEmp.add(emp2);

listOfEmp.add(new Employee(102,"Rakesh",18000));

//System.out.println("Number of records are "+listOfEmp.size());

//listOfEmp.remove(0);

//System.out.println("Number of records are "+listOfEmp.size());

//listOfEmp.remove(emp2);

//System.out.println("Number of records are "+listOfEmp.size());

Iterator<Employee> ii = listOfEmp.iterator();

while(ii.hasNext()){

//Object obj = ii.next(); //without ii.next() method return type is Object class reference.

//Employee emp = (Employee)obj; // down level type casting.

//System.out.println("id is "+emp.getId()+" Name is "+emp.getName()+"Salary is "+emp.getSalary());

//System.out.println(emp);

Employee emp = ii.next();

System.out.println(emp);

}

}

}

**Arrays and Collections -🡪classes**

Both are static classes (all methods are static). Which provide set of methods which to do some operation of primitive array(Arrays) and List (Collections) classes respectively.

Soring for primitive values

**package** main;

**import** java.util.Arrays;

**public** **class** DemoTest1 {

**public** **static** **void** main(String[] args) {

**int** abc[]= {4,3,1,6,5,2,7,9,8};

System.***out***.println("Before sorting");

**for**(**int** n:abc) {

System.***out***.print(n+" ");

}

System.***out***.println();

System.***out***.println("After sorting");

Arrays.*sort*(abc);

**for**(**int** n:abc) {

System.***out***.print(n+" ");

}

}

}

**Collections.sort**

**package** main;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** DemoTest2 {

**public** **static** **void** main(String[] args) {

List<String> listOfStd = **new** ArrayList<>();

listOfStd.add("Vijay"); listOfStd.add("Mahesh"); listOfStd.add("Dinesh");

listOfStd.add("Vikash"); listOfStd.add("Ajay");

System.***out***.println("For each loop Before sort");

**for**(String name:listOfStd) {

System.***out***.print(name+"--");

}

Collections.*sort*(listOfStd);

System.***out***.println("\nAfter sorting Asc");

**for**(String name:listOfStd) {

System.***out***.print(name+"--");

}

Collections.*reverse*(listOfStd);

System.***out***.println("\nAfter sorting Desc");

**for**(String name:listOfStd) {

System.***out***.print(name+"--");

}

}

}

**Comparable and Comparator**

By default all wrapper + String class internally implements Comparable interface. That interface contains compareTo() method. This method help to do sorting.

**Day 9**

**21/01/2020**

**Assignment 7 : Comparator**

Employee -🡪 id,name,salary

Sorting by Id Asc, Desc

Sorting by Name Asc, Desc

Sorting by Salary Asc, Desc

Write Sample example to sort by Id, Name or Salary option takes through keyboards.

Sort by what

Id

Asc /Desc

Name

Asc /Desc

Salary

Asc/Desc

**2nd Mini Project**

**Sprint 1**

Product 🡪 bean package

pid, pname, price

ProductService -🡪 service package don’t use System.out.println🡪 in service

List/Set/Map of Product object.

Business methods

addProduct : object as a parameter (product) from main

product id must be unique

display result in main

updateProduct –object as parameter

update price using pid

deleteProduct 🡪 pid

delete using pid

displayAllProudct()

return all product

retrieveProductPrice()

using productId

main

Product Operation

Main Method

do {

1:Add

2:Update

3:Delete

4:DisplayAll

5: price using Id

switch(){

}

}while()

**IO Package**

Input Output Operation

**Stream :** Flow of data or it is abstraction between source and destination.

Source 🡪 keyboard, file, network, database etc

Destination 🡪 monitor (console), file, network ,database etc.

System.out.println(“”);

System.in

System is a pre-defined class part of lang package.

in, out,err are static property or reference.

in is a Reference of **InputStream**

out and err is a Reference of **PrintStream**

PrintStream ps = System.out;

ps.println(“Welcome to Java.”);

ps is a reference of Standard output device ie monitor.

InputStream is = System.in;

is is a reference of Standard input device ie keyboard.

IO

Stream

byte char

Input Output Input Output

Abstract classes

InputStream OutputStream Reader Writer

DataInputStream DataOutputStream InputStreamReader,OutputStreamWriter

FileInputStream FileOutputStream FileReader, FileWriter

BufferedInputStream BufferedOutputStream BufferedReader ,BufferedWriter

ObjectInputStream,OutputOutputStream

PrintStream PrintWriter

**Byte wise**

Source 🡪 Keyboard

Destination 🡪 Console

**package** com;

**import** java.io.DataInputStream;

**import** java.io.PrintStream;

**public** **class** ByteWiseInputAndOutputOperation {

**public** **static** **void** main(String[] args) **throws** Exception{

DataInputStream dis = **new** DataInputStream(System.***in***);

PrintStream ps = System.***out***;

ps.println("Enter the id ");

**int** id = Integer.*parseInt*(dis.~~readLine~~()); //convert string to int

ps.println("Enter the name");

String name = dis.~~readLine~~();

ps.println("Enter the salary");

**float** salary = Float.*parseFloat*(dis.~~readLine~~());

ps.println("Id is "+id);

ps.println("Name is "+name);

ps.println("Salary is "+salary);

}

}

**Byte wise**

Source 🡪 Keyboard

Destination 🡪 File

**package** com;

**import** java.io.DataInputStream;

**import** java.io.FileOutputStream;

**public** **class** ByteWiseFileOperation {

**public** **static** **void** main(String[] args) **throws** Exception{

DataInputStream dis = **new** DataInputStream(System.***in***);

FileOutputStream fos = **new** FileOutputStream("abc.txt");

System.***out***.println("Enter the data");

**int** ch;

**while**((ch=dis.read()) != '@') {

System.***out***.println(ch +"="+(**char**)ch);

fos.write(ch);

}

fos.close();

System.***out***.println("File created");

}

}

**Byte wise**

Source 🡪 File

Destination 🡪 File

**package** com;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** ByteWiseFileOperation1 {

**public** **static** **void** main(String[] args) **throws** Exception{

FileInputStream fis = **new** FileInputStream("C:\\Users\\91990\\Desktop\\Innoserv Online Training\\Java New Training Notes\\fullstacktrainingforzensar\\Java Programs\\ExpDemo.java");

FileOutputStream fos = **new** FileOutputStream("info.txt");

**int** ch;

**while**((ch=fis.read()) != -1){ //'a' or '@' EOF in java -1 is consider as EOF

fos.write(ch);

}

fis.close();

fos.close();

System.***out***.println("File copied...");

}

}

Buffer : It is temporary memory : It is use to improve the performance of file operation.

**Byte wise with Buffered Operation with file handling**

**package** com;

**import** java.io.BufferedInputStream;

**import** java.io.BufferedOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** ByteWiseFileOperation1 {

**public** **static** **void** main(String[] args) **throws** Exception{

FileInputStream fis = **new** FileInputStream("C:\\Users\\91990\\Desktop\\Innoserv Online Training\\Java New Training Notes\\fullstacktrainingforzensar\\Java Programs\\ExpDemo.java");

BufferedInputStream bis = **new** BufferedInputStream(fis); // connect to buffer

FileOutputStream fos = **new** FileOutputStream("info.doc");

BufferedOutputStream bos = **new** BufferedOutputStream(fos);

**int** ch;

**while**((ch=bis.read()) != -1){ //'a' or '@' EOF in java -1 is consider as EOF

bos.write(ch);

}

bos.flush();

fis.close();

fos.close();

System.***out***.println("File copied...");

}

}

**Char wise classes**

Source 🡪Keyboard

Destination 🡪 console

**package** com;

**import** java.io.BufferedReader;

**import** java.io.InputStreamReader;

**public** **class** CharcaterWiseInputOperation {

**public** **static** **void** main(String[] args) **throws** Exception{

//InputStreamReader isr = new InputStreamReader(System.in);

//BufferedReader br = new BufferedReader(isr);

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

System.***out***.println("Enter the id");

**int** id = Integer.*parseInt*(br.readLine());

System.***out***.println("Enter the name");

String name = br.readLine();

System.***out***.println("Enter the salary");

**float** salary = Float.*parseFloat*(br.readLine());

System.***out***.println("id is "+id);

System.***out***.println("name is "+name);

System.***out***.println("salary s "+salary);

}

}

**Character wise file operation**

**package** com;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**public** **class** CharacterwiseFileOperation {

**public** **static** **void** main(String[] args) **throws** Exception{

FileReader fr = **new** FileReader("info.txt");

FileWriter fw = **new** FileWriter("D:\\infodetails.txt");

**int** ch;

**while**((ch=fr.read()) != -1) { // read data till end -1 EOF

fw.write(ch);

}

fr.close();

fw.close();

System.***out***.println("File copied...");

}

}

**Assignment 9:**

**File Assignment**

1. Convert all file information in upper case in target file.
2. Convert all file information in lower case in target file
3. Convert all sentence first letter in upper case.
4. Display number of character present in file.
5. Display number of words present in file.

Store and Retrieve primitive data types like id,name,salary

DataInputStream with FileInputStream :

DataOutputStrream with FileOutputStream :

**Object Serialization :** Storing the object itself or object property or converting object into stream or serializable format is known as Object Serialization.

Object -🡪property, behaviour and identity

Id,name,salary 🡪 property

toString() or user defined -🡪 behaviour

identity 🡪 emp

**Object De-Serialization** : converting stream object back to object format is known as Object De-Serialization.

Which class object do you want to do serialization that class must be implements Serializable interface.

Serializable interface is a type of maker interface.

It support to do Serialization.

**2nd Mini Project**

**Sprint 2**

Product Object Stored permanently in external file.

**Hint : (Collection Framework to store multiple objects)**

**JDBC**

**Java Database Connectivity :** JDBC is a API(Application Programming interface) which provide set of classes and interfaces which help to connect database (oracle, mysql, db2 or sql server) through Java Technologies.

Steps to connect MySQL database through Java Technologies.

1. import java packages

java.sql.\*;

javax.sql.\*;

1. Write user-defined method or main method with exception handling concept ie try-catch or throws because JDBC through SQLException checked exception.
2. Load the Driver

Driver : piece of software.

Driver : It is pre-defined API provided by vendor whose database going to connect.

**4 types**

1. Type 1 : JDBC ODBC Bridge Driver : removed from Java8 onward. OS Dependent. Performance wise slow. ODBC code written using C.
2. Type 2 : JDBC Native API Driver : Database dependent. Performance very fast.
3. Type 3 : JDBC Net protocol Driver : Server(Web Logic, JBoss, GlashFish) where we have configure Database Source Features.
4. Type 4 : JDBC Pure or thin Driver . Download jar file respective that database. Jar file is database dependent. JDBC type 4 code written using Java.

Class.forName(“driverName”)

Class is a pre-defined class part of lang package. Which contains forName() static method which help to load the driver(API).

1. Establish the connection :

DriverManager is a pre-defined class provided getConnection static method. which takes three parameter url, username,password etc.

DriverManager.getConnection(url,username,password);

**getConnection()** method return type is Connection interface reference.

1. Create Statement :

Types of Statement

1. Statement
2. PreparedStatement
3. CallableStatement

Statement, PreparedStatement, CallableStatement all are interfaces which provide set of methods which help to do some operation on table or procedure or functions.

Syntax to create the Statement

**Statement stmt = con.createStatement();**

* 1. Retrieve :

ResultSet rs = stmt.executeQuery(“select clause”);

executeQuery method return type is ResultSet interface

reference.

* 1. DML Operation

int res = stmt.executeUpdate(“DML Operation”);

Insert may be exception

Or

res 1

if(res>0) {

System.out.println(res);

}

Delete and update count depending query get effected.

No record effected count is 0 else number records effect hold by count variable.

Statement : Statement interface doesn’t support parameterized query concept.

When we execute any query using Statement, query get compile on java side, send to database, then execute in database and get the acknowledgement may be success or failure.

PreparedStatement : it support parameterized query concept.

When we execute any query using PreparedStatement query compile once and execute n number of times in database.

Using PreparedStatement we can improve the performance.

PreparedStatement reference is use for only one query at time.

Callable Statement is use to execute Stored Procedure or Function from database.

Stored Procedure or Function : Executing more than one query with terms and conditions and stored with named block.

**Sprint 3 Release**

**Day 10**

**22/01/2020**

**Java 7 Features**

In switch we can use variable type is String reference.

switch(variableName) {

case 1 or ‘a’ or “abc”:

}

Try with more than one exception in single catch block syntax

**package** com;

**public** **class** TryWithMutlipleException {

**public** **static** **void** main(String[] args) {

**try** {

} **catch** (ArithmeticException | ArrayIndexOutOfBoundsException e) {

}**catch** (NumberFormatException e) {

}

}

}

Try with resources

**Try without Resource with finally**

package com;

import java.io.FileInputStream;

import java.io.FileOutputStream;

public class TryWithOutResource {

public static void main(String[] args) {

FileInputStream fis = null;

FileOutputStream fos = null;

try {

fis = new FileInputStream("C:\\Users\\91990\\Desktop\\Innoserv Online Training\\Java New Training Notes\\fullstacktrainingforzensar\\Java Programs\\Demo.java");

fos = new FileOutputStream("xyz.txt");

int ch;

while((ch=fis.read()) != -1) {

fos.write(ch);

}

} catch (Exception e) {

System.err.println(e);

}finally {

try {

fis.close();

fos.close();

}catch (Exception e) {

// TODO: handle exception

}

}

}

}

**Try with Resources**

In Sprint 2 we have to use exception handling try with resources

**try(**

**Create the object of those classes which class implements AutoCloseable.**

**) {**

**}catch(Exception e) {**

**}**

**Try with resources**

**package** com;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**public** **class** TryWithResource {

**public** **static** **void** main(String[] args) {

**try**(

FileInputStream fis = **new** FileInputStream("C:\\Users\\91990\\Desktop\\Innoserv Online Training\\Java New Training Notes\\fullstacktrainingforzensar\\Java Programs\\Demo.java");

FileOutputStream fos = **new** FileOutputStream("xyz.txt");

)

{

**int** ch;

**while**((ch=fis.read()) != -1) {

fos.write(ch);

}

}**catch** (Exception e) {

// **TODO**: handle exception

System.***err***.println(e.toString());

}

}

}

**Try with Resources**

**package** com;

**class** Employee **implements** AutoCloseable{

**void** display() {

System.***out***.println("business method");

}

@Override

**public** **void** close() **throws** Exception {

// **TODO** Auto-generated method stub

System.***out***.println("close method called..");

}

}

**public** **class** TryWithUserdefinedResource {

**public** **static** **void** main(String[] args) {

**try**(Employee emp = **new** Employee()) {

emp.display();

}**catch**(Exception e) {}

}

}

**Java 8 Features**

**Interface :**

Interface can contains method with body from Java8.

But method must be default or static.

interface A {

void dis1();

default void dis2() {

System.out.println("Default dis2() method implementation");

}

default void dis3() {

dis2();

System.out.println("Default dis2() method implementation");

}

static void dis4() {

System.out.println("Default static implementation");

}

}

class B implements A{

public void dis1() {

System.out.println("dis1() method override by B class");

}

public void dis3() {

System.out.println("dis3() default implementation override by B class");

}

}

class Java8Features {

public static void main(String args[]) {

B obj1 = new B();

obj1.dis1();

obj1.dis2();

obj1.dis3();

A.dis4();

}

}

**Functional interfaces :**

The interface which contains only method is known as functional interface. This interface can contains more than one default as well as static but only one abstract methods.

**Functional Interface**

@FunctionalInterface

interface A {

void dis1();

//void dis2();

default void dis3() {}

static void dis4() {}

default void dis5() {}

}

class Java8Features {

public static void main(String args[]) {

}

}

**Inner classes (Java5 onwards).**

Class within another class is known as inner class.

1. Non static inner class
2. Static inner class
3. Anonymous inner class
4. Local inner class

**Non Static Inner class**

class Outer {

int a;

void dis1(){

System.out.println("Outer class dis() method");

Inner in = new Inner();

//in.dis2();

}

class Inner {

int b;

void dis2() {

System.out.println("Inner class dis2() method "+a);

//dis1();

}

}

}

class InnerClassDemo {

public static void main(String args[]) {

Outer out = new Outer();

out.dis1();

//Inner in = new Inner();

Outer.Inner in = out.new Inner();

in.dis2();

Outer.Inner in1 = new Outer().new Inner();

in1.dis2();

Outer.Inner in2 = new Outer().new Inner();

in2.dis2();

}

}

**Static Inner class**

class Outer {

int a;

void dis1(){

System.out.println("Outer class dis() method");

}

static class Inner {

int b;

void dis2() {

System.out.println("Inner class dis2() method ");

}

}

}

class InnerClassDemo {

public static void main(String args[]) {

//Outer.Inner in1=new Outer().new Inner(); //Non static Inner class

Outer.Inner in2 = new Outer.Inner(); //Static Inner class

in2.dis2();

}

}

**Anonymous Inner Class**

interface A {

void dis1();

}

class B implements A {

public void dis1() {

System.out.println("B class implements A interface method");

}

}

class InnerClassDemo {

public static void main(String args[]) {

A obj1 = new B(); //Runtime polymorphism

obj1.dis1();

A obj2 = new A(){

public void dis1() {

System.out.println("A interface implementation provided by anonymous inner class first logic");

}

};

obj2.dis1();

A obj3 = new A(){

public void dis1() {

System.out.println("A interface implementation provided by anonymous inner class second logic");

}

};

obj3.dis1();

}

}

**Local Inner class**

class InnerClassDemo {

public static void main(String args[]) {

class Inner {

void dis1() {

System.out.println("Local Inner class");

}

}

Inner in = new Inner();

in.dis1();

}

}

**Lambda Expression**

Lambda Expression is a type of anonymous Inner function or methods.

From Java8 onwards we can say Java is procedure, object oriented and functional programming language like JavaScript.

Inner or nested functions.

Lambda expression we have use only those interface which is known as Functional interfaces.

Simple Lambda Expression Example

@FunctionalInterface

interface A {

void dis1();

}

class B implements A {

public void dis1() {

System.out.println("B class implements A interface method");

}

}

class Java8Features{

public static void main(String args[]) {

//1st way

A obj1 = new B();

obj1.dis1();

//2nd way

A obj2 = new A(){

public void dis1() {

System.out.println("A interface implementation provided by anonymous inner class first logic");

}

};

obj2.dis1();

//3rd way using lambda expression

A obj3 = ()->System.out.println("dis1() method implementation provided by lambda expression");

obj3.dis1();

A obj4 = () -> {

System.out.println("1st statement");

System.out.println("2nd statement");

};

obj4.dis1();

}

}

**Another Example for lambda**

@FunctionalInterface

interface Calculation {

public int add(int x, int y);

}

class Java8Features{

public static void main(String args[]) {

Calculation c1 = (a,b)->a+b;

System.out.println(c1.add(1,2));

Calculation c2 = (int a,int b)->a+b;

System.out.println(c2.add(3,4));

Calculation c3 = (int x,int y)->x+y;

System.out.println(c3.add(5,6));

Calculation c4 = (int x, int y)->{

int sum = x+y;

return sum;

};

System.out.println(c4.add(7,8));

}

}

**Lambda Expression with passing object and return objects**

class Employee {

String name;

float salary;

}

@FunctionalInterface

interface EmployeeInterface {

public Employee passEmployee(Employee emp);

}

class Java8Features{

public static void main(String args[]) {

Employee emp1 = new Employee();

emp1.name="Ravi Kumar";

emp1.salary=24000;

EmployeeInterface empInf = (emp)->{

emp.salary = emp.salary+2000;

return emp;

};

Employee emp2 = empInf.passEmployee(emp1);

System.out.println("Name is "+emp2.name+" Salary is "+emp2.salary);

}

}

**Predefined functional interfaces**

**function** package is a package. It is sub package of util packages.

**Top level functional interfaces**

javap java.util.function.Function

javap java.util.function.Predicate

javap java.util.function.Consumer

javap java.util.function.Supplier

1. **Function :**  apply() function. Takes T parameter and return R value.
2. **Predicate :** test() : which takes T as parameter and return boolean value.
3. **Consumer :** accept() takes T as parameter and no return type.
4. **Supplier :** get() no parameter and return T value.

**Function code**

**package** com;

**import** java.util.function.Function;

**class** MyFunction **implements** Function<Integer, String>{

@Override

**public** String apply(Integer t) {

**return** "You pass value as "+t;

}

}

**public** **class** FunctionDemoTest {

**public** **static** **void** main(String[] args) {

Function<Integer, String> r = **new** MyFunction();

System.***out***.println(r.apply(100));

Function<Integer,Integer> r1 = (val)->val+100;

System.***out***.println(r1.apply(200));

}

}

**Consumer Code**

**package** com;

**import** java.util.function.Consumer;

**class** MyConsumer **implements** Consumer<Float>{

@Override

**public** **void** accept(Float t) {

System.***out***.println(t);

}

}

**public** **class** MyConsumerDemo {

**public** **static** **void** main(String[] args) {

Consumer<Float> c1 = **new** MyConsumer();

c1.accept(10.10f);

c1.accept(20.20f);

Consumer<Integer> c2 = (v)->System.***out***.println(v);

c2.accept(100);

c2.accept(200);

}

}

**Supplier Code**

**package** com;

**import** java.util.function.Supplier;

**class** DemoSupplier **implements** Supplier<String>{

@Override

**public** String get() {

// **TODO** Auto-generated method stub

**return** "Welcome to Java8 Features";

}

}

**public** **class** MySupplier {

**public** **static** **void** main(String[] args) {

Supplier<String> s = **new** DemoSupplier();

System.***out***.println(s.get());

Supplier<String> s1 = ()->"Welcome Java8 Features using Lambda Expression";

System.***out***.println(s1.get());

}

}

**Predicate Code**

**package** com;

**import** java.util.function.Predicate;

**class** MyPredicate **implements** Predicate<Integer>{

@Override

**public** **boolean** test(Integer t) {

// **TODO** Auto-generated method stub

**return** t>100;

}

}

**public** **class** PredicateDemoTest {

**public** **static** **void** main(String[] args) {

Predicate<Integer> p1 = **new** MyPredicate();

System.***out***.println(p1.test(50));

System.***out***.println(p1.test(150));

Predicate<String> p2 = (name)->name.length()>10;

System.***out***.println(p2.test("Ravi Kumar"));

System.***out***.println(p2.test("Ajay Kumar Patil"));

}

}

**ArrayList using forEach method**

**package** com;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**import** java.util.function.Consumer;

**public** **class** ForEachMethodDemo {

**public** **static** **void** main(String[] args) {

List<String> stdNames = **new** ArrayList<String>();

stdNames.add("Ravi"); stdNames.add("Seeta");stdNames.add("Reeta");stdNames.add("Ajay");

stdNames.add("Ramu"); stdNames.add("Teeta");stdNames.add("Lokesh");stdNames.add("Vijay");

System.***out***.println("Using for each loop");

**for**(String name:stdNames) {

System.***out***.print(name+" ");

}

System.***out***.println();

System.***out***.println("Using Iterator");

Iterator<String> li = stdNames.iterator();

**while**(li.hasNext()) {

String name = li.next();

System.***out***.print(name+" ");

}

System.***out***.println();

System.***out***.println("Using Constumer with Lambda Expression");

stdNames.forEach(val->System.***out***.print(val+" "));

}

}

**Stream API:**

**Stream API** is use to process collection of data from collection framework or primitive array. Stream API provide lot method, those methods we can use as a pipe line to load the data on demand. Stream hold the data form temporary purpose.

**Intermediate operator as well as terminal operator**

**Intermediate operator return stream itself.**

**Where terminal operator return non stream means void, int, float, Boolean etc.**

**Intermediate operator is use to solve the business requirement.**

**Collection**

**--🡪 Stream -🡪 in1-🡪in2-🡪in3-🡪in4-🡪Tr.**

**Array of primitive**

**Stream get destroy.**

**Stream doesn’t change your origin data.**

**Day 11**

**23/01/2020**

**Stream API**

Collection framework or Data Structure is an in-memory data structure to hold value and before we start using in Collection. First we have to load the all values or populated all values using loop or iterator or Enumeration etc. If we do any changes in data original data get effected.

Using Stream API we can load the data on-demand. Stream doesn’t effect or update origin data.

Using Stream we can apply one or more operation on stream data.

Stream operator or methods divided into two types

1. Intermediate operator or methods. This method return type is Stream reference or object.
2. Terminator operator or methods. This method return type is non Stream means primitive type or void type.

Streaming You tube video

Source Data

Collection Framework

With primitive or

Complex object -🡪 Stream -🡪 I1🡪I2-🡪In--🡪T: primitive type or

void or Collection or primitive array.

Primitive array --🡪 Stream

Stream API’s Intermediate operator or terminal operator take the help of lambda expression with functional interface or Method reference to achieve the task on demand.

**Stream Example**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.IntStream;

**import** java.util.stream.Stream;

**public** **class** StreamCreationTest {

**public** **static** **void** main(String[] args) {

//1st way Primitive types

//Stream<Integer> s1 = Stream.of(10,20,30,40,50,60);

//s1.forEach(v->System.out.print(v+" "));

//2nd Way Integer object

//Integer abc[]= {10,20,30,40,50,60};

//Stream<Integer> s1 = Stream.of(abc);

//s1.forEach(v->System.out.print(v+" "));

//3rd way primitive array

//int abc[]= {10,20,30,40,50,60};

//IntStream s1 = IntStream.of(abc);

//s1.forEach(v->System.out.print(v));

//4th way Collection

//List<String> listOfStd = new ArrayList<>();

//listOfStd.add("Ravi"); listOfStd.add("Ramesh"); listOfStd.add("Ajay");

//listOfStd.add("Vikash"); listOfStd.add("Akash");

//Stream<String> s1 = listOfStd.stream();

//s1.forEach(v1->System.out.print(v1+" "));

//5th Way Collection with complex object

List<Employee> listOfEmp = **new** ArrayList<>();

listOfEmp.add(**new** Employee(100, "Ravi", 16000));

listOfEmp.add(**new** Employee(101, "Ramesh", 19000));

listOfEmp.add(**new** Employee(102, "Raju", 12000));

listOfEmp.add(**new** Employee(103, "Rakesh", 15000));

Stream<Employee> s1 = listOfEmp.stream();

//s1.forEach(e->System.out.print(e.getId()+" "));

//s1.forEach(e->System.out.print(e.getName()+" "));

//s1.forEach(e->System.out.print(e.getSalary()+" "));

//s1.forEach(e->System.out.println(e));

//listOfEmp.stream().forEach(emp->System.out.println(emp));

}

}

**Map** map is a one type of intermediate operator. Map() produce a new stream applying for each value or data or the original stream.

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Stream;

**public** **class** StreamWithMap {

**public** **static** **void** main(String[] args) {

List<Employee> listOfEmp = **new** ArrayList<>();

listOfEmp.add(**new** Employee(100, "Vikash", 16000));

listOfEmp.add(**new** Employee(101, "Ajay", 19000));

listOfEmp.add(**new** Employee(102, "Mahesh", 12000));

listOfEmp.add(**new** Employee(103, "Rakesh", 15000));

//1st way

//Stream<Employee> s1 = listOfEmp.stream();

//Stream<Employee> s2 = s1.map(emp->emp); // generic map

//s2.forEach(emp->System.out.println(emp));

//2nd way

//listOfEmp.stream().map(emp->emp).forEach(emp->System.out.println(emp));

//3rd : Using map passing emp object and return only name

//listOfEmp.stream().map(emp->emp.getName()).forEach(emp->System.out.print(emp+" "));

//System.out.println();

//listOfEmp.stream().forEach(emp->System.out.print(emp.getName()+" "));

//4th : using map passing emp object and return only name in upper case

//listOfEmp.stream().map(emp->emp.getName().toUpperCase()).forEach(emp->System.out.print(emp+" "));

//5th : using map passing emp object and return only name in upper case with sorted..

// listOfEmp.stream().map(emp->emp.getName().toUpperCase()).sorted().forEach(name->System.out.println(name));

//6th way Dispaly by Id and salary

//listOfEmp.stream().map(emp->emp).forEach(emp->System.out.println("id is "+emp.getId()+" Salary "+emp.getSalary()));

}

}

Filter() : Filter accept a predicate to filter all elements of the stream. This operator is intermediate operator. To get the data from filter() we have to connect terminal operator.

**Filter example**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StreamWithFilter {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

List<Employee> listOfEmp = **new** ArrayList<>();

listOfEmp.add(**new** Employee(100, "Vikash", 16000));

listOfEmp.add(**new** Employee(101, "Ajay", 19000));

listOfEmp.add(**new** Employee(102, "Mahesh", 12000));

listOfEmp.add(**new** Employee(103, "Rakesh", 15000));

//1st filter by id

//listOfEmp.stream().filter(emp->emp.getId()==100).forEach(emp->System.out.println(emp));

//2nd filter by Name

//listOfEmp.stream().filter(emp->emp.getName().contains("e")).forEach(emp->System.out.println(emp));

//3rd Filter by salary

//listOfEmp.stream().filter(emp->emp.getSalary()>15000).forEach(emp->System.out.println(emp));

//4th Filter with Map to display Name with salary condition

//float salary = 16000;

//listOfEmp.stream().filter(emp->emp.getSalary()>salary ).map(emp->emp.getName()).forEach(name->System.out.println(name));

//listOfEmp.stream().filter(emp->emp.getSalary()>15000).forEach(name->System.out.println(name));

//5th filter with two conditions

listOfEmp.stream().filter(emp->emp.getSalary()>15000 && emp.getId()==101).map(emp->emp.getName()).forEach(name->System.***out***.println(name));

}

}

**collect:** This is know as Terminal operator which help to repacking the elements or data into collections.

DAO Layer

**Collect example**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.stream.Collectors;

**public** **class** StreamWithCollect {

**public** **static** **void** main(String[] args) {

List<Employee> listOfEmp = **new** ArrayList<>();

listOfEmp.add(**new** Employee(100, "Vikash", 16000));

listOfEmp.add(**new** Employee(101, "Ajay", 19000));

listOfEmp.add(**new** Employee(102, "Mahesh", 12000));

listOfEmp.add(**new** Employee(103, "Rakesh", 15000));

System.***out***.println("Before Stream Number of records are "+listOfEmp.size());

//1st Way Collect all object with filter the data.

//List<Employee> listOfEmp1 = listOfEmp.stream().filter(emp->emp.getSalary()>=16000).collect(Collectors.toList());

//System.out.println("Number of Records are "+listOfEmp1.size());

//2nd Way Collect only name with filter data

//List<String> names = listOfEmp.stream().filter(emp->emp.getSalary()>=16000).map(emp->emp.getName()).collect(Collectors.toList());

//System.out.println("Names are "+names.size());

//System.out.println(names);

//3rd Way Collect all object with filter the data and store in Set

//Set<Employee> listOfEmp1 = listOfEmp.stream().filter(emp->emp.getSalary()>=16000).collect(Collectors.toSet());

//System.out.println("Number of Records are "+listOfEmp1.size());

}

}

More intermediate operator

**skip(), limit(),distinct() etc**

**Terminal Operator**

count()

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StreamWithIntermediateOperator {

**public** **static** **void** main(String[] args) {

List<String> listOfStd = **new** ArrayList<>();

listOfStd.add("Ramesh"); listOfStd.add("Ajay");

listOfStd.add("Ajay");listOfStd.add("Vikash");

listOfStd.add("Mahesh"); listOfStd.add("Dinesh");

listOfStd.add("Seeta");listOfStd.add("Meeta");

//1st all names

listOfStd.stream().forEach(name->System.***out***.print(name+" "));

//2st all names with skip few names

System.***out***.println();

listOfStd.stream().skip(3).forEach(name->System.***out***.print(name+" "));

//3rd limit the names

System.***out***.println();

listOfStd.stream().limit(4).forEach(name->System.***out***.print(name+" "));

//4th distinct

System.***out***.println();

listOfStd.stream().distinct().forEach(name->System.***out***.print(name+" "));

//5th count

**long** numberOfRec = listOfStd.stream().count();

System.***out***.println();

System.***out***.println("Number of records "+numberOfRec);

//6th count with remove duplicate

**long** uniqueRec = listOfStd.stream().distinct().count();

System.***out***.println();

System.***out***.println("Number of records "+uniqueRec);

}

}

reduce() : It is a type of terminal operator which help to generate result base upon the conditions.

**max(),**

**min()**

**avg()**

**reduce example**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Optional;

**public** **class** StreamWithReduce {

**public** **static** **void** main(String[] args) {

List<String> listOfStd = **new** ArrayList<>();

listOfStd.add("Ramesh"); listOfStd.add("Ajay");

listOfStd.add("Ajay");listOfStd.add("Vikash");

listOfStd.add("Mahesh"); listOfStd.add("Dinesh");

listOfStd.add("Seeta");listOfStd.add("Meeta");

Optional<String> str = listOfStd.stream().filter(s->s.endsWith("b")).reduce((s1,s2)->s1+""+s2);

**if**(str.isPresent()) {

System.***out***.println("yes");

}**else** {

System.***out***.println("No");

}

}

}

**match()** : match operator are type of terminal operator which return type is Boolean base upon the intermediate filter() operator.

**Match operator**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** StreamWithMatchOperation {

**public** **static** **void** main(String[] args) {

List<String> listOfStd = **new** ArrayList<>();

listOfStd.add("Ramesh"); listOfStd.add("Ajay");

listOfStd.add("Ajay");listOfStd.add("Vikash");

listOfStd.add("Mahesh"); listOfStd.add("Dinesh");

listOfStd.add("Seeta");listOfStd.add("Meeta");

//1st match operation

**boolean** res1 = listOfStd.stream().anyMatch(name->name.startsWith("A"));

//2nd match operation

**boolean** res2 = listOfStd.stream().allMatch(name->name.startsWith("A"));

//3rd operation

**boolean** res3 = listOfStd.stream().noneMatch(name->name.startsWith("B"));

System.***out***.println(res1);

System.***out***.println(res2);

System.***out***.println(res3);

}

}

**Method Reference :**

Java 8 Provide method reference concept. Method reference is use to refer method of functional interface. It is compact and easy form of lambda expression. Method is reference is a replacement of lambda expression.

To achieve the method reference we have to use

**::** Opterator

Static method reference

Non static method reference

Constructor method reference

**package** com;

**interface** A {

**void** dis1();

}

**class** B {

**public** **static** **void** dis1() {

System.***out***.println("This is static method reference");

}

}

**class** C {

**public** **void** dis1() {

System.***out***.println("This non static method reference");

}

}

**public** **class** MethodReferenceDemo {

**public** **static** **void** main(String[] args) {

A obj1 = ()->System.***out***.println("Provided by using Lambda Expression 1 time");

obj1.dis1();

A obj2 = ()->System.***out***.println("Provided by using Lambda Expression 1 time");

obj2.dis1();

// Non static method reference

A obj3 = B::*dis1*; // Link B class with static Method reference concept.

obj3.dis1();

A obj4 = B::*dis1*;

obj4.dis1();

C ref = **new** C();

A obj5 = ref::dis1; // Link C class with Non static method reference concept

obj5.dis1();

}

}

**Stream API with method Reference**

**package** com;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.function.Function;

**public** **class** SteamWithMethodRefernce {

**public** **static** **void** main(String[] args) {

Function<String, Integer> str = (op)->Integer.*parseInt*( op);

System.***out***.println(str.apply("100"));

Function<String, Integer> str1 = Integer :: *parseInt*;

System.***out***.println(str1.apply("100"));

List<String> listOfStd = **new** ArrayList<>();

listOfStd.add("Ramesh"); listOfStd.add("Ajay");

listOfStd.add("Ajay");listOfStd.add("Vikash");

listOfStd.add("Mahesh"); listOfStd.add("Dinesh");

listOfStd.add("Seeta");listOfStd.add("Meeta");

//Retrieve all records using lambda expression

System.***out***.println("Using Lambda Express");

listOfStd.stream().forEach(v->System.***out***.print(v+" "));

System.***out***.println();

System.***out***.println("Using Method reference");

//Retrieve all records using Method reference.

listOfStd.stream().forEach(System.***out***::print);

}

}

**Spring2 Limitation : Storing the data in file base system**

**Sprint 3 : Storage file system replace by database system.**

**Database as well as Java8 Stream API**

**2nd Mini Project**

**Sprint 2**

Product 🡪 bean package

pid, pname, price

Pure Business Logic

ProductService -🡪 service package don’t use System.out.println🡪 in service

List/Set/Map of Product object.

addProduct() : product object as parameter

price > 500 Before calling database

Dao method call to store the records

deleteProduct() delete using Pid no business logic

updateProduct() no business logic

pid, price

displayProduct() after retrieve business logic

20% + actual price using Stream API

dispalyAllProductLowPrice() Asc Price

displayAllProductHighPrice() Des Price

displayAllProductName() Asc

displayAllProductName() Desc Using Stream API

ProductDao : dao package

DAO Layer (Pure JDBC Logic)

addProduct(Product pp) {

JDBC code to store the product

Primary key

}

deleteProduct(pid ) {

}

updateProduct(Product pp) {

}

List<Product> retrieveAllProduct() {

Using ResultSet retrieve all records and convert to object and store in List/Set/Map

}

Product table 🡪 Product class

PID pid

PName pname

PRICE price

main

Product Operation

Main Method

do {

1:Add

2:Update

3:Delete

4:DisplayAll

5: ProductNameAsc

6:ProductNameDesc

7:PriceLow

8:PriceHigh

switch(){

}

}while()

**Day 12**

**27/01/2020**