Day1

Collection f/w:

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

normal collection ()

Map(key-value)

Iterator

Queue

Generics

Map

List: when duplicate elements are allowed, and we want to store/get the

elements based on index.

Set: when we don't want duplicate elements.

HashSet

LinkedHashSet ------> equals() and hashCode() ---- Object class

TreeSet---> Comparable or Comparator

Collections

Arrays

Iterator(I) and ListIterator(I):

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

--Iterator interface belongs to java.util package.

--using this Iterator we can iterate each elements from any collection one by one

in a standard manner.

--Iterator is also known as universal cursor.becoz it is applicable with any

type of collection classes.

example:

Demo.java:

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

List<String> list= Arrays.asList("a","b","c","d","e");

Iterator<String> itr= list.iterator();

while(itr.hasNext()) {

String s=itr.next();

System.out.println(s);

}

}

}

--using Iterator, while iterating elements from any collection, we can

remove the elements also.

Demo.java:

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

ListIterator(I):

========

--it is the child inerface of the Iterator interface.

--it is the most powerfull cursor in collection f/w with only one

limitation, with ListIterator we can work with only List category.

we can can work with Set, Queue.

--with this LI we can navigate both side (forward and backward)

--it is a bi-diectional cursor.

public class Demo {

public static void main(String[] args) {

List<String> list= Arrays.asList("a","b","c","d","e");

ListIterator<String> ltr= list.listIterator();

while(ltr.hasNext()) {

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

}

System.out.println(list);

}

}

Queue(I):

=========

--if we want to represent a group of object prior to processing (before

processing we want to arrange the object) then we should use the Queue.

ex:

sending mail or SMS to multiple people.

--this Queue concept introduced in java 1.5 v.

--Q follows FIFO order but based on our requirement we can implement our own order also by using

PriorityQueue class.

PriorityQueue(c):

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

Queue<Integer> q = new PriorityQueue<>();

--here null insertion is not possible even a single null also.

--inside the PQ if we want any elements then that element should be Comparable, (or we can use Comparator) otherwise we get ClassCastException.

Note: inside the Q the elements will not be as sorted as inside the TreeSet, but by using its peek() and poll() method, remove() method it will work according to the Comparable or Comparator(Sorting order).

example:

Demo.java:

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

package com.masai;

import java.util.PriorityQueue;

import java.util.Queue;

public class Demo {

public static void main(String[] args) {

Queue<Integer> q = new PriorityQueue<>();

q.add(10);

q.add(20);

q.add(30);

q.add(12);

q.add(22);

q.add(35);

System.out.println(q);

Integer i1;

while((i1 = q.poll()) != null) {

System.out.println(i1);

}

System.out.println(q);

}

}

example 2:

package com.masai;

import java.util.PriorityQueue;

import java.util.Queue;

public class Demo {

public static void main(String[] args) {

Queue<Student> q = new PriorityQueue<>(new StudentMarksComp());

q.offer(new Student(10, "N1", 500));

q.add(new Student(20, "N2", 400));

q.add(new Student(30, "N3", 440));

q.add(new Student(40, "N4", 450));

q.add(new Student(50, "N5", 380));

q.add(new Student(60, "N6", 550));

System.out.println(q);

Student s;

while((s = q.poll()) != null) {

System.out.println(s);

}

}

}

example 3: Queue using LinkedList class: FIFO order

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

package com.masai;

import java.util.Iterator;

import java.util.LinkedList;

import java.util.PriorityQueue;

import java.util.Queue;

public class Demo {

public static void main(String[] args) {

Queue<Student> q = new LinkedList<>();

q.offer(new Student(10, "N1", 500));

q.add(new Student(20, "N2", 400));

q.add(new Student(30, "N3", 440));

q.add(new Student(40, "N4", 450));

q.add(new Student(50, "N5", 380));

q.add(new Student(60, "N6", 550));

System.out.println(q);

// for(Student s:q) {

//

// System.out.println(s);

// }

//

Iterator<Student> itr= q.iterator();

while(itr.hasNext()) {

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

}

}

}

Generics:

=======

--this concept is also introduced in java 1.5v.

--the main objective of Generics is to achieve typesafty.

--It makes the code stable by detecting the bugs at compile time.

--Generics concept mostly used with collection f/w related classes, but we can

apply the generics concept without collection f/w also.

Note: Generics concept is not applicable with primitive data types, it supports

only the obejcts.

--we can create our own generic classes or generic methods. and that class or

method can work with any kind of data.

adv of generics:

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

1.type checking at compile time.

2.get rid of downcasting problem.

--with this our code will become bug(most of them detected at compile time) free and more stable.

example :

MyGen.java:

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

package com.masai;

public class MyGen<T> {

T data;

public MyGen(T data) {

this.data=data;

}

public T getData() {

return data;

}

}

Demo.java:

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

package com.masai;

public class Demo {

public static void main(String[] args) {

MyGen<Integer> m1= new MyGen<>(10);

System.out.println(m1.getData());

MyGen<String> m2=new MyGen<>("Hello");

System.out.println(m2.getData());

MyGen<Student> m3= new MyGen<>(new Student(10, "Ram", 500));

System.out.println(m3.getData());

}

}

constraints in generics:

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

--In generic classes, we can apply the contraints or bound the type parameter for a

perticular range by using "extends" keyword.

ex1:

class Demo<T>{

}

--it is unbound type, without any constraint, we can pass any type of parameter

Demo<Integer> d1= new Demo<>();

Demo<String> d1= new Demo<>();

Demo<Student> d1= new Demo<>();

ex2:

class Demo <T extends X>{ // either X or any child class of X

}

--here if X is a class type then we can pass either X types elements or sub class elements of X type as parameter type.

ex3:

class Demo<T extends Number>{ // here Number class belongs to java.lang package

}

Number class will act as a parent class, for all the wrapper classes

which represent the numeric data types.

Demo<Number> d1= new Demo<>(); // valid

Demo<Integer> d1= new Demo<>(); // valid

Demo<Double> d1= new Demo<>(); // valid

Demo<String> d1= new Demo<>(); // invalid

--if X is an interfacem then we are able to pass either X type elements or X implementation type elements as parameter.

interface Java{

}

class CoreJava implements Java{

}

class AdvJava implements Java{

}

class Course<T extends Java>{

}

Course<Java> c1 = new Course<>(); valid

Course<CoreJava> c2 = new Course<>(); valid

Course<AdvJava> c3 = new Course<>(); valid

\*\*Note: Bounded parameters constraints at class level are not allowing

"implements" keyword and "super" keyword or any type of wildcards(?).

Note: In generic classes , we can use more than one type as bounded

parameter by using '&' operator.

class Demo<T extends Number & Serializable>{

}

--here Demo class is able to allow the elements which must be either same

as Number or sub class of Number class and must implements

Serializable interface.

Generic method:

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

--in the MyGen class example we place get() method inside our generic class

as a generic method.

--but we can declare a generic method inside a non-generic class also.

example:

Demo.java:

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

package com.masai;

public class Demo {

public static <T> void fun1(T t) {

System.out.println(t);

}

public static void main(String[] args) {

fun1(0);

fun1("Hello");

fun1(new Student(10, "n1", 500));

}

}

example 2: printing any type of elements:

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

package com.masai;

public class Demo {

public static <T> void fun1(T[] tr) {

for(T t1:tr) {

System.out.println(t1);

}

}

public static void main(String[] args) {

//Primitives are not supported

//int[] nums= {10,20,30,40};

Integer[] nums= {10,20,30,40};

String[] str= {"a","b","c"};

fun1(str);

fun1(nums);

}

}

Day2

Generic class: inside this we may have generic methods as well as non generic methods also.

Generic method: it could be inside a non-generic class also.

example1:

package com.masai;

public class Demo {

public static <T> void fun1(T t) {

System.out.println(t);

}

public static <K,V> void fun2(K k, V v) {

System.out.println(k+"========"+v);

}

public static void main(String[] args) {

fun2(10, 20);

fun2("Hello",20);

fun2(10, new Student(10, "N1", 500));

}

}

with return type:

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

public static <K,V> V fun2(K k, V v) {

System.out.println(k+"========"+v);

return v;

}

Wildcard in generics:

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

--It is mostly used with Collection type parameters inside a method.

--we can not use wildcard with class level.

wildcard is used in 3 ways:

1.upper bound wildcard: --> ? extends Type

2.lower bound wildcard: ---> ? super Type

3.unbounded wildcard ----> ?

unbounded wildcard:

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

public void fun1(List<?> list){

}

example :

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<?> list) {

System.out.println("inside fun1...");

//we can access the values from the unbounded type by using Object class

for(Object i1:list) {

System.out.println(i1);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<String> slist= Arrays.asList("a","b","c","d");

fun1(ilist);

fun1(slist);

}

}

--the above example (unbounded type) is simillar to not applying generics

ex:

public static void fun1(List list) {

}

--the above code we write wituout the wildcard also

ex:

public static <T> void fun1(List<T> list) {

}

Upper bound wildcard:

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

example:

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? extends Number> list) {

System.out.println("inside fun1...");

for(Number n1:list) {

System.out.println(n1);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<String> slist= Arrays.asList("a","b","c","d");

fun1(ilist);

//fun1(slist);

}

}

example2:

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? extends Person> list) {

System.out.println("inside fun1...");

for(Person p:list) {

System.out.println(p);

p.show();

}

}

public static void main(String[] args) {

List<Student> students= Arrays.asList(

new Student(12, "N1", 500),

new Student(13, "N2", 500),

new Student(14, "N3", 500),

new Student(15, "N4", 500)

);

fun1(students);

}

}

Lower bound wildcard:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? super Integer> list) {

System.out.println("inside fun1...");

for(Object obj:list) {

System.out.println(obj);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<Number> nlist= Arrays.asList(10,20,30,40);

List<Object> olist= Arrays.asList(10,20,30,40);

List<Double> dlist= Arrays.asList(10.55,20.44,30.33,40.33);

fun1(olist);

fun1(nlist);

fun1(ilist);

//fun1(dlist); //CE

}

}

example 2:

public static void fun1(List<? super Student> list) {

System.out.println("inside fun1...");

for(Object obj:list) {

System.out.println(obj);

}

}

--here we can call the above method by passing List of Student or

List of Person or List of Object.

Map:

=====

--In Map, we group multiple objects in the form of kay-value pair.

--each key-value pair is known as an Entry. so we can say Map is a collection of Entry.

--here both key and value should be an object.(primitive are not allowed)

--key should not be duplicate , but value can be duplicate.

with Map we get super fast loopup operation.

example:

state : capital

state : List of cities

country : president

manager: List of employee

mobile\_brand: list of models.

public V put(K key, V value) : if entry is inserted it returns null value.

map.put(10,"delhi"); // null

map.put(20,"chennai"); // null

map.put(10,"pune"); // delhi , it returns overwritten value

public Set<K> keySet() ; it will return all the Key in the form of Set

public Collection<V> values() ; it will return all the values in the form of Collection object

public Set<Map.Entry<K,V>> entrySet() ;; it will return all the key-value pair(Entry) in the form of Set.

--A Map is considered as a group of Entry.

--without existing a Map obj there is no-chance of existing an Entry obj.

hence an Entry is a inner interface of a Map interface.

i.e Entry interface is defined inside the Map interface.

package java.util;

interface Map{

public V put(K k, V v);

interface Entry{

Object getKey();

Object getValue();

Object setValue(Object value)

}

}

interface :

abstract methods

static and final variables

default method{}

static method{}

class

interface

Map m=

Map.Entry me=

Entry e = // invalid declaration

interface Math{

interface Algebra{

}

interface Gemetry{

}

interface calcules{

}

}

class AlgebraImpl implements Math.Algebra{

}

HashMap:

=========

--elements will be inserted based on the hashCode value of the key, so

it does not follow the sequece.

--key should not be duplicate, value can be duplicate.

--we can add only one null as a key, but any number of null as value.

example:

HashMap<Integer, String> hm=new HashMap<>();

System.out.println(hm);

System.out.println(hm.put(10, "del"));//null

System.out.println(hm.put(10, "mum"));//del

System.out.println(hm);// {10-mum}

example2:

HashMap<Integer, String> hm=new HashMap<>();

hm.put(10, "del");

hm.put(20, "mum");

hm.put(30, "che");

hm.put(40, "del");

hm.put(null, "pun");

hm.put(null, "chandigarh");

System.out.println(hm);

key as String and value as List:

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

HashMap<String, List<String>> hm = new HashMap<>();

List<String> cities = new ArrayList<>();

cities.add("kolkata");

cities.add("haldia");

cities.add("durgapur");

hm.put("Maharastra",Arrays.asList("pune","mumbai","amrawati") );

hm.put("WestBengal", cities);

public Map<String, Student> sortMapUsingValue( HashMap<String, Student> hm){

//

//get the Set<Map.Entry <String,Student>>

//convert this set to the List

Collections.sort(map, )

Map.Entry me1, Map.Entry me2 //write the sorting logic

//create a LinkedHashMap class and add each entry inside this.

}

example

hm.put("Maharastra",new Student(10,"Ganesh",950));

hm.put("Tamilnadu",new Student(12,"Surya",850));

hm.put("Telangana",new Student(15,"Venkat",920));

hm.put("Haryana",new Student(16,"Dinesh",910));

hm.put("Kerla",new Student(18,"Srinu",880));

Map<String,Student> resultMap= sortMapUsingValue(hm);

{

Tamilnadu=new Student(12,"Surya",850),

Kerla=new Student(18,"Srinu",880),

Haryana= new Student(16,"Dinesh",910),

Telangana = new Student(15,"Venkat",920)

Maharastra = new Student(10,"Ganesh",950)

}

Note: if inside a HashMap or LinkedHashMap we try to add any key as a user-defined object, then it is always recomended that, we should override equals() and hashCode() method inside that class, which is used as a key.

where as inside the TreeMap, the key object should implement Comparable interface or we need to use Comparator.

Day3

Generic class: inside this we may have generic methods as well as non generic methods also.

Generic method: it could be inside a non-generic class also.

example1:

package com.masai;

public class Demo {

public static <T> void fun1(T t) {

System.out.println(t);

}

public static <K,V> void fun2(K k, V v) {

System.out.println(k+"========"+v);

}

public static void main(String[] args) {

fun2(10, 20);

fun2("Hello",20);

fun2(10, new Student(10, "N1", 500));

}

}

with return type:

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

public static <K,V> V fun2(K k, V v) {

System.out.println(k+"========"+v);

return v;

}

Wildcard in generics:

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

--It is mostly used with Collection type parameters inside a method.

--we can not use wildcard with class level.

wildcard is used in 3 ways:

1.upper bound wildcard: --> ? extends Type

2.lower bound wildcard: ---> ? super Type

3.unbounded wildcard ----> ?

unbounded wildcard:

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

public void fun1(List<?> list){

}

example :

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<?> list) {

System.out.println("inside fun1...");

//we can access the values from the unbounded type by using Object class

for(Object i1:list) {

System.out.println(i1);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<String> slist= Arrays.asList("a","b","c","d");

fun1(ilist);

fun1(slist);

}

}

--the above example (unbounded type) is simillar to not applying generics

ex:

public static void fun1(List list) {

}

--the above code we write wituout the wildcard also

ex:

public static <T> void fun1(List<T> list) {

}

Upper bound wildcard:

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

example:

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? extends Number> list) {

System.out.println("inside fun1...");

for(Number n1:list) {

System.out.println(n1);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<String> slist= Arrays.asList("a","b","c","d");

fun1(ilist);

//fun1(slist);

}

}

example2:

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? extends Person> list) {

System.out.println("inside fun1...");

for(Person p:list) {

System.out.println(p);

p.show();

}

}

public static void main(String[] args) {

List<Student> students= Arrays.asList(

new Student(12, "N1", 500),

new Student(13, "N2", 500),

new Student(14, "N3", 500),

new Student(15, "N4", 500)

);

fun1(students);

}

}

Lower bound wildcard:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void fun1(List<? super Integer> list) {

System.out.println("inside fun1...");

for(Object obj:list) {

System.out.println(obj);

}

}

public static void main(String[] args) {

List<Integer> ilist= Arrays.asList(10,20,30,40);

List<Number> nlist= Arrays.asList(10,20,30,40);

List<Object> olist= Arrays.asList(10,20,30,40);

List<Double> dlist= Arrays.asList(10.55,20.44,30.33,40.33);

fun1(olist);

fun1(nlist);

fun1(ilist);

//fun1(dlist); //CE

}

}

example 2:

public static void fun1(List<? super Student> list) {

System.out.println("inside fun1...");

for(Object obj:list) {

System.out.println(obj);

}

}

--here we can call the above method by passing List of Student or

List of Person or List of Object.

Map:

=====

--In Map, we group multiple objects in the form of kay-value pair.

--each key-value pair is known as an Entry. so we can say Map is a collection of Entry.

--here both key and value should be an object.(primitive are not allowed)

--key should not be duplicate , but value can be duplicate.

with Map we get super fast loopup operation.

example:

state : capital

state : List of cities

country : president

manager: List of employee

mobile\_brand: list of models.

public V put(K key, V value) : if entry is inserted it returns null value.

map.put(10,"delhi"); // null

map.put(20,"chennai"); // null

map.put(10,"pune"); // delhi , it returns overwritten value

public Set<K> keySet() ; it will return all the Key in the form of Set

public Collection<V> values() ; it will return all the values in the form of Collection object

public Set<Map.Entry<K,V>> entrySet() ;; it will return all the key-value pair(Entry) in the form of Set.

--A Map is considered as a group of Entry.

--without existing a Map obj there is no-chance of existing an Entry obj.

hence an Entry is a inner interface of a Map interface.

i.e Entry interface is defined inside the Map interface.

package java.util;

interface Map{

public V put(K k, V v);

interface Entry{

Object getKey();

Object getValue();

Object setValue(Object value)

}

}

interface :

abstract methods

static and final variables

default method{}

static method{}

class

interface

Map m=

Map.Entry me=

Entry e = // invalid declaration

interface Math{

interface Algebra{

}

interface Gemetry{

}

interface calcules{

}

}

class AlgebraImpl implements Math.Algebra{

}

HashMap:

=========

--elements will be inserted based on the hashCode value of the key, so

it does not follow the sequece.

--key should not be duplicate, value can be duplicate.

--we can add only one null as a key, but any number of null as value.

example:

HashMap<Integer, String> hm=new HashMap<>();

System.out.println(hm);

System.out.println(hm.put(10, "del"));//null

System.out.println(hm.put(10, "mum"));//del

System.out.println(hm);// {10-mum}

example2:

HashMap<Integer, String> hm=new HashMap<>();

hm.put(10, "del");

hm.put(20, "mum");

hm.put(30, "che");

hm.put(40, "del");

hm.put(null, "pun");

hm.put(null, "chandigarh");

System.out.println(hm);

key as String and value as List:

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

HashMap<String, List<String>> hm = new HashMap<>();

List<String> cities = new ArrayList<>();

cities.add("kolkata");

cities.add("haldia");

cities.add("durgapur");

hm.put("Maharastra",Arrays.asList("pune","mumbai","amrawati") );

hm.put("WestBengal", cities);

public Map<String, Student> sortMapUsingValue( HashMap<String, Student> hm){

//

//get the Set<Map.Entry <String,Student>>

//convert this set to the List

Collections.sort(map, )

Map.Entry me1, Map.Entry me2 //write the sorting logic

//create a LinkedHashMap class and add each entry inside this.

}

example

hm.put("Maharastra",new Student(10,"Ganesh",950));

hm.put("Tamilnadu",new Student(12,"Surya",850));

hm.put("Telangana",new Student(15,"Venkat",920));

hm.put("Haryana",new Student(16,"Dinesh",910));

hm.put("Kerla",new Student(18,"Srinu",880));

Map<String,Student> resultMap= sortMapUsingValue(hm);

{

Tamilnadu=new Student(12,"Surya",850),

Kerla=new Student(18,"Srinu",880),

Haryana= new Student(16,"Dinesh",910),

Telangana = new Student(15,"Venkat",920)

Maharastra = new Student(10,"Ganesh",950)

}

Note: if inside a HashMap or LinkedHashMap we try to add any key as a user-defined object, then it is always recomended that, we should override equals() and hashCode() method inside that class, which is used as a key.

where as inside the TreeMap, the key object should implement Comparable interface or we need to use Comparator.

Day4

--In java we have all together 5 keywords in the concept of exception handling:

1.try

2.catch

3.throw

4.throws

5.finally

--Exception classes are categorized in 2 category:

1.checked exception (checked by the java compiler whether we have handled that exception class object or not at compile time)

2.unchecked exception (not checked by the compiler whether we handled exception class object or not)

\*\*Note: whether exceptions is a checked or unchecked exception, exceptions always occurs at runtime, it never occurs at compile time, compile time only occurs compilation error.

kid ------------------mother-----------------------------> exam

.java -----------------java compiler ---------------------> .class ---------------->jvm ------------> execute that class

responsibility of java compiler:

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

1. convert .java(source code ) to the .class(bytecode)

2. it will scan our .java file and check any kind of syntax error, and generates a compile time error.

3.if inside the class we don't have any constructor then java compiler

provides a default constructor to the .class.

4.for each and every statement which may have any logical error,

compiler will check whether we are handling the corresponding

exception class or not.

--in the process of checking, compiler will not bother even we don't handle

object of some exception classes.

but there are some statments, for which applying the exception handling concept is mandatory at compile time only, if we don't handle them, then compiler will generate a compile time error.

--the exception class object forwhich compiler force you to handle at compile time only is knwon as checked exception,

ex:

ClassNotFoundException

IOException

SQLException

FileNotFoundException

and for those exception classes for which compiler will not force to handle (it ignores at compile time) is knwon as unchecked exception / runtime exceptions

ex:

NullPointerException

ArithmaticException

etc.

Object

|

Throwable

|

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

| |

Exception (checked exception) Error

|

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

| | | |

RuntimeException ClassNotFoundException SQLException IOException , etc

|

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

| | | | |

AE AIOBE CCE NPE NFE etc

--the classes which are direct child of the Exception class (except RuntimeException class) is known as checked exception class (including Exception class also)

--the classes which are the direct child of RuntimeException including the RuntimeException class is known as

unchecked exception.

throw keyword:

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

--with 'throw' keyword we can generate the exception class object

in out program explicitly.

--generally JVM will create the exception class object, whenever a logical

error occurs in our application, and put/throw that exception class

object inside our application (at the same statement where logical error occurs.)

--but sometime, it is required that programmer need to create the exception class object explicilty and thow that exception class object inside the application manually, whenever a logical error occurs due to the business logic violation.

Note: this business logic violation error related logical error, is not a logical error to the point of view of JVM.

example:

Demo.java:

===========

package com.masai;

public class Demo{

public int getPension(int age, int salary) {

int pension = 0;

if(age > 40 && age < 100)

pension = (age \* salary)/100;

else {

ArithmeticException ae = new ArithmeticException("Invalid Age :" + age);

throw ae;

//this exception is generated based on logical error occurs due to the business logic violation

}

return pension;

}

public static void main(String[] args) {

System.out.println("inside main of Demo");

Demo d1= new Demo();

try {

int result= d1.getPension(60, 50000);

System.out.println(result);

}catch (ArithmeticException ae) {

System.out.println(ae.getMessage());

}

System.out.println("end of main...");

}

}

throws keyword:

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

--this keyword is used along with method signature to announce the caller that the method may throw/generate an exception class object at calling place.

package com.masai;

public class Demo{

public int getPension(int age, int salary)throws ArithmeticException {

int pension = 0;

if(age > 40 && age < 100)

pension = (age \* salary)/100;

else {

ArithmeticException ae = new ArithmeticException("Invalid Age :" + age);

throw ae;

//this exception is generated based on logical error occurs due to the business logic violation

}

return pension;

}

public static void main(String[] args) {

System.out.println("inside main of Demo");

Demo d1= new Demo();

try {

int result= d1.getPension(10, 50000);

}

catch (ArithmeticException ae) {

System.out.println(ae.getMessage());

}

System.out.println(result);

//5000

System.out.println("end of main...");

}

}

Rule 1:

--if we throwing any unchecked excption class object from a method

then announing that exception with method signature is optional.

otherwise(if we throw any checked exception class obj ) then it is mandatory.

Rule 2: if a method throws any checked excpetion then caller have 2 choices

at compile time:

1. caller need to wrap the method call inside try-catch block (handling the excpetion ) it is mandatory.

2. caller can delegete the exception handling duty to its own caller by using "throws" keyword (skipping the exception handling duty)

\*\*\*Note: we only handle the excption using try-catch block , using throws keyword

we just skip the exception handling duty and delegate it to the caller.

--we can apply the throws keyword with the main method also, but in realtime application it is never recomended to use throws keyword along with main method. becoz doing so ,we are ready to accept abnormal termination of our application.

example Demo.java:

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

package com.masai;

public class Demo{

public int getPension(int age, int salary)throws ClassNotFoundException{

int pension = 0;

if(age > 40 && age < 100)

pension = (age \* salary)/100;

else {

//ArithmeticException ae = new ArithmeticException("Invalid Age :" + age);

ClassNotFoundException cnfe = new ClassNotFoundException("Invalid Age :"+age);

//throw ae;

throw cnfe;

//this exception is generated based on logical error occurs due to the business logic violation

}

return pension;

}

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

System.out.println("inside main of Demo");

Demo d1= new Demo();

int result= d1.getPension(10, 50000);

System.out.println(result);

//5000

System.out.println("end of main...");

}

}

multiple excpetions using throws:

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

package com.masai;

import java.io.FileNotFoundException;

public class Demo{

public void fun1()throws ClassNotFoundException,FileNotFoundException {

System.out.println("inside fun1 ...");

}

public static void main(String[] args) {

System.out.println("inside main of Demo");

Demo d1= new Demo();

try {

d1.fun1();

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

} catch (FileNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("end of main...");

}

}

Note: whnever we want to make excpetion handling duty mandatory to the caller then throw the checked exception class object otherwise(if we make exception handing duty to the caller is optional) then throw the unchecked exception class object.

user-defined exception:

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

--if we want we can create our own user-defined exception classes as well.

InvalidAgeException.java:

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

package com.masai;

//checked exception

public class InvalidAgeException extends Exception{

public InvalidAgeException() {

// TODO Auto-generated constructor stub

}

public InvalidAgeException(String message) {

super(message);

}

}

\*\*\*--in order to create unchecked exception we need to extends from RuntimeException class.

Demo.java:

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

package com.masai;

import java.io.FileNotFoundException;

public class Demo{

public int getPension(int age, int salary)throws InvalidAgeException {

int pension = 0;

if(age > 40 && age < 100)

pension = (age \* salary)/100;

else {

//ArithmeticException ae = new ArithmeticException("Invalid Age :" + age);

//throw ae;

//ClassNotFoundException cn = new ClassNotFoundException("Invalid Age :" + age);

//throw cn;

InvalidAgeException ie=new InvalidAgeException("Invalid Age :"+age);

//this exception is generated based on logical error occurs due to the business logic violation

}

return pension;

}

public static void main(String[] args) {

System.out.println("inside main of Demo");

Demo d1= new Demo();

try {

int result= d1.getPension(10, 50000);

System.out.println(result);

}

catch (InvalidAgeException ae) {

System.out.println(ae.getMessage());

}

//5000

System.out.println("end of main...");

}

}

Day5

Nested try catch block:

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

--we can use try-catch inside a try block or a catch block and even in finally block also.

Demo.java:

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

package com.masai;

import java.io.FileNotFoundException;

public class Demo{

public static void main(String[] args) {

System.out.println("start of main");

try {

System.out.println("inside try 1");

System.out.println(100/10);

try {

System.out.println("inside try2");

A a1=null;

a1.funA();

System.out.println("end of try2");

} catch (NullPointerException npe) {

System.out.println("inside catch of try2");

}

System.out.println("end of try 1");

} catch (Exception e) {

System.out.println("catch of try1 ");

}

System.out.println("end of main...");

}

}

output:

start of main

inside try 1

10

inside try2

inside catch of try2

end of try 1

end of main...

Note: if inner try is unnable to handle the exception then that exception

is propagated to the outer try block, by suspending the remaining statement of

inner try block.

example:

Demo.java:

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

package com.masai;

import java.io.FileNotFoundException;

public class Demo{

public static void main(String[] args) {

System.out.println("start of main");

try {

System.out.println("inside try 1");

System.out.println(100/10);

try {

System.out.println("inside try2");

A a1=null;

a1.funA();

System.out.println("end of try2");

} catch (NumberFormatException nfe) {

System.out.println("inside catch of try2");

}

System.out.println("end of try 1");

} catch (Exception e) {

System.out.println("catch of try1 ");

}

System.out.println("end of main...");

}

}

output:

start of main

inside try 1

10

inside try2

catch of try1

end of main...

throws in constrcutor:

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

--since constructor is also a block of code, and there might be a chance of logical error.

--so constructor can also throw an exception.

exmaple

Demo.java:

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

package com.masai;

public class Demo{

public Demo()throws InvalidAgeException {

System.out.println("inside constructor of Demo");

}

public static void main(String[] args) {

System.out.println("start of main");

try {

Demo d1 = new Demo();

}catch (InvalidAgeException e) {

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

}

System.out.println("end of main...");

}

}

so if any constructor throws a checked exception then we need to

create object of that class inside the try-catch block or we can skip

the excption handling duty by using throws keyword.

throws with inheritance:

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

--if a parent class constructor throws any checked exception then

inorder to extends that class we need to resolve the parent class

constructor by calling the parent class constructor from the child

class constructor.

ex:

A.java:

----------

package com.masai;

public class A {

int i=10;

public A()throws InvalidAgeException {

System.out.println("inside constructor of A");

}

void funA() {

System.out.println("inside funA of A");

}

}

Demo.java:

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

package com.masai;

public class Demo extends A{

public Demo() throws InvalidAgeException {

super();

}

public static void main(String[] args) {

System.out.println("start of main");

try {

Demo d1= new Demo();

}catch (InvalidAgeException in) {

// TODO: handle exception

}

System.out.println("end of main...");

}

}

method overriding rules with the exception handling:

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

--if a parent class method throws any checked exception, then

while overriding that method inside the child class we have

following options:

1.child class overrident method may not throw any kind of exception

2.1.child class overriden method may throw the same exception

3.may throw any kind of unchecked exception

But child class overriden method can not throw super type(parent) of

specified exception.

finally block:

===========

--whether try block is executing or catch block is executing finally must execute.

--irrespective of try and catch if we want to execute some code then we can use finally

block.

--a fianlly block used with try and catch block only.

--we can use a finally block without a catch block also, i.e try with finally (but

not recomended to use).

example :

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

System.out.println("start of main");

try {

//connecting to the DB server

//Connection conn =

//after getting the DB connection,we need to perform DB operatiosn

System.out.println("100/0");

//after performing the DB opration we should close the connection

}catch (Exception e) {

System.out.println("inside catch");

}

//close the connection // by doing this we

//violate 2 coding principles 1. coding modularity,

//2 coding integrity.

//coding modulatity: related code should be in realted unit.

//comding integrity: one logic should not mixup with other logic

System.out.println("end of main...");

}

}

from java 1.7 onwards, for closing the DB connection we need not take a finally block, we can use

try with resource feature:

try("create a connection") { // try with resourse

//connecting to the DB server

//Connection conn =

//after getting the DB connection,we need to perform DB operatiosn

System.out.println("100/0");

//after performing the DB opration we should close the connection

}catch (Exception e) {

System.out.println("inside catch");

}

Exception rethrowing:

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

converting one exception to another exception object.

example:

Demo.java:

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

package com.masai;

public class Demo{

public void fun1(int num1, int num2)throws InvalidAgeException {

try {

int result = num1/num2;

System.out.println("The Result is :"+result);

}catch (ArithmeticException ae) {

System.out.println(ae.getMessage());

InvalidAgeException ie= new InvalidAgeException("Num2 should not be 0");

throw ie;

}

}

public static void main(String[] args) {

System.out.println("start of main");

Demo d1 = new Demo();

try {

d1.fun1(100, 0);

} catch (InvalidAgeException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

System.out.println("end of main...");

}

}

Functional Programming:

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

--this concept is introduced in java 1.8 onwards.

--in this type of paradigm, a function is trated as a value. (we can assign the

entire funtion in a variable, or we can pass a function to another function parameter

or we can return a function from another function as well.)

int i =10;

--the main adv of the FP is less coding, polymorphic and easy to understand.

--to achive the functional programming in java we need a functional interface.

functional interface in java:

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

--an interface which has only one abstract method is called a FI.

--A FI may have n number of static and default methods.

--A FI may contains some data members (variables) also.

--A FI can have Object class related abstract methods also.

example:

Intr.java:

-----------

@FunctionalInterface

interface Intr{

void fun1();

}

@FunctionalInterface annotation make sure we have a valid FI.

example2:

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

Intr.java:

-----------

package com.masai;

@FunctionalInterface

public interface Intr {

int x=10;

public abstract void fun1();

public abstract String toString();

static void fun2() {

}

default void fun3() {

}

}

some of the Predefined functional interface in java:

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

java.lang.Comparable : public int compareTo(Object obj);

java.util.Comparator : public int compare(Object obj1, Object obj2);

java.lang.Iterable : public Iterator iterator();

java.lang.Runnable : public void run();

\*\*\*Note: with the help of FI we achive FP in java using Lambda expression.

example:

Intr.java:

----------

package com.masai;

@FunctionalInterface

public interface Intr {

void sayHello(String name);

}

IntrImpl.java:

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

package com.masai;

public class IntrImpl implements Intr{

@Override

public void sayHello(String name) {

System.out.println("Welcome "+name);

}

}

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

//using a external or seperate class

Intr i1= new IntrImpl();

i1.sayHello("Ravi");

//using Annonymous inner class

//we can use Annonymous inner class without a FI also

Intr i2= new Intr() {

@Override

public void sayHello(String name) {

System.out.println("Welcome using Annonymous inner class" +name);

}

};

i2.sayHello("Amit");

//using Lambda expression:

Intr i3= name -> System.out.println("Welcome Using LE "+name);

i3.sayHello("Ram");

}

}

Day6

Lambda Expression:

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

--it is an expression using which we can provide the implementation of a FI.

LE comprises 3 things:

1. parameter (here data type is optional ) : if only one parameter is

then () small bracket is also optional, small () bracket is mandatory when we

have zero or more than one parameter.

2. lambda operator : ->

3.method body : if we write only one statement inside the implementation

body then {} is optional

Note: LE does not consider the method name of FI.

example :

Intr.java:

-----------

package com.masai;

@FunctionalInterface

public interface Intr {

void sayHello(String name,int age);

}

Demo.java:

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

package com.masai;

import java.util.jar.Attributes.Name;

public class Demo{

public static void main(String[] args) {

//using Lambda expression:

Intr i1= (String name, int age) -> {

System.out.println("Name is "+name+"----"+"Age is :"+age);

};

i1.sayHello("Name1", 25);

Intr i2= (x,y) -> System.out.println("Name is "+x+"----"+"Age is :"+y);

i2.sayHello("Name2", 30);

}

}

LE with method return type:

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

--if inside the method body only one statement is there then {} is optional

, and return keyword is not allowed.

--return keyword is allowed only with {} bracket.

example:

Intr.java:

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

package com.masai;

@FunctionalInterface

public interface Intr {

String fun1(int num);

}

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

Intr i1= num -> "This is the number "+num;

//String result= i1.fun1(10);

//System.out.println(result);

System.out.println(i1.fun1(20));

}

}

example 3:

Intr.java:

-----------

package com.masai;

@FunctionalInterface

public interface Intr {

Student createStudentObject(int roll, String name, int marks);

}

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

Intr i1= (r,n,m) -> new Student(r, n, m);

Student s1= i1.createStudentObject(1, "Name1", 780);

System.out.println(s1);

}

}

example 4: sorting a List of Student according to the marks:

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

package com.masai;

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 500));

students.add(new Student(12, "N2", 400));

students.add(new Student(13, "N3", 520));

students.add(new Student(14, "N4", 450));

students.add(new Student(15, "N5", 410));

//using external class

//Collections.sort(students,new StudentMarksComp());

//using annonymous inner class

Collections.sort(students,new Comparator<Student>() {

@Override

public int compare(Student s1, Student s2) {

return s1.getMarks() > s2.getMarks() ? +1: -1;

}

});

//using LE

Collections.sort(students, (s1,s2) -> s1.getMarks() > s2.getMarks() ? +1: -1);

}

}

LE as a method parameter:

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

package com.masai;

@FunctionalInterface

public interface Intr {

void sayHello();

}

Demo.java:

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

package com.masai;

public class Demo{

void fun1(Intr i1) {

System.out.println("inside fun1 of Demo");

System.out.println(i1);

i1.sayHello();

}

public static void main(String[] args) {

Demo d1 = new Demo();

d1.fun1(() -> System.out.println("Welcome to LE"));

}

}

Method reference:

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

--it is the simplified form (short-cut) of LE.

-- it is represented by using :: double colon symbol.

--instead of creating a LE with all the details , with the help of MR we can

refer an existing class method to the functional interface variable.

Note:

we can take a reference of a static method using Classname::methodName;

we can take a reference of a non-static method using object::methodName;

we can take a reference of a constructor also using Classname::new;

\*\* in case of constructor method should not have any return type

Demo.java:

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

package com.masai;

public class Demo{

public static void funX() {

System.out.println("inside funX of Demo");

System.out.println("other statments of Demo");

//5000

}

public static void main(String[] args) {

Intr i1 = () -> System.out.println("Welcome to LE");

i1.sayHello();

Intr i2 = Demo::funX;

i2.sayHello();

}

}

if funX is non-static :

Demo d1= new Demo();

Intr i2 = d1::funX;

//Intr i2 = new Demo()::funX;

i2.sayHello();

example :

Intr.java:

-----------

package com.masai;

@FunctionalInterface

public interface Intr {

void sayHello(String name);

}

Demo.java:

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

package com.masai;

public class Demo{

public static void funX(String n) {

System.out.println("inside funX of Demo "+n);

//5000

}

public void funY(String n) {

System.out.println("inside funY of Demo "+n);

//5000

}

public Demo(String n) {

System.out.println("inside constructor of Demo "+n);

}

public Demo() {

// TODO Auto-generated constructor stub

}

public static void main(String[] args) {

Intr i1 = Demo::funX;

Intr i2 = new Demo()::funY;

Intr i3= Demo::new;

Intr i4 = n -> System.out.println("Using LE "+n);

i1.sayHello("Name1");

i2.sayHello("Name2");

i3.sayHello("Name3");

i4.sayHello("Name4");

}

}

example:

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

Intr.java:

-----------

package com.masai;

@FunctionalInterface

public interface Intr {

int getTheNumber(String number);

}

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

// Intr i1 = snum -> {

//

// int result= Integer.parseInt(snum);

//

// return result;

//

// };

Intr i1 = snum -> Integer.parseInt(snum);

System.out.println(i1.getTheNumber("100") + 200);

Intr i2 = Integer::parseInt;

System.out.println(i2.getTheNumber("200")+ 300);

}

}

example2:

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

Intr.java:

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

package com.masai;

@FunctionalInterface

public interface Intr {

void printSomething(String message);

}

Demo.java:

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

package com.masai;

public class Demo{

public static void main(String[] args) {

Intr i1 = m -> System.out.println(m);

i1.printSomething("Welcome");

Intr i2= System.out::println;

i2.printSomething("Hello");

}

}

Some of the functional interface introduced in Java 8 to perform functional programming in Java:

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

--these interfaces belongs to "java.util.function" package.

1.Predicate(I)

2.Consumer(I)

3. Supplier(I)

4. Function(I)

1.Predicate(I):

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

this interface has only one abstract method :

interface Predicate<T>{

public boolean test(T t);

}

--this test() method checks whether supplied object is satisfying a condition or not.

example: test a Student object whether his marks is greater than 500 or not.

MyPredicate.java:

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

package com.masai;

import java.util.function.Predicate;

public class MyPredicate implements Predicate<Student>{

@Override

public boolean test(Student s) {

// if(s.getMarks() > 500)

// return true;

// else

// return false;

return s.getMarks() > 500;

}

}

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.function.Predicate;

public class Demo{

public static void main(String[] args) {

Predicate<Student> p = new MyPredicate();

System.out.println(p.test(new Student(10, "n1", 480)));

Predicate<Student> p2 = s -> s.getMarks() > 500;

System.out.println(p2.test(new Student(10, "n1", 480)));

}

}

--from java 1.8 onwards inside the Collection interface one new method is added called

public boolean removeIf(Predicate filter);

--based on the the condition of Predicate, this method will remove/filter the

elements from the collection classes.

example :

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.List;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 780));

students.add(new Student(12, "N2", 480));

students.add(new Student(13, "N3", 680));

students.add(new Student(14, "N4", 380));

students.add(new Student(15, "N5", 580));

students.add(new Student(16, "N6", 720));

//students.removeIf(new MyPredicate());

students.removeIf(s -> s.getMarks() > 500);

for(Student s:students) {

System.out.println(s);

}

}

}

Day7

LE: it is an expression using which we can provide implementation of an FI.

--Using LE we can represent the object a FI.

2.java.util.function.Consumer<T>:

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

public void accept(T t);

--this method only accept the object of generric type and does not return anything/

MyConsumer.java:

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

package com.masai;

import java.util.function.Consumer;

public class MyConsumer implements Consumer<Student>{

@Override

public void accept(Student s) {

System.out.println("Roll is :"+s.getRoll());

System.out.println("Name is :"+s.getName());

System.out.println("Marks is :"+s.getMarks());

}

}

Demo.java:

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

package com.masai;

import java.util.function.Consumer;

public class Demo{

public static void main(String[] args) {

// Consumer<Student> c= new MyConsumer();

//

// c.accept(new Student(10, "N1", 500));

//

Consumer<Student> c2= s -> {

System.out.println("Roll is :"+s.getRoll());

System.out.println("Name is :"+s.getName());

System.out.println("Marks is :"+s.getMarks());

};

c2.accept(new Student(10, "N1", 500));

}

}

forEach method:

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

public void forEach(Consumer action); // action for each element of a collection

--this method is a default method belongs to Iterable interface.

--as we know that every collection is iterable (refer the Collection hirarchy diagram)

---so we can call this forEach method on any collection object.

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

import java.util.function.Consumer;

public class Demo{

public static void main(String[] args) {

List<String> names=Arrays.asList("Amit","Ravi","Sunil","Mukesh");

//normal for loop

//enhanced for loop

//Iterator

//ListIterator

names.forEach(name -> System.out.println(name.toUpperCase()));

}

}

example 2:

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

public class Demo{

public static void main(String[] args) {

List<Student> students= new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(20, "N2", 750));

students.add(new Student(30, "N3", 750));

students.add(new Student(40, "N4", 750));

students.forEach(s -> {

System.out.println(s);

//write that object to the File(Serialize the object)

});

}

}

3. java.util.function.Supplier<T>:

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

public T get();

example

MySupplier.java:

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

package com.masai;

import java.util.function.Supplier;

public class MySupplier implements Supplier<String>{

@Override

public String get() {

return "This message from the external class";

}

}

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.function.Supplier;

public class Demo{

public static int getANumber() {

return 1000;

}

public static void main(String[] args) {

Supplier<String> s= new MySupplier();

String str= s.get();

System.out.println(str);

Supplier<String> s2 = () -> "This message from the LE";

System.out.println(s2.get());

Supplier<Student> s3 = () -> new Student(10, "N1", 450);

System.out.println(s3.get());

Supplier<Integer> s4 = Demo::getANumber;

System.out.println(s4.get());

}

}

4.java.util.function.Function<T,R>:

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

public R apply(T t)

example :

Getting a Student object and returning the result of that student if marks > 500

return Pass otherwise return fail

example:

MyFunction.java:

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

package com.masai;

import java.util.function.Function;

public class MyFunction implements Function<Student, String>{

@Override

public String apply(Student s) {

// if(s.getMarks() > 500)

// return "Pass";

// else

// return "fail";

return s.getMarks() > 500 ? "Pass" : "fail";

}

}

Demo.java:

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

package com.masai;

import java.util.function.Function;

public class Demo{

public static void main(String[] args) {

Function<Student, String> f1= new MyFunction();

System.out.println(f1.apply(new Student(10, "N1", 450)));

Function<Student, String> f2 = s -> s.getMarks() > 500?"Pass": "Fail";

System.out.println(f2.apply(new Student(10, "N1", 850)));

}

}

Java Stream api:

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

--this api is introduced in java 1.8

--this api belongs to "java.util.stream" package

--this api is different from IO stream, this IO-stream api belongs to java.io package and java.nio package

here we represent flow of data between peripherals (input output devices) in the form of bytes or charecters.

this java.io stream represents flow of data in bytes or charecters

--this java.util.stream package contains some library classeas and interfaces

by using which we can perform functional style of programming on a

group of objects(Collection of data) in the form of Objects.

this java.util stream represents flow of data in the form of objects.

\*\*this api has one main interface :

java.util.stream.Stream(I)

Note: object of this Stream interface represents flow/sequence of objects

from a source like collection objects.

Feature of Stream :

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

1.stream does not store the elements, it only represents elements in a sequence

ex: wire does not hold/store the electicity

2.it represent only flow of objects , not the primitives.

3. operations (filtering/mapping,etc) performed on the stream object

does not modify its source.

ex: filtering a stream obtained from a source (collection) produces a

new stream with the filtered elements rather than removing the elements from the source collection.

4. with the help of stream object w can perform various operations on the collection data in functional style, like filterning some elements , printing some elements, transforming some elements,etc.

--Collection interface provides 2 methods to get a Stream object.

1. Stream<T> stream();

2. Stream<T> parrellalStream(); // this stream obj is used on multithreaded application.

methods of the Stream(I) interface:

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

there are 2 types of method in the Stream interface:

1. intermediate methods

2.terminal methods

1. intermediate methods: these methods returns a new Stream object,

instead of a final output.

--these methods never gives the final output.

some of the commonly used intermediate methods are:

map() , filter() method

2. terminal methods:- stream objects returns the final output only when

terminal method is called on the stream object.

these methods consumes that stream object. and after that we can not

re-use that stream obj again.

Note: if we try to use a consumed stream obj once again , then we will get

an exception.

some of the commnly used terminal methods are :

forEach(Consumer action) // similar to the Iterable interface forEach method

collect()

min()

max()

count()

get()

anyMatch()

allMatch()

example:

Demo.java:

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

package com.masai;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<String> list= Arrays.asList("one","two","three","four");

Stream<String> str1= list.stream();

str1.forEach(s -> System.out.println(s)); // terminal method

str1.forEach(s -> System.out.println(s)); // Runtime exception

}

}

filter() methods:

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

--it is one the intermediate method.

--this method takes a Predicate object as an argument ,and filter the stream

based on the Predicate condiction, and returns the filtered elements in another

stream object.

example:

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

//from the above list get another list of students whose marks is

//less that 500.

Stream<Student> str1= students.stream();

Stream<Student> str2= str1.filter(s -> s.getMarks() < 500);

str2.forEach(s -> System.out.println(s));

students.stream()

.filter(s -> s.getMarks() < 500)

.forEach(s -> System.out.println(s));

}

}

--creating another list based on filtered elements instead of printing them on the console.

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

//from the above list get another list of students whose marks is

//less that 500.

// Stream<Student> str1= students.stream();

//

// Stream<Student> str2= str1.filter(s -> s.getMarks() < 500);

//

// str2.forEach(s -> System.out.println(s));

List<Student> filteredList= students.stream()

.filter(s -> s.getMarks() < 500)

.collect(Collectors.toList());

System.out.println(students);

System.out.println(filteredList);

}

}

map() method:

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

--it is also a intermediate method.

--this method is used to transform the object.

--this method takes java.util.function.Function(I) object as an argument

and map/transform the element to a new element and returns the mapped

elements in another stream.

exmaple :

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

// Stream<Student> str1= students.stream();

//

//

// Stream<Student> str2= str1.map(s -> {

//

// Student s2 = new Student(s.getRoll(), s.getName(), s.getMarks()+50);

//

// return s2;

//

// });

//

// List<Student> modifiedStudents= str2.collect(Collectors.toList());

List<Student> modifiledList= students.stream()

.map(s -> new Student(s.getRoll(), s.getName(),s.getMarks()+50))

.collect(Collectors.toList());

modifiledList.forEach(s -> System.out.println(s));

}

}

min and max methods:

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

these methods are the terminal methods which will takes a Comparator

object, using which we can decide max and min elements.

--this min() and max() method will return the minimum and maximum

object in the form of "java.util.Optional" class object.

--this Optional class introduced in java1.8 and it is basically used to avoid the NullPointerException

--to get the element from this Optional class ,we need to call get() method.

ex:

Demo.java:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

// Stream<Student> str1= students.stream();

//

//

// Optional<Student> opt= str1.min((s1,s2) -> s1.getMarks() > s2.getMarks() ? +1:-1 );

//

// Student s= opt.get();

//

//

// System.out.println(s);

Student minStudent= students

.stream()

.min((s1,s2) -> s1.getMarks() > s2.getMarks() ? +1: -1)

.get();

System.out.println(minStudent);

}

}

count() method:

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

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

long result= students.stream().filter(s-> s.getMarks() < 500).count();

System.out.println(result);

}

}

allMatch() anyMatch, nonMatch()

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

--these methods takes the Predicate object and returns boolean

ex:

package com.masai;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import java.util.stream.Collectors;

import java.util.stream.Stream;

public class Demo{

public static void main(String[] args) {

List<Student> students = new ArrayList<>();

students.add(new Student(10, "N1", 750));

students.add(new Student(12, "N2", 450));

students.add(new Student(13, "N3", 650));

students.add(new Student(14, "N4", 850));

students.add(new Student(15, "N5", 410));

boolean result= students.stream().allMatch(s -> s.getMarks() < 1000);

System.out.println(result);

}

}

Day8

MultiThreading:-

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

---before learning about multithreading,we should know what nacessiated multithreading,for that we need to know about multitasking :-

consider the follwing program:-

class Test{

fun1()

{

--

--3 gb

--

}

fun2()

{

---

---

---

}

fun3()

{

---

---

---

}

psvm()

{

Test t=new Test();

t.fun1();

t.fun2();

t.fun3();

}

}

normally,if the class is compiled and jvm executes the program,then the order of excecution is that,first fun1() is called and after complete excution of fun1() control comebacks to the main() and then fun2() is called and so on...

---now let us assume that fun1() has some statment which involves data transfer,we know that data transfer is not a job of the processsor,it is a duty of a seperate individual circuit (DMA)(direct memory aceess) which functions under the control of the processor.

---since fun1() has data transfer statements,processor assigns that job to DMA.during this time the processor should remain idle.

----this made s/w developers to think that,efficiency of the processor would be increeased if processor is made to do some other useful work during this idle state.

the useful work is nothing but,executing other functions/methods present in the program

---this need lead to the concept of multitasking..

---Remember that processor is also an electric circuit and at any instance of time,it can execute only one statement.it can not execute multiple statement simulteniously...

mutitasking:-it is the concept of executing multiple tasks/functionality simulteniously.(functionality may be from same domain(same application) or from diff domain(diff applications).)

if we apply multitasking then a part of the one fun get executed then control switchs to the another fun now here also some part of second fun get executed then conrol switchs to the third fun

then again control comes back to the fun1 now it continues fun1 from where it had stopped earliar.

now when we see the output we feel that three fun executing simulteneously.

---thus,concept of multitasking come into existence to avoid the idle state of cpu...

---in mutitasking ,a part of a funtionality is executed one at a time

and it that part how many statement will be executed will be decided by scheduling.

Scheduling:-it is the process in which a specific time period is allocated to a fun where the control remains in that particular fun for that specific time period.

once time period lapsed control switches another fun with another time slice and so on.

Scheduling is supervised by the Scheduler(either OS scheduler or ThreadScheduler).

---Thread-Schedular in java is the part of jvm that decides which thread should run....

The time slice allocated will be in nano seconds.

thus by the time our eye recognize the execution of one part control switches to another part of the program.

Adv of multitasking:-

1.it is invented to avoid Idle states of the cpu.

2.make the fun get executed independetly.//mostly used becoz for a small project idle state of a cpu is not a big concern.

---animation along with form submition is a very good example of multitasking.

Multitasking is of two type:-

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

1.process based multi tasking

2.threadbased multitasking

---java supports thread based multi tasking.

process based multitasking:-

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

concept of executing more than one program simulteniously which r present in diff location of ram is known as process based multitasking.

here processer has to maintained address of both the program since control has to shift from one part of ram to another.

it incerase overhead on processor.

--here OS scheduler will perform the scheduling.

ThreadBased multitasking:-

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

concept of executing more than one fun simul belonging to the same memory domain is known as ThreadBased multitasking:-

---here thread-scheduler will do the scheduling

Note:- the main adv of the mt is increase performance and reduce the response time of the system(reduce the idle time of CPU or proper utilization of resources)

\*\*Application areas to apply multithreading

to develop multimedia graphics

to develop animations

to develop video games

to develop Webserver or ApplicationServer

when compared with other languages,developing multi-threded appl in java is very easy bcoz,java proviedes in-built support with rich api.

\*\*what is thread :-

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

--An application when it is under execution is called process.

--a thread is a part or sub process of an application.

--a thread is a seperate flow of execution that execute some functionality of a program with other part of program simulteniously.

Multithreaded application:-

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

--in java,every program/application has a default flow of execution,a defult thread,it is called as a main thread.if we can start another flow of execution(another thread) along with main thread simultenously then it is called a multithreaded application or program.

Implementing thread in java:-

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

impl thread in java is two step process:-

1.first of all we have to define a funtionality which can be executed as a thread along with the main thread(define job for a worker)

2.this fun should start as a thread.(assign job to worker)

----the signature of a fun using which we imple a thread(or what job a thread has to do) is defined in an interface by name Runnable

--this interface belongs to java.lang package.

this interface has only one method i.e

public void run();

for which we have to provide a body.

after providing body we need to execute this funtionality as thread(i.e simulteniously with the other part of the program).

--there is a class by name Thread present in java.lang package,which has a method called start(),this start() method is used to execute a given functionality as a seperate thread.

--this start methods recognize the run() method of the Runnable interface and then run() method is executed as a seperate individual thread.

---here Thread class is like a worker who has to start the job individually defined by run() method..in Runnable interface...

NOTE:-with the help of run() method we define a job that has to execute as thread,and with the help of Thread class start() method we need to start the job as a seperate individual thread.

---Thread class and Runnable interface r the two structure using which we imple Thread based multitasking in java.

interface Runnable{

public void run();

}

class Thread implements Runnable{

@Override

public void run(){

//Thread class internally implements Runnable interface and overrides the run() method with empty //implementation

}

other methods of Thread class

}

we imple threads either of the follwing two ways:

1.By implementing Runnable Interface

2.By extending Thread class itself..

1.class A imple Runnable{

@Override

public void run(){

--

}

--

}

2.class A extends Thread

{

@Override

public void run(){

--

}

--

}

----Internally thread class imple Runnable interface and override run() method with empty implementaion...

like:-

class Thread implements Runnable{

public void run(){

//it is empty body overriden from Runnable interface

}

public void start(){//this is thread class own method....

}

--

--//other methods of the Thread class

}

Note:-wheather we extends Thread class or imple Runnable inface dirctly,we have to use run() method of the Runnable interrface .

Thread class sudo code:

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

public class Thread implements Runnable {

@Override

public void run(){

//empty implementation

}

public void start(){

//registering our thread with the Thread-scheduler,

//perforing all the low level task to start a separate flow of execution.

//60000

run();

}

//other methods like (sleep, join, getName. getPriority)

}

Day9

Each thread is bydefault joined with as a last statement of the main method, this ensure that after completion of the threads then only main thread will terminate.

class A{

static sync funA(){ 2hour}

static sync funB(){}

sync funC(){} --object level --- t2{ 3hour }

funD(){}

static funE(){}

}

A a1=new A();

a1.funA(); ---class level----t1

--with the help of synch method we can access only one lock

(either object level or classs level)

--but with the sycnh block we can access multiple locks also.

Note: A thread can acquire multiple locks.

ex:

A a1=new A();

A a2= new A();

public void fun1(){

synchronized(A.class){

synchronized(B.class){

synchronized(a1){

synchronized(a2){

---//critical section (4 hour)

---

}

}

}

}

}

---> this critical section can be executed by only that thread

which is having 4 locks(2 class level and 2 object level)

Day12 sql

RDBMS:

=======

--Java is mostly used to develop business applications

Business Organizations:

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

Objectives of any BO, profit.

1. small scale business organizations (grossary shop, petrol pump)

2. large scale business organizations (HDFC bank, Indian railway, SBI,PEPSI, MASTERCARD)

Enterprises.

These BO provides their services to the client/customer. and to computerized those services what ever application we develop is known as business application.

Common general things in any business organizations:

1. store and maintain business data in a secure and easily retrival manner. (inside the RDBMS)

2.processing that data according to the business rule.

3.presenting the data in user-understandable format.

Data and information:

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

Data : it is a collection of raw and isolated facts.

Information : when we process the data , then we gets meaningfull results, this is called infromation.

Datastore:

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

it is a store where we can store or keep the data.

1. normal books and papers.

2. flat files in computer system(notepad, excel sheet, word files)

disadv of flat files:

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

1. data maintainace.

2. data redundency.

3. data integrity

4. security

5. data retrival

to overcome these problems we need to store the data inside the DBMS s/w (RDBMS s/w)

Database: it is a organized collection of interrelated data or structured collection of data.

DBMS is a type of s/w there we can mange multiple databases.

RDBMS: in this model, the data is stored in 2 dimentaional tables.

--we have multiple RDBMS s/w are there:

Mysql (Oracle) :- mysql workbench

Orcale s/w

postgres

ms-access

sql-server

db2

etc

RDBMS is an extension of DBMS s/w

Note: every RDBMS is a DBMS but reverse is not true.

--all the RDBMS s/w have a "DB-engine" which is the heart of the RDBMS s/w, and it is repsonsible to execute sql query.(that query could be suppied by either command line client or from mysql workbench)

--in order to work with RDBMS we need to use SQL (structured query language),it is an interface by using which we can work with any kind of RDBMS s/w.

--sql is a case insensetive language.

--sql language is a collection of predefined commads

these commands are categorized into following categories:

1.DDL (Data defination language)

(create, alter, drop, truncate, rename)

2. DML (Data manipulation Language)

(insert, update, delete)

3.DRL (Data Retrival language)

(select)

4.TCL (Transaction control language)

(commit, rollback, savepoint)

5. DCL (Decision Control language)

(grant revoke)

DBA : DDL , DCL

Java developer :(Select, Insert, update, delete ) ---C(create/insert (record creation))R(select retrieve a record)U (update a record)D(delete record) operation

>show databases;

>create database web19sb101db;

after creating the db inside the mysql rdbms s/w we need to move inside that db in order to work with that db.

>use web19sb101db;

>show tables;

>create table student(roll int,name varchar(12),marks int);

or

create table student

(

roll int,

name varchar(12),

marks int

);

>desc student;

>select \* from student;

1.DDL (Data defination language)

(create, alter, drop, truncate, rename)

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

Datatypes in mysql:

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

1.numeric types

2. string types

3. date and time types

1.numeric types

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

tinyint : 1byte

smallint --2 byte

mediumint - 3byte

int ---4byte

bigint--8byte

floating point:

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

float(6,2) :- the column can store 6 digit with 2 decimal places

2. string type:

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

1. char : fixed length of string range bt 0 to 255 char

2.varchar: variable length of string bt 1 to 65500, here we must define the length

ex:

char(4)

varchar(4)

value char(4) storage\_required

'a' -------> 4 bytes

'ab' -------> 4 bytes

'abcdef' ------> error data is too long

value varchar(4) storage\_required

'a' -------> 1 bytes

'ab' -------> 2 bytes

'abcdef' ------> error data is too long

Note: in the term of efficiency , if we r storing string with variable length then we should use varchar, and if the length is always fixed then we should use char, here char is slightly faster than varchar.

3 date and time:

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

a. date yyyy-mm-dd

b. datetime : yyyy-mm-dd hh:mm:ss

(1.create, 2.alter, 3.drop, truncate, rename)

2.alter: --it is used to change the strcuture of the existing table

--this command has 4 sub commands :

a.add

b. modify

c.drop

d.change

a. add: it is used to add the new cols in the existing table

ex:

> alter table student add address varchar(15);

b.modify : it is used to change the col datatype of col size.

ex:

> alter table student modify address varchar(20);

c. drop : to drop a single or multiple columns

ex:

>alter table student drop column address;

d. change:

--to rename a column

ex:

> alter table student change name sname varchar(12);

3. drop : to drop entire table .

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

>drop table student;

Note: it can not be rolledback. (DDL command can not be rolled back)

4. truncate : this command is used to truncate all the rows/records permanently from the table

--this command also can not be rolledback.

>truncate table student;

5.rename: it is used to rename table.

ex:

>rename table student to student1;

DML (insert,update,delete)

1.insert:

---------

inserting all columns values:

>insert into student values(10,'ram',780);

inserting partial columns values:

>insert into student(roll,name) values(10,'amit');

2.update:

--it is used to update the date within a table

ex1: updating all the marks for all the students.

> update student set marks=marks+50;

ex2: updating marks for only one student.

>update student set marks=marks+50 where name='ram';

>update student set marks=marks+50 where marks > 800;

>update student set name='mukesh' where roll=30;

3.delete:

-----------

--it is used to delete the records/rows from the table.

>delete from student; // it will delete entire record from the table like truncate command.

(this delete command we can rollback inside the transactional area, whereas truncate command we can not rollback).

>delete from student where name = 'amit';

DRL(select)

==========

--this command is used to quering a database tables.

syntax:

select col1,col2,..

from tablename

where condition

[groupby colname]

having condition

orderby colname [asc/desc]

ex1:

>select \* from student; // all the column and all the rows

ex2: restricting the rows by using where clause.

> select \* from student where roll =10;

>select sname from student;

>select sname from student where roll =10;

select (projecting columns )

from (from which tables(s) )

where (condition )

using order by: to sort the record.

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

> select \* from student order by marks desc;

operators:

========

1. Aritmatic operators: (\*,/, + , -, %)

Note: mostly arithmatic operator r used after select statement (90%)

and all other type of remaining operators r used in where clause only

2. relational operators (= > < <= >= [!= or <>])

3.logical operator (AND OR NOT)

4.Special operator(IN, LIKE,..)

ex: Arithmatic operator:

> select name,marks, marks+100 from student;

>select sname,marks, marks+100 Gracemarks from student;

Note: this temparory name of a column we can not use inside where clause.

ex:

> select name,marks from student where roll != 10;

using distinct:-- getting unique data

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

>select distinct marks from student;

special operators:

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

IN .... NOT IN

BETWEEN ----> NOT BETWEEN

IS NULL ----> IS NOT NULL

LIKE ----> NOT LIKE

> select \* from student where marks IS NULL;

>select \* from student where marks BETWEEN 500 AND 800;

or

>select \* from student where marks>=500 AND marks <=800;

LIKE ---> NOT LIKE

--it is used to retrive the data based on charecter pattern

1. % -- it represents string or group of charecters.

2. \_ -- it represents a single charecter.

ex:

select \* from student where name LIKE 'r%'; : name should start with r

ex: in name r should be any charecter

>select \* from student where name LIKE '%r%';

Day13

Constraints:

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

--constraints are created on the columns.

--it prevents invalid data entry into our tables.

1. not null

2.unique

3.Primary key

4. Foreign key

5.check : mysql does not support this check constraint.

Note: some contraints we can apply at column level and some constraints we can apply at table level.

column level : where we define the column

not null

unique

Primary key

table level : after defining all the columns

composit key (multi-column primary key)

foreign key

1.not null:

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

--null value is not allowed.

2.unique:

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

--here duplicate values are not allowed.

--here we can insert null values, multiple time.

Note:--whenever we define a unique then automatically DB engine will create an index on those column.

--so searching based on unique column will become fast.

3.Primary key:

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

--here also DB engine create index of that column.

--value can not be duplicate

--null is also not allowed.

\*\*\*another diff with PK and unique:- in one table we can have multiple unique constraints but in one table we can have only one PK.

--if we want to apply PK on multiple column then it will become composit key.

\*\*\*\*Note: with the help of PK column we can uniquely identify one record of a table.

create table student

(

roll int primary key,

name varchar(12) unique not null,

address varchar(12) unique not null,

marks int not null

);

teacher( tname, subject, age, phone, email )

create table teacher

(

tname varchar(12),

phone varchar(10),

email varchar(18),

age int,

subject varchar(12),

Primary key(tname,phone)

);

here tname and phone will become a composit key, this combination can not be duplicate.

Foreign key:

==========

--with the help of FK we inforce the refrential integrity.

--with the help of FK we can establish relationship bt 2 tables.

--second table FK must refer to first table PK.

--PK related FK column must belongs to the same datatype but colmun name can be diff.

--FK can accept the duplicate and null value also.

create table dept

(

did int primary key,

dname varchar(12),

location varchar(12)

);

create table emp

(

eid int primary key,

ename varchar(12),

address varchar(12),

salary int,

deptid int,

foreign key (deptid) references dept(did)

);

--with the help of FK we establish parent and child relationship among tables.

here dept table will act as parent table

and emp table will act as a child table

--the table which contains the FK column will be considered as child table.

Note: whenever we try to establish a relationship using FK then DB violates following

2 rules:

1. deletion or updation in parent table (even we can not drop the parent tables also)

2. insertion in child table.(we are not allowed any other value which is not there in parent table (PK))

--to overcome this updation and deletion problem we should use

on delete cascade

or

on delete set null

simillarly for update also.

on update cascade

or

on update set null

--while creating the child table.

create table emp

(

eid int primary key,

ename varchar(12),

address varchar(12),

salary int,

deptid int,

foreign key (deptid) references dept(did) on delete cascade

);

create table emp2

(

eid int primary key,

ename varchar(12),

address varchar(12),

salary int,

deptid int,

foreign key (deptid) references dept(did) on delete cascade on update set null

);

composit FK: it should refer the composit PK of the parent table.

===========

create table a1

(

name varchar(12),

address varchar(12),

primary key(name, address)

);

create table b1

(

nm varchar(12),

location varchar(12),

foreign key (nm,location) references a1(name,address)

);

adding a constraint to an existing table:

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

>create table a1(id int,name varchar(12));

> alter table a1 modify id int primary key;

adding foreign key:

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

>create table b1(bid int);

>alter table b1 add foreign key(bid) references a1(id) on delete set null;

functions in mysql:

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

--it is used to solve a perticular task.

--a sql function must return a value.

--in sql we have 2 types of functions:

1.predefined functions

2.user-defined functions(it is in PL/SQL)

predefined functions:

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

--it is devided into 4 categories:

1.number functions

2.charecter functions

3.group functions or aggregrate functions

4.date functions.

1.number functions:

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

a. abs(): it returns the absolute number.

ex:

>select abs(-40) from dual; //40 //here dual is a one kind of sudo table..

b. mod(m,n) : - it returns the reminder of m/n;

ex:

>select mod(10,2) from dual;

round(m,n)

truncate(m,n)

ex:

select round(12.43482,3) from dual; // 12.435

select truncate(12.43482,3) from dual; // 12.434

ceil()

floor()

greatest() least():

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

--it will return biggest and smallest value from the list of arguments.

ex:

select greatest(10,12,8,15) from dual; // 15

Note: from a single column if we want to max and min value then we should use group functions

like max() min();

2.charecter functions:

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

1.upper()

2.lower()

3.length()

4.replace()

5.concat()

6.substr()

ex:-

> select upper(ename) from emp where eid =100;

>select substr('ratan',3,2) from dual; // ta

4. date function:

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

1. sysdate() : it will return the current date and time;

ex:

>select sysdate() from dual;

2.date\_format()

> select date\_format(sysdate(), '%d %m %Y');

3. adddate()

syn:

adddate(date, INTERVAL value unit);

DAY

HOUR

YEAR

MONTH

WEEK

group function or aggregrate function:

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

--these functions operates over several values of a single column and then result in a

single value.

1.max()

2.min()

3.avg()

4.sum()

5.count(\*) // number of all records

6.count(columnName) // number of all records excepts null value

group by clause:

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

--the main purpose of group by clause is to group the records.

--this clause is mostly used with the group functions only.

--it is used to devide the similar data items into set of logical groups.

short syn:

-----------

select col\_name from table group by col\_name;

full syn:

----------

select col\_names

from

tablename

[where condition] ----opt

group by col\_names

[having <cond>]----opt

eid | ename | address | salary | deptid |

+------+---------+---------+--------+--------+

| 100 | ram | pune | 7800 | 10 |

| 102 | dinesh | pune | 7800 | 13 |

| 103 | manoj | delhi | 8500 | 13 |

| 104 | chandan | mumabi | 8200 | 14 |

| 105 | manoj | NULL | 4500 | 14 |

| 1001 | ramesh | patna | 8800 | 14 |

--the above data is called as detailed data and after performing the group by ,we get the

summerized data which is usefull for analysis.

mysql> select sum(salary) from emp; // it will calculate the salary from whole table

>select deptid, sum(salary) from emp group by deptid; // dept wise total salary

>select deptid,max(salary), min(salary), avg(salary) from emp group by deptid;

rule:

1. group functions we can not use inside the where clause.

2.other than group function all the columns mentioned inside the select clause must be there after the group by clause otherwise (oracle db will give an error and mysql may give the unexpected result).

> select deptid,ename,max(salary) maximum, min(salary) minimum, avg(salary) from emp group by deptid,ename;

Having:

----------

after group by clause we r not allowed to use where clause in place of where clause we should we having clause after group by clause.

with where clause:-

> select deptid,sum(salary) from emp where deptid IN(13,14) group by deptid having sum(salary) > 10000;

without using where clause:

>select deptid,sum(salary) from emp group by deptid having sum(salary) > 10000;

mysql> select deptid, count(\*) from emp where deptid IN(11,12,14) group by deptid having count(\*) > 2;

Note: it is not mandatory to put only group function inside the having clause.

Day14

Joins

=====

--Join is used to receive data from multiple tables or by using joins we can combine records from multiple tables.

there r following types of joins:

1. Inner Join

2. Outer Join

Left Outer join

Right Outer Join

Full Outer Join

3.self join

4.Cross Join (cartesion product)

Note: when we try to get the data from more than one table without using joining condition, then it is called cross join, in this case every record of the first table will be mapped with every record of the second table.

--with the cross join we don't get the meaningfull data, in order to get the meaningfull data we need to use other types of joins.

INNER JOIN:

==========

--here we need to apply joining condition on the common data from both table

--if ambiguity is there in column name(both table having the same col name) then we need to use alias support.

--this inner join returns the matching record from the DB tables based on common column.

> select \* from dept INNER JOIN emp ON dept.did = emp.deptid;

Q/- give the emp details who is working in 'marketing' dept.

>select eid,ename,address,salary from dept INNER JOIN emp ON dept.did = emp.deptid AND dept.dname='marketing';

with alias support:

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

>select e.eid,e.ename,e.address,e.salary,d.dname,d.location from dept d INNER JOIN emp e ON d.did = e.deptid AND d.dname='marketing';

another syntax of INNER JOIN (without using INNER JOIN command).

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

>select e.eid,e.ename,e.address,e.salary,d.dname,d.location from dept d,emp e where d.did = e.deptid AND d.dname='marketing';

Left Outer Join:

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

--to get the unmatched records from the left table use left outer join (it shows the details of left table and null value for the right table).

mysql> select d.dname,e.ename,e.address,e.salary from dept d LEFT JOIN emp e ON d.did=e.deptid;

Right Outer Join:

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

--to get the unmatched records from the right table use right outer join (it shows the details of right table and null value for the left table).

mysql> select d.did,d.dname,e.ename,e.address,e.salary from dept d RIGHT OUTER JOIN emp e ON d.did=e.deptid;

full outer join:

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

--it is a combination of LEFT outer join and RIGHT outer join.

--it display the null values both side for all the unmatched records.

Note: Full Outer join is not supported by the mysql DB.

--in order to use full outer join in mysql, then we should use union of left join and right join.

select d.did,d.dname,e.ename,e.address from dept d LEFT JOIN emp e ON d.did = e.deptid UNION

select d.did,d.dname,e.ename,e.address from dept d RIGHT JOIN emp e ON d.did = e.deptid;

Self Join:

========

--here we use joining a table to itself.

--here joining condition col must belongs to same datatype.

Note:- if we want to compare two table same col value then we use INNER JOIN whereas if we want to compare 2 diff col values within a single table then we must use self join.

\*\*\*whenever a table contains hirarical data then only we allow to use self join.

ex:

emp ----> manager

student ---> monitor

when we use self join, we must take the support of alias.

create table emp5

(

eid int primary key,

ename varchar(12),

salary int,

mgr int

);

mysql> insert into emp5 values(100,'Ram',7800,null); // RAM does not have any manager

mysql> insert into emp5 values(110,'Ravi',7200,100); // Ravi manager is RAM

mysql> insert into emp5 values(112,'amit',7500,100); // amit manager is RAM

mysql> insert into emp5 values(114,'sunil',7000,110); // sunil manager is RAVI

Q/- display the emp name and their manager name.

--here we need to use SELF JOIN

> select e1.ename EMPLOYEE, e2.ename MANAGER from emp5 e1,emp5 e2 where e1.mgr = e2.eid;

subqueries:

===========

--a query inside another query is called subquery or nested query.

--sub queries r used to retrieve the data from single or multiple tables based on more than one step process.

--here outer query is called parent query and inner query is called child query.

--child query will execute first then only parent query will be executed.

Child query :- it provides values/data to the parent query.

Parent query : it recieves the values/data from the child query.

--in child query we can not use order by clause, but parent query can use.

--group by clause can be used in both queries.

subqueries we can categories into following categories:

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

1.single row and single col SQ(scalar value SQ)

2.multiple row single col SQ

3.multiple col SQ

1.single row and single col SQ(scalar value SQ):

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

--here child query will return only a single value.

--here mostly same col name which is in the where clause of the parent query , will be there inside the select clause of child query.

Q/- WAQ to display emp details who is working in marketing dept.

using JOIN:-

>select eid,ename,address from dept INNER JOIN emp ON dept.did = emp.deptid AND dname='marketing';

using SQ:-

>select eid,ename,address from emp where deptid = (select did from dept where dname = 'marketing');

Q/- WAQ to display emp details who is working with suresh;

> select \* from emp where deptid = (select deptid from emp where ename='chandan');

Q/- WAQ to display emp details who are getting more sal than avg sal from emp table.

> select \* from emp where salary > (select avg(salary) from emp);

Q/ WAQ to display second higest salary employee details.

first highest salary emp:

> select \* from emp where salary = (select max(salary) from emp);

second highest salary emp:-

>select \* from emp where salary = (select max(salary) from emp where salary < (select max(salary) from emp));

Q/- WAQ to display details of emp who is working under RAVI;

select \* from emp5 where mgr = (select eid from emp5 where ename='ravi');

2.Multiple row,single col SQ:

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

--in multi-row, single col SQ, child query will return multiple rows and single col to the parent query.

--in this case in parent query we should use one of following operators:

IN

ANY

ALL

ex:

>select \* from emp where salary IN (select salary from emp where eid > 100);

IN : it check equal to any number in the list (using OR)

ANY : it compare any value in the SQ retured list

All : it compares all values in the SQ returned list

ex: --

salary > any(----); here it checks salary should be greater than any of 4 values in the list

salary > all(----); here it checks salary should be greater than all of 4 values in the list

< any(10,20,30,40)

ex:

< any(): less than any :- less than maximum

> any(): greater than any : - greater than equal to minimum

= any(): equal to any :- it is equal to IN operator.

5 < all(10,20,30,40)

< all(): less than all :- less than minimum

> all(): greater than all : - greater than maximum

= all(): equal to all :- it is meaningless (becoz one value can not be equal to 3 or 4 value).

ex:

>select \* from emp where salary = ANY(select salary from emp where eid > 1005);

Q/- WAQ to display the emp who is getting max salary in each dept ?

>select \* from emp where salary IN(select max(salary) from emp group by deptid);

3.multicolumn subquery:

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

--if we try to compare multiple col values of the child query with the multiple col values of the parent query then we use this type of SQ.

syn:

select \* from tab\_name where (col1,col2,....) IN (select col1, col2,.... from table where condition)

Q/- WAQ to display the emp whose salary and did matches with the salary and did of a RAM.

>select \* from emp where (salary,deptid) IN (select salary,deptid from emp where ename='suresh');

SQ in DML:

==========

SQ in insert :

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

>create table x1(id int, name varchar(12));

> insert into x1 (select eid,ename from emp);

>insert into x1 values(500, (select ename from emp where eid=1000));

SQ in update:

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

here SQ is allowed inside where clause or Set clause.

ex:

> update x1 set name= 'ramesh' where id=(select eid from emp where ename='ram');

> update x1 set name= (select ename from emp where eid=100) where id=(select eid from emp where ename='ram');

SQ in delete:

> delete from x1 where id = (select eid from emp where ename='ram');

Autoincrement in MYSQL:

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

--this is for auto generate the ID field.

--mysql supports the auto\_increment where as Oracle DB use sequence

concept to generate the ID field automatically.

create table student2

(

roll int primary key auto\_increment,

name varchar(12),

marks int

);

>alter table student2 auto\_increment=1001;

Limit:

--------

Q/ WAQ to get 4 highest salary paid employee:

select \* from emp order by salary desc LIMIT 4;

getting the record from 3 to 6 row

day13

relationship among tables:

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

At table level we have 3 type of relationships:

1. One to One (Person -- DL , Person ---AadharCard)

2. One to many (Father--child, Teacher---Student, Dept---Emp, School---Student)

3.Many to Many(Student--Course, Book--Author, Movies---Actors)

OTO:

=====

create table Person

(

pid int primary key,

pname varchar(12),

address varchar(12),

mobile varchar(10),

email varchar(15)

);

create table DL

(

dlid int primary key,

issueDate date,

expDate date,

rto varchar(12),

personId int unique ,

foreign key (personId) references Person(pid)

);

OTM:

=====

Dept(did, dname, location);

Emp(eid, ename, address, salary, deptId ---FK refer dept(did));

MTM:

=====

Student(roll, sname, address, mobile)

Course(cid , cname, fee, duration)

Note: Whenever we have a MTM relationship we need to take the help of 3rd linking table

student\_course(roll, cid);

create table student(

roll int primary key,

sname varchar(12),

address varchar(12),

mobile varchar(10)

);

create table course(

cid int primary key,

cname varchar(12),

fee int,

duration varchar(12)

);

create table student\_course

(

roll int,

cid int,

foreign key (roll) references student(roll),

foreign key (cid) references course(cid)

);

mysql> select \* from student;

+------+---------+---------+----------+

| roll | sname | address | mobile |

+------+---------+---------+----------+

| 1 | ram | delhi | 7878782 |

| 2 | ravi | delhi | 8878782 |

| 3 | chandan | hyd | 77228782 |

| 4 | venkat | chennai | 67228782 |

+------+---------+---------+----------+

4 rows in set (0.05 sec)

mysql> select \* from course;

+------+-----------+------+----------+

| cid | cname | fee | duration |

+------+-----------+------+----------+

| 1000 | Java | 8500 | 45-days |

| 1001 | Spring | 9500 | 55-days |

| 1003 | SQL | 6500 | 25-days |

| 1004 | Hibernate | 7500 | 35-days |

+------+-----------+------+----------+

//getting the Student details who enrolled in Java.

>select s.roll, s.sname, s.address, s.mobile, c.cname, c.fee, c.duration

from student s INNER JOIN course c INNER JOIN student\_course sc

ON

s.roll = sc.roll AND c.cid=sc.cid AND c.cname='Java';

or

>select s.roll, s.sname, s.address, s.mobile, c.cname, c.fee, c.duration

from student s, course c, student\_course sc

where

s.roll = sc.roll AND c.cid=sc.cid AND c.cname='Java';

select .....

from ...

ON (where)

[group by, having, order by, limit]

DBA : DDL :-- ER diagram(Entity relationship diagram)

Java Developer : is allowed to perform CRUD operation (DML + DRL)

Create a record (insert)

Retrieve a record (select)

Update a record (update)

Delete a record (delete)

JDBC:

======

Java Database connectivity.

--java mostly used in industry to develop Business application.

--common and general things required in a business application.

1. maintaining the business data permanently in a secure and easily retrival manner.

2. processing the data according to the business rule.

3. presenting the data to the user in user understandable format.

--we store the business data for a typicall business application inside the RDBMS s/w.

nosql (mono db ---> json data )

Q/ Why Java and DB communication is required ?

A/- DBMS are excellent in data storege in secure and easily retrival manner but it is very poor in processing and presenting the data in user-understandable format.

--java is excellent in processing the data and presenting the data but java is very poor in storing the data(in java we can store the data using Serialization processing).

--so inorder to develop a powerfull business application we need to communicate our java application with the DB server.

Q/- How Java-DB communication is possible ?

A/-

--Java App can do any task only through the method calls and objects, these method calls and objects are not directly understandable to the DB s/w.

--and the DB can understand only sql, Java compiler does not accept the sql syntax directly.

--inspite of heteregenious platform Java-DB communication is possible through the JDBC.

translator

chinese ---------------- Russian

Jdbc driver s/w

Java ------------------- DB

--JDBC is a technology that enables any kind of java application to communicate with any kind of DB s/w in a standard manner.

JDBC technology is given by sun-microsystem

JDBC technology comes in the form of a specification (it is documentation which describes rules and guidelines to develop a perticular s/w "Jdbc driver s/w").

--Jdbc driver s/w is the implementation of the Jdbc specification , which will act as a translator s/w to communicate our java app with the DB s/w.

--JDBC specification will be implemented by the DB verndor or any 3rd party vendor also and develop the "Jdbc driver s/w"

--each DB s/w have their own JDBC driver s/w, this driver s/w comes in the form of a .jar file.

Mysql : mysql-connector.jar

Oracle : ojdbc6.jar

Postgres : postgress.jar

--Java developer need to get/download the jdbc driver related jar file from the internet, and in order to work with that jar file, we need to set that jar file inside the classpath of our application.

Java application {

classes (method)

interface

}

---compile all the .java files ------> .class files -----> zip all the .class --inside a .jar file.

ratanapp{ A.java, B.java, Demo.java}---> make them a1.jar file.

class X { // here we need to set the a1.jar file in the classpath of our application

main(){

A a1 = new A();

}

}

--in order to communicate java application and DB server , Java developer need the "jdbc driver s/w" and

"jdbc api" to perform the DB operation from the Java application.

--jdbc api comes in the form of 2 packages:

1.java.sql package

2. javax.sql package

JDBC client:

===========

--In java-DB communication , our Java application will act as a client becoz Java application needs the services of the DB server. So our Java application is a JDBC client.

Responsibility of the JDBC client:

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

1. Requesting the DB connection.

2. Submitting the appropriate SQL statement to the DB server in the form of String.

3.Processing the result given by the DB server.

4.Dealing with exceptions if any.

5.Managing the transactions whenever it is required.

6.closing the connection once done with the DB operation.

JDBC driver :

==========

--it is a translation s/w written in Java according to the JDBC specificaiton (it comes in the form of Jar file.)

Responsibility of JDBC driver s/w :

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

1. establishing the Connection.

2. receiving the JDBC method calls (java method call ) and translate them into DBMS understandable format.(SQL) and forward them to the DB s/w.

3.Translating the DB s/w given result into the Java format (Java objects) and returns that object toe hte JDBC client.

step of connect our java application with the DB server (mysql):

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

1. download/get the jdbc driver related jar file (mysql-connector.jar) and set that jar file inside the classpath of our application.

2.Load the Driver related main class into the memory.

3. prepare the Connection string.

4. establish the connection.

5.after performing the CRUD operation close the connection.

Setting the jar file in the classpath of our application:

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

rightclick on the project----> build path ----> configure build path---->libraries---->Classpath---->Add External jars--->select the downloaded jar file ----> apply and close.

Demo.java:

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

package jdbcproject1;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class Demo {

public static void main(String[] args) {

try {

Class.forName("com.mysql.cj.jdbc.Driver");

} catch (ClassNotFoundException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

String url = "jdbc:mysql://localhost:3306/web19sb101db";

try {

Connection conn= DriverManager.getConnection(url, "root", "root");

if(conn != null)

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

}catch(SQLException e) {

e.printStackTrace();

}

}

}