Day 2

01-03-2022

Object declaration

In Java We can take the value through keyboards lot of ways

1. Scanner class.

Scanner is a pre-defined class part of util package. Package is a collection classes and interfaces.

Syntax to create the Scanner class object.

Scanner obj = new Scanner(System.in);

Day 3

02-03-2022

OOPS Concept

object : any real word entity.

Properties or state ---🡪 have

Person

Behaviour ---🡪do/does

Place

Bank

Animal

Car

Employee

Customer

class : blue print of object or template of object or user-defined data type which help describe the object.

method or function : it is use to write set of instruction to perform a specific task.

Syntax

returnType methodName(parameterList) {

}

void info() { void means to return type.

}

int getNumber() {

coding……

return 100;

}

int add(int a, int b) {

}

Syntax to create the class object.

ClassName refereceName = new ClassName();

Types of variable or fields.

1. Instance variable:
2. The variable which declared inside a class but outside a method is known as instance variable.
3. It hold default value with respective their data types : int family 🡪0, float 🡪 0.0, char 🡪 space, String null, and boolean false.
4. Instance variable we can use directly inside a all methods but method must be part of same class and it must be non static.
5. Local variable
   1. The variable which declared inside a method is known as local variable.
   2. It hold doesn’t hold default value.
   3. Scope within that method where it declared.
6. Static variable

Constructor :

Constructor is special method which help to create the object.

Points

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

If we write any constructor by default empty constructor is present.

If we want we can write more than one constructor.

Encapsulation : Binding or wrapping data and code in a single unit is known as Encapsulation.

Class

**JavaBean class or POJO class(Plain Old Java Object)**

For all variable must be private and for each variable we have to provide setter and getter methods.

Setter to set the value and getter to get the value.

When we display the reference in println internally it will call toString() method of object class.

That method it return the output as [packageName.className@coede](mailto:packageName.className@coede)

So if you want meaningful output we have to override toString() method in user defined class.

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

class A {

}

class B extends A {

}

Types of inheritance

1. Single inheritance

Class A { }

Class B extends A{}

1. Multilevel inheritance

Class A { }

Class B extends A { }

Class C extends B{ }

Class D extends D{ }

1. Hierarchical inheritance

Class A { }

Class B extends A{ }

Class C extends A{ }

1. Multiple inheritance

Class A { }

Class B { }

Class C extends A,B{ } Wrong in Java. Java doesn’t support this type of inheritance.

It support using interface.

OOPs relationship

1. Manager/Developer Is a relationship Employee
2. Has a relationship

abstract class Employee {

id,name,salary

}

class Manager extends Employee {

numberOfEmp;

Address add = new Address();

}

class Developer extends Employee{

techName;

}

class ProjectManager extends Manager{

}

class Address {

city,state etc.

}

Has a relationship

Association

class A {

B obj = new B(); 0, 1 or many

}

class B {

A obj1 = new A(); 0, 1 or many

}

Aggregation : but is known as weak association.

class Manager {

Address ladd = new Address();

Address padd = new Address();

}

class Address {

}

Composition it is known as strong association : composition.

class Student {

StudentHistory sh = new StudentHistory();

}

class StudentHistory {

}

Polymorphism

One name many forms.

Compile time polymorphism or static binding or early binding

Method overloading

Run time polymorphism or late binding or dynamic binding

Method overriding

abstract keyword

abstract method

abstract class

Inheritance Example

**package** com;

**class** Bike {

**void** speed() {

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

}

}

**class** Honda **extends** Bike {

**void** color() {

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

}

}

**class** Pulsar **extends** Bike {

**void** color() {

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

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

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

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

}

}

Method overriding

**package** com;

**class** Bike {

**void** speed() {

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

}

}

**class** Honda **extends** Bike {

**void** color() {

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

}

}

**class** Pulsar **extends** Bike {

**void** speed() {

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

}

**void** color() {

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

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

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

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

}

}

**abstract class example**

**package** com;

**abstract** **class** Bike {

**abstract** **void** speed();

**void** mailage() {

System.***out***.println("50km/lt");

}

}

**abstract** **class** Honda **extends** Bike {

**void** color() {

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

}

}

**class** Pulsar **extends** Bike {

**void** speed() {

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

}

**void** color() {

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

}

}

**public** **class** InheritanceDemoWithAbstrat {

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

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

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

}

}

Final keyword

final variable : if variable is final we can’t change the value of the variable.

final int A=10;

A=20; Error

final method : if method is final we can’t override that method.

final class : if class is final we can’t extends that class.

static keyword.

interface

class extends only one class

interface extends more than one interface

class implements more than one interface

interface can’t extends or implements class

interface is known as 100% pure abstract class.

interface Abc {

fields ; public static final int A=10;

methods public abstract void dis1();

}

interface Mno {

int B=20;

void dis2();

}

interface Xyz extends Abc,Mno{

int C=30;

void dis3();

}

class Demo implements Abc,Mno {

public void dis1() { }

public void dis2() { }

}

08-03-2022

**Exception Handling**

**Java**

**Compile time error run time error**

**Syntax error**

**Run time error**

**Error Exception**

**Both are pre-defined classes part of lang package.**

**The error which generate at run time which we can’t handle it .**

**JVC crash or software or hardware issue.**

**It is a type of run time error which we can handle it.**

**Divided by zero.**

**Exception pre-defined class**

**Checked exception Unchecked exception**

**RuntimeException**

**SQLException ArithmeticException**

**IOException NumberFormatException**

**ClassNotFoundException**

5

try

catch

finally

throw

throws

Try and catch example

Custom Exception

throw vs throws

try and catch

throw keyword is use to raise or generate pre-defined or user-defined exception base upon the conditions.

Syntax

throw new Exception ()

or

throw new ExceptionSubClass();

throws keyword is use to throw the exception to caller method.

void display() throws Exception, ExceptionSubClass {

}

Collection Framework

int a=10;

int abc[];

class Employee {

id,name,salary

}

Employee emp = new Employee();

emp.id=100;

emp.name=”Ravi”;

emp.salary = 12000;

array object

Employee employees[]=new Employee[10];

Collection Framework :

Collection --🡪 interface

Set, List, Queue and Map -🡪 interfaces

Set, List, Queue internally extends Collection but Map doesn’t extends Collection.

Set : Doesn’t allow duplicate. Set classes can be order, unorder and sorted.

HashSet, LinkedHashSet, TreeSet

List : allow duplicate and it maintain the order.

Stack, ArrayList and LinkedList

Queue : First in First Out

PriortyQueue

Map : help to store information in the form of key value pairs key is unique value may be duplicate.

HashMap, LinkedHashMap, TreeMap and Hashtable

Iterator

Iterable

HashMap : it is a type of Map API. It allow to store information in key value pairs.

Key must be unique.

HashSet : it is a type of Set API. It allow to store only value. Value must unique

for(Map.Entry entry: hm.entrySet())…..is this same or different?

Generics Overview

How to access elements from nested HashMap

Heap Memory

Map

HashMap

Generic wild card

Multithreading

Program : test of instruction to perform a specific task.

Process : program in execution or time taken to execute the code.

Processor : responsible to execute the code.

Thread : small execution of a code within a process.

Multi tasking

1. Process base
2. Thread base

We can create thread using two ways

1. Extends Thread
2. Implements Runnable interface.

Concurrent API

package

git

graded project

multithreading.

Synchronization : it is a concept which help to lock or block the thread. It allow only one thread to use all resources at time. To achieve synchronization we have to use synchronized keyword with method or we can use synchronized block inside method.

wait(), notify() and notifyAll()

wait : this method is use to suspend the thread.

Notify : this method is use to resume the thread.

1. More than one thread created in same memory
2. The method must be synchronized
3. These methods are part of Objet class.

deadlock

volatile

nextLine()

week 3 contents.

Lambda Java 8 Features

IO package.

Io package (input and output operation).

Stream : flow of data or it is a abstraction between source and destination.

System.in

System.out

System.err

System.out.println(“Welcome”);

Scanner obj = new Scanner(System.in);

System is a pre-defined class part of lang package. Out is reference of PrintStream class and it is a static reference.

PrintStream ps = System.out;

ps.println(“Welcome to Java”);

System.out.println(“Welcome to Java.”);

PrintStream is a pre-defined class always refer to standard output device ie console.

System.in is consider as Standard input device reference ie keyboard.

InputStream : standard output device ie keyboard.

Scanner sc = new Scanner(System.in);

Stream

byte char

1 byte 2 byte

Input Output input Output

InputStream OutputStream Reader Writer

DataInputStream DataOutputStream InputStreamReader OutputStreamWriter

FileInputStream FileOutputStream FileReader FileWriter

BufferedInputStream BufferedOutputStream BufferedReader BufferedWriter

ObjectInputStream ObjectOutputStream PrintWriter

PrintStream

Object Serialization : storing the object into external file or converting into byte format is known as object serialization.

Object

Property : variable store only property

Behaviour : methods not store

Identity : reference name not store

Java 8 Features

From Java8 onwards interface can contains method with body. But method must be default or static.

**package** com;

**interface** Abc {

**void** dis1(); // by default public and abstract consider

**public** **default** **void** dis2() {

System.***out***.println("dis2 is a Abc interface default method");

}

**public** **default** **void** dis3() {

System.***out***.println("dis3 is a Abc interface default method");

}

**public** **static** **void** dis4() {

System.***out***.println("dis4 is a Abc interface static method");

}

}

**class** Xyz **implements** Abc {

@Override

**public** **void** dis1() {

System.***out***.println("For Abc interface dis1() method body providedd by Xyz class");

}

@Override

**public** **void** dis2() {

System.***out***.println("Xyz class overrided dis2() default method");

}

//@Override

**public** **static** **void** dis4() {

System.***out***.println("It is static Xyz own method");

}

}

**public** **class** Java8InterfaceExample {

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

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

Xyz obj = **new** Xyz();

obj.dis1(); // overrided method

obj.dis2(); // overrided but optional

obj.dis3(); // default method

obj.*dis4*(); // calling Xyz class methodd

Abc.*dis4*(); // calling interface method

}

}

Functional interface :the interface which contains only one abstract method is known as functional interface. It can contains more than default as well as static but only one abstract method is known as functional interface.

Marker interface : The interface contains zero method or no method is known as marker interface.

Example : Serializable.

Inner classes : class within another class is known as inner class.

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

**package** com;

**class** Outer {

**int** a;

**void** dis1() {

System.***out***.println("Dis1 is a outer class method");

}

**class** Inner1 {

**void** dis2() {

System.***out***.println("Dis2 is a inner non static class method");

}

}

**static** **class** Inner2 {

**void** dis3() {

System.***out***.println("Dis3 is a inner static class method");

}

}

}

**public** **class** InnerClassExample {

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

Outer out = **new** Outer();

out.dis1();

// 1st approach

//Inner1 obj = new Inner1();

Outer.Inner1 in1 = out.**new** Inner1();

in1.dis2();

//2nd approach

Outer.Inner1 in2 = **new** Outer().**new** Inner1();

in2.dis2();

Outer.Inner2 in3 = **new** Outer.Inner2();

in3.dis3();

}

}

Lambda Expression : It is as Greek word. Using Lambda expression we can do functional programming in java.

Lambda expression is known as anonymous function or methods.

Lambda expression we can use with functional interface.

**package** com;

**interface** A {

**void** dis1();

//void dis2();

}

**class** B **implements** A {

**public** **void** dis1() {

System.***out***.println("B class provided body for dis1() method");

}

}

**public** **class** AnonymousInnerClassExample {

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

//1st approach

A obj1 = **new** B();

obj1.dis1();

//2nd appoach

A obj2 = **new** A() {

**public** **void** dis1() {

System.***out***.println("A interface method override by anonymous inner class - Logic 1 ");

}

};

obj2.dis1();

A obj3 = **new** A() {

**public** **void** dis1() {

System.***out***.println("A interface method override by anonymous inner class - Logic 2");

}

};

obj3.dis1();

//3rd approach

A obj4 = ()->System.***out***.println("This is lambda expression");

obj4.dis1();

}

}

Lambda Expression :

Lambda expression it return the value without return keyword.

Stream API

Pre-defined functional interfaces.

Stream API is a part of functional package and it is sub package of util package.

Collection Framework (Data Structure). If collection framework hold huge data of type integer,

Float, double, or user defined class object. if we want to apply any business logic first we have to load all data one by one using loop or iterator and then we have apply business rules.

Stream is use to load the data on demand. Stream doesn’t hold the data permanently.

Source data stream

Array

Or stream -🡪 IO1-🡪IO2-🡪O3---🡪 Terminal operator.

Collection

Intermediate operator return type is stream itself.

Terminal operator return type is non stream means void or any primitive type.

Function

apply() : Takes T is a parameter and return R value.

Predicate

test() : Take T is a parameter and return boolean value

Consumer

accept() : It take T parameter but no return type.

Supplier

get() : no parameter but return T value.

**16-03-2022**

Git

Version Control system tool : version control system tool is use to record the changes done in a project.

Local version control system : RCS : Revision control system

Centralized version control system : SVN all client machine connect to server machine once they do coding they push this code to remote repository (network folder).

Distributed version control system : local repository as well as remote repository.

Git is a open source Distributed version control system.

Which is developed using unix.

Open the git bash inside a folder

git --version

git init This command is use to make the local folder as a local repository.

git status : This command is use to check the current status of local repository

git add filename : This command is use to add the file from local file system to staging area.

Staging area : it is area provided by git where all files and folder store before commit.

git branch : it is like a pointer which hold more than one commit details.

Git by default one branch created it may be master or main.

The command to check the branch

git branch

syntax to create the user-defined branch

git branch branchname

to switch to branch

git checkout branchname

merge code from one branch to another branch

git merge branchName ( this command to merge branch code to current branch)

to link local repository to remote repository

git remote add origin URL

Week 3

RDBMS Using My SQL

JDBC

Hibernate

To download the existing remote repository in local machine you have run the command as

git clone URL

if you want to get new updated in existing repository you have to run the command as

git pull

MySQL

File System we done using IO package

1. Data redundancy (duplicate records).
2. Data consistency (format of the file or delimiter between two records).
3. CRUD Operation (Create, Read, Update and Delete).
4. Security

Data : raw fact

Information : meaningful data or processed data.

Database : storing the data in table format.

DBMS : Database Management system : it is software which help to store the data in table format.

Excel :

TrainerStudentDetails

TId TName Tech SId SName Age

1 Raj Java 100 Seeta 21

1 Raj Java 101 Meeta 22

1 Raj Java 102 Reeta 23

Dr. EF codd’s Rules 0 to 11 rules

MySQL

Oracle

Db2

SQL Server 2021

RDBMS

SQL : Structure Query Language

5 types

DRL or DQL : select clause (Data query language

DDL :Data definitional language (create, alter, drop, rename, truncate)

DML : Data manipulation language : insert, delete, and update

TCL : transactional control language (commit, rollback and save point)

DCL : granted and revoke (Data control language).

Show databases

Create database databaseName

use databaseName

show tables;

DDL

create table tableName(columnName datatype,columnName datatype);

desc sample

DML :

Insert

insert into sample values(1,'Ravi');

Delete

delete from tableName; it delete all records.

delete from sample where srno=1

Delete from sample where name =’Ravi’;

Update

update tableName set columnName = value

update sample set name = ‘Raj’;

update sample set name='Ajay Kumar' where srno=3;

DRL

select \* from tableName

Keys

Candidate keys

FK

PK

Cid,CName,Age,PhNumber,Accno,Amont,TypeOfAccount,PId,PName,Price,MgrId,Name etc

**Keys** : A key is an attribute (column or field) or combination of more than one column which is use to identify records.

Super key : an attribute or a combination of attribute that is used to identify the record uniquely.

Cid

PhNumber

AccNo

PId

MgrId

Cid,CName

Cid,PhNumber

typeOfAccount

Cid,AccNumber

Etc

Candidate key it is defined as minimal super key or irreducible super key is known as Candidate key.

Primary key : primary key is type of candidate key. Single table we can create only one column as pk. Pk doesn’t allow duplicate as well as null value

Unique key : in single table we can create more than one column as pk. It allow null value but doesn’t allow duplicate.

FK : FK is use to refer to pk of same table or different table. Fk allow only do value which present in pk it allow null value.

reate table trainer(tid int, tname varchar(10) not null, tech varchar(10) unique not null,

primary key(tid));

create table student(sid int, sname varchar(10) not null, age int, tsid int,

primary key(sid),

foreign key(tsid) references trainer(tid));

TCL :

Sub Query

DCL

21-03-2022

JDBC : Java Database Connectivity

Maven : Maven is build tool responsible to download the dependencies.

DAO : Data Access Object : Pure database logic.

Limitation of JDBC

1. Using JDBC we can’t store as well as can’t retrieve object form database means we have to convert java object into sql and vice-versa.
2. Jdbc use sql language. SQL is database dependent language.
3. Jdbc always throw checked exception. So we have to handle it mandatory.
4. JDBC doesn’t support is a and has a relationship.

ORM : Object Relation Mapping

JavaBean or POJO or Entity Relation

class Employee { Employee

id,name,salary ID,Name,Salary

}

Mapping

Employee – Employee

Id -🡪ID PK  
 name🡪Name

Salary 🡪SALARY

Using xml file or annotation

ORM : Hibernate, JPA, iBaties etc.

JPA is a technologies part of JEE (Java Enterprise Edition).

Hibernate is framework.

JPA is known as specification and Hibernate is implementation.

Configuration details (database details

Drivername,url, username, password and dialects class)

Configuration details we can provide using xml or properties or java classes.

22-03-2022

SQL : Structured Query language : SQL is database dependent. It retrieve sql query

Select \* from employee (employee table sql is not a case sensitive)

Select name from employee;

Select id from employee;

Select id,name, from employee

Select \* from employee where salary > 12000;

HQL : Hibernate Query Language : HQL is database independent. It retrieve entity class object.

select emp from Employee emp (Employee is entity class name and it is a case sensitive).

Select emp.id from Employee emp

Select emp.name from Employee emp;

Select emp.name,emp.salary from Employee emp

Select emp from Employee emp where emp.salary > 12000

23-03-2022

Hibernate Relationship

4 types one : PK many : FK

One - to – one Person Passport

Pk fk

Pk pk (shared primary key)

One – to – many Trainer Student

Project Employees

Many – to –one Students Section

Employees Departments

Many – to – many Employees SkillSet

Employees

EmpId EmpName

1. Raj
2. Ravi
3. Ramesh

SkillSet

SkillSet SkillName

100 Java

101 Python

102 Angular

Employee\_SkillSet

1. 100
2. 101
3. 102
4. 100
5. 102

Using Hibernate we are store, delete and update using method but retrieve the records

The we are depending upon HQL.

Criteria : Criteria is hibernate API we can do select operation on entity using methods.

Select name from employee sql

Select emp.name from Employee emp HQL

Oracle Sequence

create sequence myseq;

insert into employee values(myseq.nextval,'Ajay',12000);

25-03-2022

No SQL Database

Limitation of RDBMS

Schema base database

Table 🡪Employee

Number of columns and data type for that columns

Employee

Id Name Salary age City

1 Raj 12000 null null

2 Ravi 14000 null null

3 Ajay 16000 21 null

4 Mahesh 18000 null Bangalore

Trainer

PK

TId TName Tech

1 Raj Java

2 Ravi Python

Student

SID FK

SId SName Age TsId

100 Meeta 21 1

101 Reeta 22 1

No SQL

Mongo DB

Neo4j

Cassandra

HBase

Mongo DB : Mongo DB is open source document base No SQL Database which is use to store the data in the form of json.

JSON is like a Map in the form key-value pairs. Key is unique and value may be duplicate.

Java Script Object Notation.

{“id”:100,”name”:”Ravi”,age:21};

C:\Program Files\MongoDB\Server\5.0\bin

In C Drive you have to create the folder as

data --🡪 inside data folder you have to create db folder.

db ----🡪

To run the mongo db server you have to run the command as

mongod : inside a bin folder

This command to start the service

Then open another command prompt inside a bin folder and run the command as

mongo

this terminal is use to run the mongo db commands.

In mongo terminal

Cntr + L

use databasename; if database present it will switch to existing database else it will create and switch to that database.

In Mongo DB table is known as collection and row is known as document.

In Mongo DB column is as a key.

Syntax to create the collection

db.createCollection("Sample");

syntax to insert the document in collection

db.Sample.insert({name:”Ravi”})

db.Sample.insert({name:”Ram”,age:”21”});

view the document from a collection.

db.Sample.find();

db.Employee.insert({\_id:1,name:"Ravi",age:21,city:"Bangalore"});

WriteResult({ "nInserted" : 1 })

> db.Employee.insert({\_id:2,name:"Ramemsh",age:24,city:"Delhi"});

WriteResult({ "nInserted" : 1 })

> db.Employee.insert({\_id:3,name:"Ajay",age:26,city:"Mumbai"});

WriteResult({ "nInserted" : 1 })

> db.Employee.insert({\_id:4,name:"Vijay",age:25,city:"Bangalore"});

WriteResult({ "nInserted" : 1 })

> db.Employee.find();

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

{ "\_id" : 2, "name" : "Ramemsh", "age" : 24, "city" : "Delhi" }

{ "\_id" : 3, "name" : "Ajay", "age" : 26, "city" : "Mumbai" }

{ "\_id" : 4, "name" : "Vijay", "age" : 25, "city" : "Bangalore" }

Retrieve the document from a collection using index positon

> db.Employee.find()[0];

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

> db.Employee.find()[1];

{ "\_id" : 2, "name" : "Ramemsh", "age" : 24, "city" : "Delhi" }

> db.Employee.find()[1].name;

Ramemsh

> db.Employee.find()[1].age;

24

Retrieve the document using condition

db.CollectionName.find({condition},{filterTheField});

db.Employee.find({\_id:1});

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

> db.Employee.find({city:"Bangalore"});

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

{ "\_id" : 4, "name" : "Vijay", "age" : 25, "city" : "Bangalore" }

> db.Employee.find({age:{$gt:25}});

{ "\_id" : 3, "name" : "Ajay", "age" : 26, "city" : "Mumbai" }

> db.Employee.find({age:{$gt:24}});

{ "\_id" : 3, "name" : "Ajay", "age" : 26, "city" : "Mumbai" }

{ "\_id" : 4, "name" : "Vijay", "age" : 25, "city" : "Bangalore" }

> db.Employee.find({age:{$lt:24}});

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

> db.Employee.find({age:{$lte:24}});

{ "\_id" : 1, "name" : "Ravi", "age" : 21, "city" : "Bangalore" }

{ "\_id" : 2, "name" : "Ramemsh", "age" : 24, "city" : "Delhi" }

Retrieve specific fields from a documents

> db.Employee.find({},{name:1});

{ "\_id" : 1, "name" : "Ravi" }

{ "\_id" : 2, "name" : "Ramemsh" }

{ "\_id" : 3, "name" : "Ajay" }

{ "\_id" : 4, "name" : "Vijay" }

> db.Employee.find({},{name:1,\_id:0});

{ "name" : "Ravi" }

{ "name" : "Ramemsh" }

{ "name" : "Ajay" }

{ "name" : "Vijay" }

> db.Employee.find({},{name:1,\_id:0,age:1});

{ "name" : "Ravi", "age" : 21 }

{ "name" : "Ramemsh", "age" : 24 }

{ "name" : "Ajay", "age" : 26 }

{ "name" : "Vijay", "age" : 25 }

Update document using conditions

db.Employee.update({\_id:1},{$set:{age:22}}) : using \_id property

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

db.Employee.updateMany({city:"Bangalore"},{$set:{city:"Mysore"}}) : using other property

delete query

db.Employee.find();

**28-03-2022**