Core JAVA – Notes

====Complete Java===================

=============20-01-2025==============================+

===================Core Java=========================

public class Main {

public static void main(String[] args) {

System.out.println("hello world");

}

}

public class Main {

public static void main(String[] args) {

String myText = "My name is komal";

System.out.println(myText);

}

}

public class Main {

public static void main(String[] args) {

int x = 25;

int y = 30;

System.out.println(x + y);

}

}

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

Java Operators :-

Operators are used to perform operations on variable and values.

> Arithmatic

>Assignment

>Comparison

>Logical

>Bitwise

Ex:-

float first = 2.20f, second = 2.85f;

float temp = first;

first = second;

second = temp;

System.out.println(first);

System.out.println(second);

Ex:

int x = 10, y = 20;

System.out.println(x != y);

Ex:

int x = 10;

System.out.println(!(x > 5 || x > 15));

Type Casting :-

// int num1 = 8;

// double num2 = num1;

// System.out.println(num1);

// System.out.println(num2);

// double num1 = 9.45d;

// int num2 = (int) num1;

// System.out.println(num1);

// System.out.println(num2);

int max = 500;

int score = 450;

float percentage = (float) score / max \* 100.0f;

System.out.println(percentage);

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

Control flow management :-

1) Conditional statement :-

=> if-else statement :-

Ex:-

int x = -10;

if(x > 0) {

System.out.println("Positive");

}

else if (x < 0) {

System.out.println("Negative");

}

else {

System.out.println("the value is 0");

}

Ex:-

int x = 8;

if (x % 2 == 0) {

System.out.println("even");

}

else {

System.out.println("odd");

}

Ex:-

char ch = 'i';

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

System.out.println(ch + " is vowel");

}

else {

System.out.println(ch + " is consonant");

}

Ex:-

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner reader = new Scanner(System.in);

System.out.println("Enter a number : ");

int num = reader.nextInt();

if (num % 2 == 0) {

System.out.println(num + "even");

}

else {

System.out.println(num + "odd");

}

}

}

Ex:-

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a character: ");

char ch = scanner.next().charAt(0);

if ((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z')) {

System.out.println(ch + " is an alphabet.");

} else {

System.out.println(ch + " is not an alphabet.");

}

scanner.close();

}

}

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

Task :

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

double balance = 1000.00;

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("ATM Menu");

System.out.println("1. Check Balance");

System.out.println("2. Deposite");

System.out.println("3. Withdraw");

System.out.println("4. Exit");

System.out.println("Choose an option: ");

int choice = scanner.nextInt();

if (choice == 1) {

System.out.println("Your balance is : " + balance);

}else if(choice == 2) {

System.out.println("Enter a deposit amount : ");

double deposit = scanner.nextDouble();

if (deposit > 0) {

balance += deposit;

System.out.println("Deposit successful, New balance" + balance);

} else {

System.out.println("Invalid amount");

}

}else if (choice == 3) {

System.out.println("Enter withdrawl amount : ");

double withdrawl = scanner.nextDouble();

if (withdrawl > 0 && withdrawl <= balance) {

balance -= withdrawl;

System.out.println("Withdrwal successful, Remaining balance : " + balance);

}else {

System.out.println("insufficient balance or invalid amount ");

}

}else if (choice == 4) {

System.out.println("Thankyou for using the ATM, GOod bye");

break;

}else{

System.out.println("Invalid choice, Please try with the option ");

}

}

}

}

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

Switch case :-

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

double balance = 1000.00;

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("ATM Menu");

System.out.println("1. Check Balance");

System.out.println("2. Deposite");

System.out.println("3. Withdraw");

System.out.println("4. Exit");

System.out.println("Choose an option: ");

int choice = scanner.nextInt();

switch (choice) {

case 1: //check balance

System.out.println("Your balance is : " + balance);

break;

case 2: //Deposit

System.out.println("Enter deposite amount : ");

double deposit = scanner.nextDouble();

if (deposit > 0) {

balance += deposit;

System.out.println("Deposit successful, New balance" + balance);

} else {

System.out.println("Invalid amount");

}

break;

case 3: //withdraw

System.out.println("Enter withdrawl amount : ");

double withdrawl = scanner.nextDouble();

if (withdrawl > 0 && withdrawl <= balance) {

balance -= withdrawl;

System.out.println("Withdrwal successful, Remaining balance : " + balance);

}else {

System.out.println("insufficient balance or invalid amount ");

}

break;

case 4: //Exit

System.out.println("Thankyou for using the ATM, GOod bye");

scanner.close();

return; //exit the programme

default: //invalid option

System.out.println("Invalid choice, Please try with the option ");

}

}

}

}

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

Task :

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a string : ");

String input = scanner.nextLine();

//check if the input string is palindrome

if (isPalindrome(input)) {

System.out.println("The string is palindrome");

} else {

System.out.println(" not a palindrome ");

}

}

public static boolean isPalindrome(String str) {

int left = 0;

int right = str.length() - 1;

while (left < right) {

if (str.charAt(left) != str.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

}

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

Task : Find the Largest and Smallest Elements in an Array.

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

=> Repeatational statements :-

1 ) while loop :-

> int i = 0;

while ( i < 10) {

System.out.println(i);

i++;

}

> // int num = 50, sum = 0;

// int i = 1;

// while (i <= num) {

// sum += i;

// i++;

// }

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

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

> Count number of digits in an integer

> Reverse a number using loop

> calculate a power of a number

> Find the factorial of a number

> Define the factors of a number

> Make a advanced level calculator using switch case

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

Ex:-

public static void main(String[] args) {

int n = 10, firstTerm = 0, secondTerm = 1;

System.out.println("Fibonacci series till " + n + " terms");

for (int i = 1; i <= n; ++i) {

System.out.print(firstTerm + " ,");

//Logic to build the next term

int nextTerm = firstTerm + secondTerm;

firstTerm = secondTerm;

secondTerm = nextTerm;

}

}

Ex:-

System.out.println("Prime Numbers between 1 and 100");

for (int num = 2; num <= 100; num++) {

boolean isPrime = true;

for(int i = 2; i <= Math.sqrt(num); i++){

if (num % i == 0) {

isPrime = false; //Not prime if divisible

break;

}

}

if (isPrime) {

System.out.print(num + " ");

}

}

Ex:-

String text = "we have completed total 5 classes";

int vow = 0, conso = 0, digi = 0, space = 0;

text = text.toLowerCase();

for(int i = 0; i < text.length(); ++i) {

char ch = text.charAt(i);

if (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ) {

++vow;

}

else if (ch >= 'a' && ch <= 'z') {

++conso;

}

else if (ch >= '0' && ch <= '9') {

++digi;

}

else if (ch == ' ') {

++space;

}

}

System.out.println("Vowels " + vow);

System.out.println("Consonants " + conso);

System.out.println("Digits " + digi);

System.out.println("White spaces " + space);

a) do-while loop

2 ) For loop : -

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

System.out.println(i);

}

> // for(int i = 1; i <= num; ++i) {

// sum += i;

// }

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

a) for-each loop

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

//Arrays

// Find the second largest element in an array

// public class Main {

// public static int findSecondLargest(int[] arr) {

// if (arr.length < 2) {

// System.out.println("Array must have at least two elements");

// return -1;

// }

// int first = Integer.MIN\_VALUE, second = Integer.MIN\_VALUE;

// for(int num: arr) {

// if(num > first) {

// second = first;

// first = num;

// } else if (num > second && num != first) {

// second = num;

// }

// }

// return (second == Integer.MIN\_VALUE) ? -1 : second;

// }

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

// int[] arr = {10, 20, 5, 30, 52};

// System.out.println("second largest number " + findSecondLargest(arr));

// }

// }

//Rearrange Array Alernately (Max and Min Order)

//CAlculate the average of array elements

//Add the two matrics

//find the Transpose of a matrix

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

=> Merge two arrays

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

int[] arr1 = {1,2,3};

int[] arr2 = {5,6,7};

int a = arr1.length;

int b = arr2.length;

int[] result = new int[a + b];

System.arraycopy(arr1, 0, result, 0, a);

System.arraycopy(arr2, 0, result, a, b);

System.out.println(Arrays.toString(result));

}

}

Ex: -

import java.util.Arrays;

public class Main {

public static void main(String[] args) {

int[] arr1 = {1,2,3};

int[] arr2 = {5,6,7};

int length = arr1.length + arr2.length;

int[] result = new int[length];

int pos = 0;

for(int element: arr1) {

result[pos] = element;

pos++;

}

for (int element: arr2) {

result[pos] = element;

pos++;

}

System.out.println(Arrays.toString(result));

}

}

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

=> String: -

> Find the total frequency of charcter in a string

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

//class scope'

class Person {

String name; //Instance variable (class scope)

static String role = "Admin"; //Static variable (Class Scope)

void display() {

System.out.println(name + role);

}

}

public class Main {

public static void main(String[] args) {

Person p1 = new Person();

p1.name= "vaibhav";

p1.display();

}

}

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

Recursion :-

Ex:

//Optimized recursion (Tail recursion) :

//Tail recursion is a special form where the rcursive call is the last operation in the function, making easier for java to optimize

//Normal recursion

public class Main {

public static int fact(int n, int result) {

if (n == 0 || n == 1) return result;

return fact(n - 1, n \* result);

}

public static void main(String[] args) {

System.out.println("Fact : " + fact(5, 1));

}

}

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

OOPS :-

Class :- blueprint or template for creating objects, it defines the properties(variables) and behaviour(methods) that an object can have.

class Car {

String brand; //property

int speed;

void display() {

System.out.println(brand);

System.out.println(speed);

}

}

public class Main {

public static void main(String[] args) {

Car obj = new Car();

obj.brand = "Toyota";

obj.speed = 150;

obj.display();

}

}

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

key diffrence :

features class object

defination blueprint instance of the class

memory allocation No memory memory will aloc

Usage Defines pro and methods uses prop and methods

> we can create the multiple objects with one class

> Type of Attributes(variable)

1) Instance Variable : Belong to an object and have unique values for each object.

2) Class Variable(Static Variable) : Shared among all instances of the class

Static variable memory allocation in java :

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

Constructor:

1) Default

2) Parameterized

3) Copy Const

Ex:

class Car {

String brand;

int speed;

//Parameterized const

Car(String b, int s){

brand = b;

speed = s;

}

//Cop Const

Car(Car obj) {

brand = obj.brand;

speed = obj.speed;

}

void display() {

System.out.println("Brand " + brand + "Speed " + speed);

}

}

public class Main {

public static void main(String[] args) {

//Creates a new object by copying values from another object

Car car1 = new Car("Tata", 180);

Car car2 = new Car(car1);

car1.display();

car2.display();

}

}

4) Constructor Overloading :

Multiple constructors with diffrent parameter lists in the same class.

class Car {

String brand;

int speed;

//Default const

Car(){

brand = "Unknown";

speed = 0;

}

//parameterized const

Car(String b, int s) {

brand = b;

speed = s;

}

void display() {

System.out.println("Brand " + brand + "Speed " + speed);

}

}

public class Main {

public static void main(String[] args) {

//Creates a new object by copying values from another object

Car car1 = new Car();

Car car2 = new Car("Tata", 200);

car1.display();

car2.display();

}

}

4) this keyword in constructor : -

class Car {

String brand;

int speed;

//this

Car(String brand, int speed){

this.brand = brand;

this.speed = speed;

}

void display() {

System.out.println("Brand " + brand + "Speed " + speed);

}

}

public class Main {

public static void main(String[] args) {

Car car = new Car("Tata", 200);

car.display();

}

}

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

=> Modifiers :-

1) Access modifiers :

abstract class User{

abstract void isAdmin();

void display () {

System.out.println("hello");

}

}

class Emp extends User {

void isAdmin() {

System.out.println("abstract method body");

}

}

public class Main {

public static void main(String[] args) {

Emp obj = new Emp();

obj.isAdmin();

obj.display();

}

}

=> Encapsulation :-

// Abstract Class

abstract class AccountOperations{

abstract void deposit(double amount);

abstract void withdraw(double amount);

}

//Interface

interface InterestCalculable{

double calculateInterest();

}

//Parent class

class BankAccount extends AccountOperations{

private String accountNumber;

private String accountHolderName;

private double balance;

//Constructor

public BankAccount(String accountNumber, String accountHolderName, double balance){

this.accountNumber = accountNumber;

this.accountHolderName = accountHolderName;

this.balance = balance;

}

//Encapsulation (get & set)

public String getAccountNumber(){

return accountNumber;

}

public String getAccountHolderName(){

return accountHolderName;

}

public double getBalance(){

return balance;

}

public void setBalance(double balance) {

this.balance = balance;

}

@Override

public void deposit(double amount){

balance += amount;

System.out.println("Deposited : " + amount + "New balance : " + balance);

}

@Override

public void withdraw(double amount){

if (balance >= amount) {

balance -= amount;

System.out.println("Withdrawn : " + amount + "New Balance : " + balance);

}else{

System.out.println("insufficient balance!!");

}

}

}

//Sub class

class SavingsAccount extends BankAccount implements InterestCalculable{

private double InterestRate;

public SavingsAccount(String accountNumber, String accountHolderName, double balance, double InterestRate){

super(accountNumber, accountHolderName, balance);

this.InterestRate = InterestRate;

}

@Override

public double calculateInterest(){

return getBalance() \* (InterestRate / 100);

}

}

class CurrentAccount extends BankAccount{

private double overdraftLimit;

public CurrentAccount(String accountNumber, String accountHolderName, double balance, double overdraftLimit){

super(accountNumber, accountHolderName, balance);

this.overdraftLimit = overdraftLimit;

}

@Override

public void withdraw(double amount){

if (getBalance() + overdraftLimit >= amount) {

setBalance(getBalance() - amount);

System.out.println("Withdrawn : " + amount + " New Balance :" + getBalance());

}else {

System.out.println("Exceeded overdraft limit");

}

}

}

//Driver Class

public class Main{

public static void main(String[] args) {

SavingsAccount sa = new SavingsAccount("SA123", "Abhishek", 20000, 5);

CurrentAccount ca = new CurrentAccount("CA456", "Kajal", 30000, 2000);

//Deposite and withdrawal

sa.deposit(2000);

sa.withdraw(1000);

System.out.println("interest : " + sa.calculateInterest());

ca.deposit(1500);

ca.withdraw(3500);

ca.withdraw(500);

}

}

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

=> Java does not allow multiple inheritance using classes to avoid the Diamond Problem

>> Diamond problem occures in language that support multiple inheritance (like c++), where a class can inherit from two parent classes that have same method.

This create ambiguity about which method the child class should inherit.

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

=> Polymorphism :-

//Upacasting :-

class Parent {

void show(){

System.out.println("one");

}

}

class Child extends Parent {

@Override

void show() {

System.out.println("two");

}

}

public class Main {

public static void main(String[] args) {

Parent obj = new Child(); //Upcasting

obj.show();

}

}

=> Abstraction in java :-

Ex:-

abstract class Vehicle {

abstract void start(); //Abstract method(no body)

void stop() { //concrete method

System.out.println("to stop");

}

}

class Car extends Vehicle {

@Override

void start() {

System.out.println("start a car");

}

}

class Bike extends Vehicle {

@Override

void start() {

System.out.println("Bike start");

}

}

public class Main {

public static void main(String[] args) {

Vehicle car = new Car();

car.start();

car.stop();

Vehicle bike = new Bike();

bike.start();

bike.stop();

}

}

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

Collections Framework:-

//Collection framework :- they provides ready to use data structure and algorithms for storing, manipulating and retriving collection of object efficiency.

//java.util.package

//lists, sets, maps and more

//key interfaces

//1) collection : the root interface for most collections

//> List :- ArrayList, LinkedList

//ArrayList

import java.util.ArrayList;

import java.util.List;

class Main {

public static void main(String[] args) {

// List<String> list = new ArrayList<>();

// list.add("one");

// list.add("two");

// list.add("three");

// list.add("three");

// list.indexOf(list);

// System.out.println("List : " + list.contains("two"));

// list.add(1, "number");

// System.out.println("List : " + list);

// list.get(0);

// list.set(2, "new");

// System.out.println("List : " + list);

// list.remove(2);

// list.remove("number");

// System.out.println("List : " + list);

// // isEmpty()

// System.out.println("List : " + list.size());

// list.clear();

// System.out.println("List : " + list);

// System.out.println("List : " + list.isEmpty());

}

}

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

import java.util.Arrays;

import java.util.Iterator;

import java.util.List;

public class Main {

public static void main(String[] args) {

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

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

while (it.hasNext()) {

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

}

}

}

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

import java.util.\*;

// Sorting in collections

public class Main {

public static void main(String[] args) {

List<Integer> numbers = new ArrayList<>(Arrays.asList(5,2,8,1));

Collections.sort(numbers);

System.out.println(numbers);

}

}

//Task : Sorting in descending order

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

>> SubList and Conversion methods :-

List<String> list = new ArrayList<>(Arrays.asList("Java", "Python", "Javascript", "C++"));

List<String> sublist = list.subList(1, 3);

System.out.println(sublist);

String[] arr = list.toArray(new String[0]);

System.out.println(Arrays.toString(arr));

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

>> synchronizedList :-

List<Integer> syncList = Collections.synchronizedList(new ArrayList<>());

synchronized (syncList) {

syncList.add(1);

syncList.add(2);

syncList.add(3);

}

System.out.println(syncList);

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

=> Sorting Objects using comparators

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

class Student {

String name;

int age;

Student(String name, int age) {

this.name = name;

this.age = age;

}

@Override

public String toString() {

return name + "(" + age + ")";

}

}

public class Main {

public static void main(String[] args) {

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

Students.add(new Student("Neha", 25));

Students.add(new Student("Gopal", 24));

Students.add(new Student("Karam", 27));

//Sorting by Age (ascending)

Students.sort(new AgeComparator());

System.out.println("Sorted age : " + Students);

//Sorting by Name(Alphabetically)

Students.sort(new NameComparator());

System.out.println("Sorted Name : " + Students);

}

}

//Comparator for sorting age

class AgeComparator implements Comparator<Student> {

public int compare(Student s1, Student s2){

return Integer.compare(s1.age, s2.age);

}

}

//Comparator for sorting Name

class NameComparator implements Comparator<Student> {

public int compare(Student s1, Student s2){

return s1.name.compareTo(s2.name);

}

}

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

Linked List :-

LinkedList<Integer> numbers = new LinkedList<>();

//Add elements

numbers.add(30);

numbers.add(40);

numbers.add(50);

numbers.add(60);

System.out.println(numbers);

// numbers.add(2, 60);

numbers.addFirst(20);

System.out.println(numbers);

numbers.addLast(70);

System.out.println(numbers);

// remove elements

numbers.remove(5);

System.out.println(numbers);

numbers.removeFirst();

numbers.removeLast();

System.out.println(numbers);

// System.out.println(numbers.getFirst());

// System.out.println(numbers.getLast());

//looping

for (int num : numbers) {

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

}

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

Ex:-

LinkedList<String> names = new LinkedList<>();

names.add("gopal");

names.add("neha");

names.add("karam");

names.add("ram");

System.out.println(names);

names.remove("ram");

System.out.println(names);

//Sorting

Collections.sort(names);

System.out.println(names);

Collections.sort(names, Collections.reverseOrder());

System.out.println(names);

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

=> Set :-

> HashSet :

HashSet<String> set = new HashSet<>();

set.add("One");

set.add("Two");

set.add("Three");

System.out.println(set);

System.out.println(set.contains("Two"));

set.remove("Three");

System.out.println(set);

System.out.println(set.size());

Ex:-

HashSet<Integer> nums = new HashSet<>();

nums.add(20);

nums.add(30);

nums.add(40);

nums.add(40);

nums.add(null);

nums.add(50);

System.out.println(nums);

System.out.println(nums.contains(null));

//Iteration

for(Integer num:nums) {

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

}

Ex:- Convert the HashSet into a list

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

> TreeSet :-

Ex:-

TreeSet<Integer> numbers = new TreeSet<>();

numbers.add(10);

numbers.add(20);

numbers.add(30);

numbers.add(40);

System.out.println(numbers);

System.out.println(numbers.ceiling(25));

System.out.println(numbers.floor(25));

System.out.println(numbers.higher(20));

System.out.println(numbers.lower(20));

====Comparator=====

Ex:-

TreeSet<Integer> numbers = new TreeSet<>(Comparator.reverseOrder());

numbers.add(10);

numbers.add(20);

numbers.add(15);

System.out.println(numbers);

Ex: -

TreeSet<String> names = new TreeSet<>();

try {

names.add(null);

} catch(Exception e) {

System.out.println("Canot add null to treeset");

}

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

Task : Pass ArrayList as function parameter by converting into Array :

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

Queue<String> queue = new LinkedList<>();

queue.add("one");

queue.add("Two");

queue.add("Three");

System.out.println("Head : " + queue.peek());

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

System.out.println(queue);

Prirority queue :

//Elementsare sorted in s sorted in sorted order(natural or custom)

// not allow null values

//Uses a min-heap internally

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

queue.add(30);

queue.add(10);

queue.add(20);

System.out.println(queue);

//Remove elements(sorted order)

while (!queue.isEmpty()) {

System.out.println("polled : " + queue.poll());

}

//Always removes the smallest element first

Deque (Double-Ended Queue) :

//Allow insertion and deletion from both ends

//Faster Than linkedList

Deque<Integer> deque = new LinkedList<>();

deque.add(10);

deque.add(20);

deque.add(10);

deque.add(30);

System.out.println(deque);

Deque<String> deque = new ArrayDeque<>();

deque.addFirst("First");

deque.addLast("Last");

System.out.println(deque);

System.out.println("Removed First : " + deque.pollFirst());

System.out.println("Removed Last : " + deque.pollLast());

System.out.println(deque);

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

=> Map :- used store type data in key-value pair

1) HashMaps :-

>> No duplicate keys

>> Allow null (one or multiple)

>> unordered

>> Provide Fast Operations

eX:-

import java.util.HashMap;

import java.util.Map;

public class Main {

public static void main(String[] args) {

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

map.put(1, "Gopal");

map.put(2, "Gaurav");

map.put(3, "Karampal");

System.out.println(map);

System.out.println("Value for key 2 : " + map.get(2) );

System.out.println("After removal : " + map.remove(2));

System.out.println(map.containsKey(1));

System.out.println(map.containsValue("Gopal"));

System.out.println(map);

map.put(2, "neha");

System.out.println(map);

//iteration

for (Integer key : map.keySet()) {

System.out.println(key);

}

for (String value : map.values()) {

System.out.println(value);

}

for (Map.Entry<Integer, String> entry: map.entrySet()) {

System.out.println(entry.getKey() + " = " + entry.getValue());

}

//Map.Entry :- nested interface

//entrySet() :- returns a Set of Map.Entry objects, Each Map.Entry contains a key and a value.

}

}

2) TreeMaps :-

>> No duplicate

>> Sorted order will ascending

>> custom sorting can be perform using comparator

>> Allows null

Ex:-

TreeMap<Integer, String> map = new TreeMap<>();

map.put(3, "Apple");

map.put(1, "Banaras");

map.put(2, "Chemistry");

System.out.println(map);

//Get method use

//remove method use

//first and last key

System.out.println("First key " + map.firstKey());

System.out.println("Last key " + map.lastKey());

Iterator :- an interface allows traversing(itreating) the elements in a collections, it provide methods to fetch, remove and check the elements while iterating.

ArrayList<String> names = new ArrayList<>();

names.add("sorting");

names.add("apple");

names.add("ship");

Iterator<String> iterator = names.iterator();

//check if next element exists

while (iterator.hasNext()) {

String name = iterator.next();

System.out.println(name);

}

Ex: -

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

public class Main {

public static void main(String[] args) {

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

map.put(1, "Gopal");

map.put(2, "Gaurav");

map.put(3, "Karampal");

System.out.println(map);

Iterator<Map.Entry<Integer, String>> iterator = map.entrySet().iterator();

while (iterator.hasNext()) {

Map.Entry<Integer, String> entry = iterator.next();

System.out.println("key" + entry.getKey() + "Values : " + entry.getValue());

}

}

}

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

Java 8 (advanced) :

=> Lambda expression :-

// interface MathOperation {

// int operate(int a, int b);

// }

// class Addition implements MathOperation {

// public int operate(int a, int b) {

// return a + b;

// }

// }

// public class Main {

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

// MathOperation add = new Addition();

// System.out.println(add.operate(4, 3));

// }

// }

interface MathOperation {

int operate(int a, int b);

}

public class Main {

public static void main(String[] args) {

MathOperation add = (a, b) -> a + b;

System.out.println(add.operate(5, 3));

}

}

=========

>> pass multiple statement :

Ex:-

interface Greeting {

void sayHello(String name);

}

public class Main {

public static void main(String[] args) {

Greeting greet = name -> {

String msg = "Hello" + name + "!";

System.out.println(msg);

};

greet.sayHello("Gopal");

}

}

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

import java.util.HashMap;

import java.util.Map;

public class Main {

public static void main(String[] args) {

Map<Integer, String> map = new HashMap<>();

map.put(1, "App");

map.put(2, "Bank");

map.put(3, "Cash");

map.forEach((key, value) -> System.err.println(key + "-->>" + value));

}

}

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

import java.util.Arrays;

import java.util.List;

public class Main {

public static void main(String[] args) {

List<String> names = Arrays.asList("App", "Bank", "Cash");

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

}

}

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

=> Function Interfaces :- That contains exactly one abstract method. It can have multiple default and static methods. They are use with Lambda and method refrences.

>> java.util.function package

Ex:-

@FunctionalInterface

interface MyFunction {

void sayHello();

}

public class Main {

public static void main(String[] args) {

MyFunction myFunction = () ->

System.out.println("Hello");

myFunction.sayHello();

}

}

> Predicate<T> :- Tests a condition and returns a boolean.

Ex:-

public static void main(String[] args) {

Predicate<Integer> isEven = num -> num % 2 == 0;

System.out.println(isEven.test(4));

System.out.println(isEven.test(7));

}

> Function<T, R> : Takes an argument and returns a result.

Ex:

T : input type

R : output type

Ex: -

Function<String, Integer> stringLength = str -> str.length();

System.out.println(stringLength.apply("Java"));

System.out.println(stringLength.apply("Hello world"));

> Consumer<T> : Takes an argument and return nothing(void)

Ex:-

Consumer<String> printMsg = msg -> System.out.println("Message:" + msg);

printMsg.accept("Hello this is java8");

> Supplier<t> : Returns a value but takes no input.

Ex:-

Supplier<Double> randomNumber = () -> Math.random();

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

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

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

Ex:-

import java.util.Arrays;

import java.util.List;

import java.util.function.Predicate;

public class Main {

public static void main(String[] args) {

List<String> names = Arrays.asList("One", "Two", "Three");

Predicate<String> Alpha = name -> name.startsWith("T");

names.stream()

.filter(Alpha)

.forEach(System.out::println);

}

}

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

=> Stream API :-

>> Stream API use to process collections and in a functional programming style, It allows operations like filtering, mapping, sorting and reducing data using sequence of pipeline operations.

>> It does not store data but operates on a source, processing elements in parallel or sequentially.

>> Stream vs Collection :-

1) A collection stores a data, whereas a stream process it.

2) A stream is lazy, means commputations are only perform when needed.

3) Stream can be processed only onced.

>> Create a stream from a collection. array and direct from values

Ex:

import java.util.Arrays;

import java.util.List;

import java.util.stream.Stream;

public class Main {

public static void main(String[] args) {

List<String> names = Arrays.asList("Two", "Three", "Twice");

//Stream from a collection

Stream<String> stream1 = names.stream();

//Stream from an array

Stream<Integer> stream2 = Arrays.stream(new Integer[]{1, 2, 2, 4});

//Stream from values

Stream<String> stream3 = Stream.of("A", "B", "C");

//Empty Stream

Stream<String> emptyStream = Stream.empty();

}

}

>> Stream Operations :- There are two type of operation

1) Intermediate Operations : return a new stream (lazy evaluation)

Ex:- filter(), map(), sorted(),distinct(), limit()

Ex:

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Main {

public static void main(String[] args) {

List<String> names = Arrays.asList("Two", "Three", "Twice", "One");

List<String> filteredNames = names.stream()

.filter(name -> name.startsWith("T"))

.collect(Collectors.toList());

System.out.println(filteredNames);

}

}

Ex:

import java.util.Arrays;

import java.util.List;

import java.util.stream.Collectors;

public class Main {

public static void main(String[] args) {

List<String> names = Arrays.asList("Two", "Three", "Twice");

List<String> upperCaseNames = names.stream()

.map(String :: toUpperCase) //Coverts each string to uppercase

.collect(Collectors.toList());

System.out.println(upperCaseNames);

}

}

Ex:

List<Integer> numbers = Arrays.asList(5,3,8,1,9);

List<Integer> sortedNumbers = numbers.stream()

.sorted()

.collect(Collectors.toList());

System.out.println(sortedNumbers);

Ex:

.distinct()

.collect(Collectors.toList());

2) Terminal Operations : produce a result (end the stream)

Ex:- forEach(), collect(), count(), reduce(), min(), max()

Ex :-

List<String> names = Arrays.asList("Two", "Three", "Twice");

names.stream().forEach(System.out::println);

Ex:

List<String> names = Arrays.asList("Two", "Three", "Twice");

List<String> nameList = names.stream().collect(Collectors.toList());

System.out.println(nameList);

Ex:

List<Integer> numbers = Arrays.asList(5,3,8,1,9);

long count = numbers.stream().count();

System.out.println("Count" + count);

Ex:-

List<Integer> numbers = Arrays.asList(1,2,3,4,5);

int sum = numbers.stream()

.reduce(0, Integer::sum);

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

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

>> Parallel Streams(Multithreading) :

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

=> Some Other Features related to java 8 :-

Ex:

//Default and static methods in interfaces

//Interfaces can now have default and static methods.

interface MyInterafce {

default void show() {

System.out.println("its default method");

}

static void display() {

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

}

}

class MyClass implements MyInterafce{}

public class Main {

public static void main(String[] args) {

MyClass obj = new MyClass();

obj.show(); //call a default method

MyInterafce.display(); //Calls a static method

}

}

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

=> Optional Classes : They helps to avoid NullPointerException

Optional<String> optional = Optional.ofNullable(null);

System.out.println(optional.orElse("Default Value"));

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

=> New Date and Time API :- It provides a modern approach to handling dates, times, time zones and duration, this api replaces old java.util.Date and Calender packages.

Ex: -

LocalDate date = LocalDate.now();

LocalTime time = LocalTime.now();

LocalDateTime dt = LocalDateTime.now();

System.out.println("Date:" + date);

System.out.println("Time:" + time);

System.err.println("DateTime:" + dt);

>> If create a specfic date LocalDate.of(1992, 8, 15)

>> If Add/Subtract days month and years

>> Same for time

today.plusDays()

today.minusYears()

Ex:-

//get current date-time in specific time zone

ZonedDateTime nowInIndia = ZonedDateTime.now(ZoneId.of("Asia/Kolkata"));

System.out.println("Current Date-Time in india : " + nowInIndia);

>> duration and datetimeformatter

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

=> Multithreading :-

>> thread :- It is a smallest unit of execution in a program. Java uses threads to perform multiple task simultaneously Ex: background operations, improving the performance and utilizing system resources.

>> Multithreading in java is a feature that allows concurrent executions of two or more parts of a program to maximize use of CPU utilization, Each part of that program is called thread and java provides built in support for multithreading.

>> Life Cycle of Thread :-

> New :- Thread is created but not started

> Runnable :- ready to run, waiting for CPU

> Running : Thread execute

> Blocked/waiting : Waiting for a resources or signal

> Terminated : Thread has completed execution

>> Create Threads in java :-

A) Extending Thread Class

Ex:-

public class Main extends Thread{

public void run() {

for(int i = 1 ; i <= 10; i ++) {

System.out.println(Thread.currentThread().getName() + " - " + i);

try {

Thread.sleep(500);

} catch (InterruptedException e) {

System.out.println(e);

}

}

}

public static void main(String[] args) {

Main t1 = new Main();

Main t2 = new Main();

t1.start(); //Calls run() method

t2.start();

}

}

B) Implementing Runnable interface

Ex:-

class Main implements Runnable {

public void run(){

for(int i = 0; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " - " + i);

}

}

public static void main(String[] args) {

Thread t1 = new Thread(new Main());

Thread t2 = new Thread(new Main());

t1.start();

t2.start();

}

}

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

>> Built-in methods :-

> start () : call run()

> run() : deine the thread's execution logic

> sleep(ms) : puts the thread to sleep for a specified time

> join() : Waits for the thread to finish execution

> yield() : Suggest that the thread schedular give CPU to another thread

> isAlive() : checks thread is still running

> getName() : retrives teh thread's name

> setName("Thread-name") : sets the name for the Threads

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

\*\*\*\* Thread pooling(ExecutorService) :

>> Instead of creating new threads each time, Java provides a thread pool to manage mutiple threads.

>> Thread-pool is collection of worker threads that execute tasks, we reuse exiting thread from pool

>> Java provides ExecutorService, to manage a thread pool

Ex:-

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class Main {

public static void main(String[] args) {

//create a thread pool of 3 threads

ExecutorService executor = Executors.newFixedThreadPool(3);

//Submit tasks using lambda express

for (int i = 0; i <= 5; i ++) {

int taskNumber = i;

executor.execute(() -> {

System.out.println("Task :" + taskNumber + " Executed By " + Thread.currentThread().getName());

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

}

executor.shutdown();

}

}

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

=> Synchronized Multi-threading in Java :- vSynchronization ensures that multiple threads access a shared resource safely by allowing only one thread at a time to execute a critical section of code.

//Printing a table using multiple threads

class TablePrinter {

//Method to print multiplication table(without synchronization)

void printTable(int number) {

for(int i =1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " - " + number + " x " + i + " = " + (number \* i));

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

//Thread class for printing the table

class TableThread extends Thread {

TablePrinter tablePrinter;

int number;

public TableThread(TablePrinter tablePrinter, int number) {

this.tablePrinter = tablePrinter;

this.number = number;

}

public void run() {

tablePrinter.printTable(number);

}

}

public class Main {

public static void main(String[] args) {

TablePrinter tablePrinter = new TablePrinter();

TableThread t1 = new TableThread(tablePrinter, 5);

TableThread t2 = new TableThread(tablePrinter, 7);

t1.start();

t2.start();

}

}

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

Problem :- The output is interleaved because both threads are modifying the printTable() method.

>> Incorrect order due to race condition :

>> A race condition occurs in multithreading when multiple threads access and want to modify a shared recsource, leading to incorrect results.

\*\*\*\*\* we can prevent race condition with synchronization(synchronized) by allowing only one thread to execite the critial section.

Ex: With synchronization (correct output) :

//Printing a table using multiple threads

class TablePrinter {

//Method to print multiplication table(with synchronization)

synchronized void printTable(int number) {

for(int i =1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " - " + number + " x " + i + " = " + (number \* i));

try {

Thread.sleep(500);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

//Thread class for printing the table

class TableThread extends Thread {

TablePrinter tablePrinter;

int number;

public TableThread(TablePrinter tablePrinter, int number) {

this.tablePrinter = tablePrinter;

this.number = number;

}

public void run() {

tablePrinter.printTable(number);

}

}

public class Main {

public static void main(String[] args) {

TablePrinter tablePrinter = new TablePrinter();

TableThread t1 = new TableThread(tablePrinter, 5);

TableThread t2 = new TableThread(tablePrinter, 7);

t1.start();

t2.start();

}

}

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

Don't do:-

=> Java Struts

=> Java AWT

=> Desktop AWT

=> Java Swing

=> Java Applet

=> Java Events

=> JSP(over stretch)

=> J2EE over projects

=> Spring framework

=> Java FX

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

>>>>>>>> Jsp & Servlet <<<<<<<<<<<<<<<

Jsp & servlet :-

>JSP:- Java server pages

JSP stands for Jakarta Server Pages( (JSP; formerly JavaServer Pages)). It is a server-side technology that is used for creating web applications. It is used to create dynamic web content. JSP consists of both HTML tags and JSP tags. In this, JSP tags are used to insert JAVA code into HTML pages. It is an advanced version of Servlet Technology i.e. a web-based technology that helps us to create dynamic and platform-independent web pages. In this, Java code can be inserted in HTML/ XML pages or both. JSP is first converted into a servlet by the JSP container before processing the client’s request.

Servlets :- Always work on the server side

=> capable to handle complex requests obtained from the JSPs and able to process it and able to provide the right direction.

It is written in java and platform independent.

=> Java servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the web server, process the request, produce the response, and then send a response back to the web server.

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

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

Three Layer Arch :-

The 3-layer architecture in a Java Servlet application typically consists of three layers:

Presentation Layer/view layer (UI Layer):

This layer is responsible for handling the user interface and user interactions.

In a Java Servlet application, this layer consists of JSP (JavaServer Pages) files and Servlets.

JSP files are used to display data to the user and collect inputs, while Servlets act as controllers that manage requests from JSP and forward them to the appropriate business logic.

Business Logic Layer (Service Layer):

This layer contains the core business logic of the application. It processes user inputs and applies business rules.

It may include Java classes (often called services) that perform specific tasks and interact with the Data Access Layer to fetch or manipulate data.

The business logic layer is responsible for ensuring the data integrity and implementing the business operations needed by the application.

Data Access Layer (DAO Layer):

This layer interacts with the database to perform CRUD (Create, Read, Update, Delete) operations.

It includes Data Access Objects (DAOs), which are Java classes responsible for communicating with the database using JDBC (Java Database Connectivity).

The DAO layer provides an abstraction so that other parts of the application do not need to directly interact with the database.

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

=> XML :-

XML stands for Extensible Markup Language. It's a file format and markup language that allows users to store, transmit, and reconstruct data. XML is human- and machine-readable, and is often used to store and transport data

=> How to set the Dynamic web module in eclipse IDE 24 :-

> file > dynamic web project create > choose web module 6.0 > generate the web.xml also

> click on server in the footer panel and downlaod install Apache Tomcat 10.1 Stable version with choosing prefered folder

> create index.html to test in file > click on run with server to open the port > http://localhost:8080/

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

=> Servlet Structure with doGet and doPost custom select :

Ex: -

package myPackage;

import jakarta.servlet.ServletException;

import jakarta.servlet.annotation.WebServlet;

import jakarta.servlet.http.HttpServlet;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException;

@WebServlet("/MyServlet")

public class MyServlet extends HttpServlet {

private static final long serialVersionUID = 1L;

public MyServlet() {

}

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.getWriter().append("Served at: ").append(request.getContextPath());

}

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

}

Keywords in structure :

> serialVersionUID = 1L

-- This is used when the servlet class implements the serializable interface (directly or indirectly) to support serialization.

\*\*\* Session Serialization :

-- Servlet may store objects in the session(HttpSession), which can be serialized and store.

- the serialVersionUID ensures that the session data remains compatible after redeployment.

> public MyServlet() {

super();

}

This shows the constructor.

In doGet :-

> getWriter() :- to write something on servlet

>append("<h2>Hello</h2>") to diplay something on browser side

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

The data that is being submitted to the server will be visible in the URL using query parameters like this “http://localhost:8080/HelloServlet/hello?myParam=myValue”.

So, if you are sending any sensitive information like passwords, you should not use the GET method as the data entered can be clearly visible in the browser URL.

POST method type and doPost() method

The doPost() method in servlets is used to process the HTTP POST requests. It is used to submit the data from the browser to the server for processing. The data submitted with POST method type is sent in the message body so it is secure and cannot be seen in the URL. And there is no limit on the data that can be sent through the POST method. Ideally, we need to use the POST method, to send the form data to the webserver. So, we will be using the doPost() method in this example. And we will learn how to handle some of the common HTML fields data such as Text field, Checkbox, Radio button, Dropdown, etc., values in the servlet.

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

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IMp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Data sent by the GET method is visible in the URL (as parameters after the question mark), the maximum length is 1024 characters

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

NewFile.jsp :

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title>First Jsp and servlet</title>

</head>

<body style="background-color : #ccc">

<form action="MyServlet">

<input name="num1" placeholder="Enter first number...">

<input name="num2" placeholder="Enter Second number...">

<button name="btn" value="1">Add</button>

<button name="btn" value="1">Sub</button>

<button name="btn" value="1">Multi</button>

<button name="btn" value="1">Devide</button>

</form>

</body>

</html>

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

Servlet Life Cycle :-

Step 1 : - Loading and instantiation :- Servlet is created

Step 2 :- Initialization = intit() method is called

Step 3:- Request handling :- service method(depends to doGet () , doPost () is call on request)

Step 4:- Destruction :- destroy() method clean up the resources

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

Session management:-

In servlet this technique is used to maintain the state of the user intracting with the web application across multiple HTTP requests.

=> HTTP is stateless, each request is independent, so session management ensure the specific information to a user maintain accross the request.

Hypertext Transfer Protocol (HTTP) is a protocol that allows web browsers and web servers to communicate and exchange information. It's the foundation of the World Wide Web and is used to load web pages.

=> Http cookies:-

A small piece of information that the server sends to the client, and the browser stores it. For session management, a cookie is sent with each HTTP response from the server to the client.

Ex:-

Cookies cookie = new Cokkie("user", "alok");

response.addCookie(cookie);

\*\*\* disabled by the user's choice

=> URL Rewriting:-

Session information is appended to the the URL as a query Parameter.

Servlet container automatically appends a session identifier to the URL for tracking user sessions.

Ex: -

String url = response.encodeURL("index.jsp");

>> sensitive information are exposed in this.

=> Hidden Form Fields: -

Session data can be passed through hidden fields in HTML forms

Each time the server receives the session data through these hidden fields

Ex:

<from action="ServiceURL" method="post">

<input type="hidden" name="sessionID" value="<% = sessionID %>">

<input type="Submit" value="Submit">

</form>

=> \*\*\* HttpSession Object: -

This is the most common approach for session management in a servlet.

>> the session object is created by the servlet container and can store user-specific data on the server side

Ex:-

HttpSession session = request.getSession();

session.setAttribute("username", username);

=> Session TimeOut =>

<session-config>

<session-timeout>30</session-timeout>

</session-config>

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

@Override : -

Overriding is a feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super-classes or parent classes. When a method in a subclass has the same name, the same parameters or signature, and the same return type(or sub-type) as a method in its super-class, then the method in the subclass is said to override the method in the super-class.

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

JDBC (Java Database Connectivity) is a Java API that enables Java applications to interact with databases. It provides methods to query and update data in relational databases like MySQL, Oracle, PostgreSQL, etc. JDBC is part of the Java Standard Edition and helps bridge Java code with SQL databases.

Key Components of JDBC

=>> JDBC Driver: This is a software component that establishes a connection between Java applications and databases. Different databases have their specific drivers (e.g., MySQL JDBC Driver, Oracle JDBC Driver).

.

=>> JDBC API: JDBC consists of several interfaces and classes that allow Java applications to communicate with the database. Key interfaces include:

DriverManager: Manages a list of database drivers and establishes connections.

Connection: Represents a connection to the database.

Statement: Executes static SQL queries.

PreparedStatement: Used to execute parameterized SQL queries.

ResultSet: Represents the result of a query, and allows retrieval of data from it. executeQueryse()

SQLException: Handles database-related errors. Alway use e.printStackTrace()

=>> Basic Steps to Use JDBC

1. Load the Driver: Load the appropriate JDBC driver for the database you’re connecting to.

2. Establish a Connection: Use DriverManager to connect to the database.

3. Create a Statement: Use Statement or PreparedStatement to execute SQL queries.

4. Execute the Query: Execute the SQL query using the statement object.

5. Process the Results: Use ResultSet to fetch the data.

6. Close the Connection: Always close the connection, statement, and result set to avoid memory leaks.

==>> Explanations of the Login-Registration Project Code:

1. Load the Driver: The line Class.forName("com.mysql.cj.jdbc.Driver"); loads the MySQL JDBC driver.

2. Establish a Connection: DriverManager.getConnection() is used to create a connection to the database using the provided URL, username, and password.

3. Create and Execute a Query: We create a PreparedStatement with a SQL query and execute it with pstmt.executeQuery().

4. Process the Result: The results of the query are fetched using a ResultSet, which iterates through the results row by row.

5. Close Resources: It's important to close the ResultSet, PreparedStatement, and Connection objects after use to prevent memory leaks.

Key Interfaces in JDBC:

1. Connection Interface:

Used to establish a connection with the database.

Example:

Connection conn = DriverManager.getConnection(URL, USER, PASSWORD);

2. Statement Interface:

Used to execute SQL queries.

There are two main types:

Statement: Used for static queries.

PreparedStatement: Used for dynamic or parameterized queries (provides better performance and security).

Example:

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM users");

3. ResultSet Interface:

Used to retrieve the data returned by a SQL query.

Methods to fetch data from a ResultSet: getString(), getInt(), getBoolean(), etc.

Example:

while (rs.next()) {

String name = rs.getString("name");

int id = rs.getInt("id");

}

Best Practices in JDBC:

Always close connections, statements, and result sets in a finally block or use a try-with-resources block.

Use PreparedStatement instead of Statement to prevent SQL injection attacks.

Manage exceptions properly with proper logging and error handling.

Use connection pooling in production environments to improve performance.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JDBC provides a low-level interface to interact with databases and is foundational to many Java applications that require database connectivity. However, frameworks like Hibernate, JPA, or Spring Data JPA often sit on top of JDBC to provide higher-level abstraction for database access.

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

Project:- Login & Register Project using MySQL Connectivity

========================Hibernate Framework========================

Hibernate/ORM:- Hiberante is an object relational mapping (ORM) framework for java. Database interactions by mapping java objects to database tables.

Framework:- this is the special software that provides abstraction on one or more technologies and simplify the app development of the programms.

Persistence operations on the database with fewer/no database queries.

\*\*\* the process stroring and managing the data for long time.

Persistance Strore:- it is the place where data is hold for the long time. ex:- files, db(mySQL, postgresql,oracle etc)

Presistance Data:- Tha data which we have hold in the strore. ex:- file\_info, DB table and their records.

Persistance Operations:-

=> insert, update, delete and select operations perofrmed on persistance data.

=> CRUD, CURD, SCUD operations.

Persistance Logics:-

the logics which we write down for the CRUD/ CURD/SCUD operations.

Persistance Technology/Framework:-

ex:- JDBC, hibernate(framework), ORM/Spring Data(framework) etc

Programming language:-

these languages are like raw materials provide basic fundamental features of application dev.

=> they have the syntex and sementic overview.

=> basic requrinments will fullfil with language.

Software technology:-

=> this is the software specification that provides set of rules and guidelines for the vendor componies to develop implemntation by taking support one or more programming language.

=> JDBC, JSP, Servlet

Technology APIs:-

=> interfaces, Abstract Classes represent both rule and guidelines.

=> concrete classes represent guidelines only

Java abstract class:- is a class that can not be initiated by itself, it needs to be subclassed by another class to use its properties. An abstract class is declared using the “abstract” keyword in its class definition.

A concrete class:- is a class that has an implementation for all of its methods. They cannot have any unimplemented methods. It can also extend an abstract class or implement an interface as long as it implements all their methods. It is a complete class and can be instantiated.

Open Technology:- open to use

proprietary technology:- all APIs are not avilable for everyone, its available only for special vendor.

ORM(object relational mapping) framework:-

it provides abstraction on JDBC technology and simplifies to develop db software for implementing the persistance Logics.

Ex:- Hibernate(Softree/redhat), Eclipse link(Eclipse), Toplink(Oracle), iBatis, OJB(Apache), OpenJPA(SunMicrosystem/Oracle corportation)

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

\*\*OR-Mapping (object relational mapping) it means linking/mapping classes to db tables.

\*\*The process of mapping java classes with db tables, member variables,of java class with columns of db table and representing the record db tables with the object of java classs having synchronization between them is called OR-mapping.

\*\*Synchronization means whatever modification that will do in all objects will directly reflects to db table records and vice-versa.

\*\*\*\*Based on mode of app development, there are two types of frameworks:-

1.) Invasive Framework

\*\* The Programmer developed classes of framework based app development are tightly coupled with frameworks APIs, thses classes must be developed either implementing framework APIs interfaces or extending classes.

\*\*Due to this we can't take out our project classes to other frameworks for execution.

\*\* Ex:- Struts(Outdated), servlets etc

2.)Non Invasive Framework

The Programmer developed classes are loosely coupled with framework, not placed for the long time.

ex:- spring, hibernate and top links etc.

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

Hibernate-configuration DTD(Document type defination file)

The Hibernate-configuration DTD is a file-based configuration document that contains references to mapping files and property settings. It contains elements such as class-cache, collection-cache, event, and listener.

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

===========Dependency storage file====================

Pom.xml :-

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.testhibernate</groupId>

<artifactId>MavenProject</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>MavenProject</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.0.2.Final</version>

<type>pom</type>

</dependency>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>6.0.2</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

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

hiberanate.cfg.xml :-

<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/mydb\_test</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">mysql@12345</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="show\_sql">true</property>

</session-factory>

</hibernate-configuration>

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

> Dialect: is a class that acts as a bridge between Java JDBC types and SQL types, which contains the mapping between java language data type and database datatype. Dialect allows Hibernate to generate SQL optimized for a particular relational database.

> hibernate.hbm2ddl.auto : Automatically validates or exports schema DDL to the database when the SessionFactory is created. With create-drop, the database schema will be dropped when the SessionFactory is closed explicitly.

e.g. validate | update | create | create-drop

So the list of possible options are,

validate: validate the schema, makes no changes to the database.

create-only: database creation will be generated.

drop: database dropping will be generated.

update: update the schema.

create: creates the schema, destroying previous data.

create-drop: drop the schema when the SessionFactory is closed explicitly, typically when the application is stopped.

none: does nothing with the schema, makes no changes to the database

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

=> Dialect specifies the type of database used in hibernate so that hibernate can switch to the database-specific SQL generator code.

=> Data Definition Language (DDL) is used to create and modify the structure of objects in a database using predefined commands and a specific syntax. These database objects include tables, sequences, locations, aliases, schemas and indexes.

=> hibernate. ddl-auto explicitly and the standard Hibernate property values are none , validate , update , create-drop

==================Important====================

Annotations in Hibernate:

=> @Table(name = "abc")

To mention the table name manually otherwise hibernate generates it automatically.

=> @Column

@Column(name="user\_column" nullable=false length=100)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Imortant\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

=> @GeneratedValue

>>> This is used with @Id to indicate how the primary key should be generated.

@GeneratedValue(strategy = GenerationType.AUTO)

GenerationType.AUTO:- JPA chooses a strategy based on the database

GenerationType.IDENTITY:- the DB auto-increment feature

GenerationType.TABLE:- uses a separate table to generate primary keys (minimum use)

GenerationType.SEQUENCE:- uses a DB sequence (only used in Oracle)

@SequenceGenerator(name = " ", sequenceName="", allocationSize = 1) >>> only use with GenerationType.SEQUENCE

@Temporal >> This annotation is used to map java.util.Date or Calendar to specify the type of (Date, Time, Timestamp)

Ex: -

@Temporal(temporal.Type.Date)

@Enumerated: - used to enum types to database columns

Ex: -

@Enumerated(EnumType.String)

@Lob:- it is used to map large objects like binary data (BLOB) and char type data (CLOB) to a column in the database.

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

Entity: It represents a lightweight persistence domain object that is mapped to a table in a relational database. Entities are part of the JPA, manage relational data in Java applications.

Properties:-

1) Mapped to a table :- An entity class represents a table in database. and each instance of this class represent a row.

2) Presistance :- Entities are used to store data that you want to preist accross diffrent states (that need to be saved in a database)

3) \*\*\* POJO (Plain old Java Object) Class :- Entity Classes are regular Java Classes (POJOs) but are annoted for persistance

Always use the @Entity Annotation to use this.

=======================Mapping========================

Mapping in hibernate : -

Mapping defines the relationship between Java Objects(entities) and database tables.

=> Basic Mapping : - this is use to map single data type like int , string etc to relatable databases column.

\*\*\*\*\*\*\*\*\*\*

=> Association Mapping : hibernate supports relationshps between entities in the form of association.

> @OneToOne :

Specifies a single-valued association to another entity that has one-to-one multiplicity. It is not normally necessary to specify the associated target entity explicitly since it can usually be inferred from the type of the object being referenced.

Ex : -

@OneToOne

@JoinColumn(name="name")

> @OnetoMany :-

Specifies a many-valued association with one-to-many multiplicity.

Ex:-

@OneToMany(mappedBy="employee")

> @ManytoOne :

@ManyToOne

@JoinColumn(name="employee\_details")

> ManytoMany :

@ManyToMany

@JoinTable(name="Cart\_items", joinColumns= @JoinColumn(name="Cart\_id"), inverseJoinColumns=@JoinColumn(name="item\_id"))

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

HQL:- Hibernate Query Language: Hibernate Query Language is an object-oriented query language similar to SQL but built specifically for Hibernate.

> It's used to perform database operations with Hibernate entities rather than directly with tables.

> This abstraction allows us to work with your Java objects and classes, providing more flexibility and aligning with Better object-oriented principles.

Aspects of HQL: -

=> Entity-Based Querying

=> SQL-like syntax

=> Automatic Table and Column Mapping

=> Supports Relationships and associations based upon SQL Join.

In HQL, queries operate on entity classes mapped to a table

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

Small task: - add the annotation for table generation and columns

@Data

public class EmployeeEntity {

private Long id;

private String name;

private String phone;

private String email;

}

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

Common HQL Syntax and Examples:

=> Basic Select Query:

Ex:-

Query query = session.createQuery("FROM Employee");

List<Employee> employee = query.list();

=> Select with Where Clause:

Ex:-

Query query = session.createQuery("FROM Employee WHERE salary > :minSalary");

query.setParameter("minSalary, 50000");

List<Employee> employee = query.list();

=> Aggregate Functions:

HQL always supports the aggregate function from SQL. ex:- SUM, AVG, COUNT, MIN, MAX

Ex:

Query query = session.createQuery("SELECT AVG(e.salary) FROM Emplyoee e");

Double avgSalary = (Double) query.uniqueResult();

\*\*\*\*

=> Pagination:

HQL allows us to limit the number of rows returned in a query, which is useful for pagination results.

Query query = session.createQuery("FROM Employee");

query.setFirstResult(0); //Offset

query.setMaxResults(10); //Limit

List<Employee> employee = query.list();

Update Query:-

Query query = session.createQuery("UPDATE Employee SET salary = salary + :increase WHERE department = :dept");

query.setParameter("increase", 5000);

query.setParameter("dept", sales);

int result = query.executeUpdate();

Delete Query:-

Query query = session.createQuery("DELETE FROM Emloyee Where id = :empId");

query.setParamter("empId", 4);

int result = query.executeUpdate();

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

Criteria API in Hibernate:

This is object oriented approach to query the database without writing SQL directly. It allows developers to construct queries using Java code, making queries more readable and easier to use.

> Type-safe Queries

> Dynamic Queries

> Object Oriented approach

> Automatic fetching of the data

> Easy for association and joins

How to create a criteria API : -

=> Setup a session and criteria builder

Ex :-

Session session= sessionFactory.openSession();

CriteriaBuilder builder = session.getCriteriaBuilder();

=> Create Criteria Query : -

CriteriaQuery<Employee> query = builder.createQuery(Employee.class);

Root<Employee> root = query.from(Employee.class);

query.select(root);

=> Add Predicates(Conditions) :- use for the filter

uPredicates condition = builder.equal(root.get("department"), "Sales");

query.where(condition)

=> Execute Query : -

List<Employee> employee = session.createQuery(query).getResultList();

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

Ex:-

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Employee> query = builder.createQuery(Employee.class);

Root<Employee> root = query.from(Employee.class);

//Define the predicates now

Predicates departmentPredicates = builder.equal(root.get("department"), "Sales");

Predicates salaryPredicates = builder.greaterThan(root.get("salary"), 50000);

//Apply the predicates now

query.select(root).where(builder.and(departmentPredicates, salaryPredicates ));

//Execution the query

List<Employee> employee = session.createQuery(query).getResultList();

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

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

=> Spring Framework java :- It is a powerful framework for java that simplfies enterprise application development, It provides comprehnsive support for dependency injection technology and integration with newly java technology.

>> Dependency injection :-Manages object dependencies to make application more moduler and modified.

>> AOP (Aspect oriented programming) :- Sperate cross cutting like logging and security

>> Spring MVC :- A web framework for building java web applications.

>> Spring Security :- Manage authentication and authrization in java app.

>> Spring Cloud :- Supports building microservices with service dicovery and confguration.

>> Spring Boot :- A sub-project that simplify the spring application development with auto configuration and some extra features.

>> Spring Framework Modules :-

> Spring core : Provides DI and IoC (Inversion of control)

> Spring web : supports building web application with MVC pattern and REST APIs

> Spring JDBC : database connectivity(application.properties with boot)

> Spring Security : role-based access control

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

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

Spring Boot is a popular framework for building standalone, production-grade Spring-based applications.

Maven:- a powerful automation tool managing the project's life cycle and dependencies.

@Controller Annotation: Spring @Controller annotation is also a specialization of @Component annotation. The @Controller annotation indicates that a particular class serves the role of a controller.

@RestController Annotation: RestController is used for making restful web services with the help of the @RestController annotation. This annotation is used at the class level and allows the class to handle the requests made by the client.

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

Inversion of Control (IoC):-

Inversion of Control (IoC) is a principle where the control of objects is transferred to a container or framework. This means that instead of your code creating objects and managing their lifecycles, a framework (like Spring Boot) does it for you.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Imp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

in simple word to manage and create Objects in beans.

Dependency Injection:-

Dependency Injection (DI) is a way of providing the dependencies an object needs rather than the object creating them itself. This helps in making the code more modular, easier to test, and maintain.

\*\*\*\* In Spring Boot, the IoC container is responsible for managing the lifecycle and dependencies of the objects (called beans). Here’s how it works:

Bean: An object that is managed by the Spring IoC container.

ApplicationContext: The Spring container that holds and manages beans.

a) Define a Bean:- We will tell Spring which classes it should manage by annotating them with special annotations.

==>> Use @Component, @Service, @Repository, or @Controller to indicate that a class should be managed by Spring.

==>> Use @Configuration and @Bean to define beans in a configuration class.

b) Inject Dependencies: You inject the dependencies a class needs by using annotations like @Autowired.

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

Data Access Object (DAO):-

DAO stands for Data Access Object. It's a design pattern used to separate the data persistence logic in an application. By using DAO, you can isolate the application/business layer from the persistence layer, which makes the code easier to maintain, test, and reuse.

Key Concepts of DAO

Separation of Concerns: The DAO pattern separates the data access logic from the business logic of the application.

Encapsulation: It encapsulates the details of the data source and how data is retrieved, stored, and manipulated.

Interface: Typically, a DAO defines an interface that specifies the operations for accessing data (CRUD operations - Create, Read, Update, Delete).

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

The 3-layer architecture, also known as the three-tier architecture, is a software design pattern and well-established software architecture that separates an application into three main logical layers. These layers are:

Presentation Layer (UI Layer)

Business Logic Layer (BLL) or Application Layer

Data Access Layer (DAL) or Data Layer

1. Presentation Layer (UI Layer)

Purpose: This layer is responsible for handling all user interface and user experience (UI/UX) aspects of the application. It displays data to the user and sends user commands to the business logic layer.

Components:

Views (e.g., HTML pages, JSP pages, Thymeleaf templates)

Controllers (e.g., Spring MVC controllers) Http requst and responses

User Interface components (e.g., buttons, forms)

Technologies:

HTML/CSS/JavaScript for web interfaces

Frameworks like Angular, React, or Vue.js for rich client-side applications

JSP, Thymeleaf for server-side rendering

ex:-

@RestController

@RequestMapping("/users")

public class UserController {

private final UserService userService;

@Autowired

public UserController(UserService userService) {

this.userService = userService;

}

@GetMapping("/{id}")

public User getUserById(@PathVariable Long id) {

return userService.getUserById(id);

}

@PostMapping

public User createUser(@RequestBody User user) {

return userService.createUser(user);

}

}

2. Business Logic Layer (BLL) or Application Layer

Purpose: This layer contains the core functionality and business rules of the application. It processes data received from the presentation layer and communicates with the data access layer to retrieve or store data.

Components:

Services

Business logic classes

Domain models

Technologies:

Spring Framework (e.g., Spring Boot, Spring Service)

EJB (Enterprise JavaBeans)

@Service

public class UserService {

private final UserDao userDao;

@Autowired

public UserService(UserDao userDao) {

this.userDao = userDao;

}

public User createUser(User user) {

return userDao.save(user);

}

public User getUserById(Long id) {

return userDao.findById(id).orElse(null);

}

public void deleteUser(Long id) {

userDao.deleteById(id);

}

}

3. Data Access Layer (DAL) or Data Layer

Purpose: This layer is responsible for interacting with the database or any other storage system. It provides an abstraction layer for data access and manipulation.

Components:

Data Access Objects (DAOs)

Repositories

Database connection and querying logic

Technologies:

Spring Data JPA

Hibernate

JDBC

@Repository

public interface UserDao extends CrudRepository<User, Long> {

// Custom query methods can be added here

}

Coupling in Java

In object-oriented design, Coupling refers to the degree of direct knowledge that one element has of another. In other words, how often do changes in class A force-related changes in class B.

Types of Coupling

There are two types of coupling:

1. Tight coupling:

In general, Tight coupling means the two classes often change together. In other words, if A knows more than it should about the way in which B was implemented, then A and B are tightly coupled.

Example: If you want to change the skin, you would also have to change the design of your body as well because the two are joined together – they are tightly coupled. The best example of tight coupling is RMI(Remote Method Invocation).

// Java program to illustrate

// tight coupling concept

class Subject {

Topic t = new Topic();

public void startReading()

{

t.understand();

}

}

class Topic {

public void understand()

{

System.out.println("Tight coupling concept");

}

}

Explanation of the above Program: In the above program the Subject class is dependents on Topic class. In the above program Subject class is tightly coupled with Topic class it means if any change in the Topic class requires Subject class to change. For example, if Topic class understand() method change to gotit() method then you have to change the startReading() method will call gotit() method instead of calling understand() method.

// Java program to illustrate

// tight coupling concept

class Volume

{

public static void main(String args[])

{

Box b = new Box(5,5,5);

System.out.println(b.volume);

}

}

class Box

{

public int volume;

Box(int length, int width, int height)

{

this.volume = length \* width \* height;

}

}

Output:

125

Explanation:

In the above example, there is a strong inter-dependency between both the classes. If there is any change in Box class then they reflects in the result of Class Volume.

2. Loose coupling

In simple words, loose coupling means they are mostly independent. If the only knowledge that class A has about class B, is what class B has exposed through its interface, then class A and class B are said to be loosely coupled. In order to over come from the problems of tight coupling between objects, spring framework uses dependency injection mechanism with the help of POJO/POJI model and through dependency injection its possible to achieve loose coupling. Example : If you change your shirt, then you are not forced to change your body – when you can do that, then you have loose coupling. When you can’t do that, then you have tight coupling. The examples of Loose coupling are Interface, JMS.

// Java program to illustrate

// loose coupling concept

public interface Topic

{

void understand();

}

class Topic1 implements Topic {

public void understand()

{

System.out.println("Got it");

}

} class Topic2 implements Topic {

public void understand()

{

System.out.println("understand");

}

} public class Subject {

public static void main(String[] args)

{

Topic t = new Topic1();

t.understand();

}

}

Explanation :

In the above example, Topic1 and Topic2 objects are loosely coupled. It means Topic is an interface and we can inject any of the implemented classes at run time and we can provide service to the end user.

// Java program to illustrate loose coupling concept

// Define Interface

interface calVolume {

int volResult(int length, int width, int height);

}

// Box implements calVolume interface

class Box implements calVolume {

public int volResult(int length, int width, int height) {

return length \* width \* height;

}

}

public class Main{

public static void main(String args[]) {

calVolume calVolume = new Box();

int volume = calVolume.volResult(5, 5, 5);

System.out.println(volume);

}

}

Output:

125

Explanation :

In the above program, there is no dependency between both the classes. If we change anything in the Box classes then we dont have to change anything in Volume class.

Which is better tight coupling or loose coupling?

In general, Tight Coupling is bad in but most of the time, because it reduces flexibility and re-usability of code, it makes changes much more difficult, it impedes test ability etc. loose coupling is a better choice because A loosely coupled will help you when your application need to change or grow. If you design with loosely coupled architecture, only a few parts of the application should be affected when requirements change. Lets have a look on the pictorial view of tight coupling and loose coupling:Coupling in JavaDifference between tight coupling and loose coupling

Tight coupling is not good at the test-ability. But loose coupling improves the test ability.

Tight coupling does not provide the concept of interface. But loose coupling helps us follow the GOF principle of program to interfaces, not implementations.

In Tight coupling, it is not easy to swap the codes between two classes. But it’s much easier to swap other pieces of code/modules/objects/components in loose coupling.

Tight coupling does not have the changing capability. But loose coupling is highly changeable.

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

=> Difference between Three Layer Arch. and MVC Arch.

> Use Three Layer Arch. When designing a structured backend for REST APIs

> Use Spring MVC when building a full-stack web application with a dedicated UI.

> Both work together in a Spring MVC project, where backend code follows the three-layer architecture. while the frontend uses MVC for rendering views.

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

>> groupId:-

Represents the organization or group that owns the project.

Often based on a reversed domain name (e.g., com.example for a project from example.com).

Helps to uniquely identify a project across all projects, even if multiple projects have the same artifactId.

Used to organize and find projects within the Maven repository.

>> artifactId:-

Represents the unique name of the project artifact.

Used to construct the artifact's final name (e.g., myapp-1.0.jar).

Along with groupId and version, it uniquely identifies the artifact in Maven.

The artifact is typically a JAR file but can also be other formats like WAR or EAR.

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

=> MyController.java : part of the MVS arch. and control the application REST services with Request Mapping annotation

Ex:

package com.myapplication.MyBootProject;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class MyController {

@RequestMapping("u")

public static String myMethod() {

return "<p>Hello world</p>";

}

}

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

>> Show the landing page/home

resource/static/index.html :-

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Document</title>

</head>

<body>

<form action="cal-form" method="get">

<input type="text" name="num1">

<input type="text" name="num2">

<button type="submit">Submit</button>

</form>

</body>

</html>

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

>> MyController :-

package com.myapplication.MyBootProject;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class MyController {

@RequestMapping("u")

public static String myMethod() {

return "<p>Hello world</p>";

}

@GetMapping("cal-form")

public static String getData(@RequestParam int num1, @RequestParam int num2) {

int add = num1 + num2;

return "Result of Arithmatic Op : " + add;

}

}

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

Dependency in spring initizer : -

=> Spring Web:-

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

=> MySQL JDBC driver :- Driver and connector j for database connection

=> Spring Data JPA :- Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

=> Spring Data MongoDB NoSQL :- Store data in flexible, JSON-like documents, meaning fields can vary from document to document and data structure can be changed over time.

=> H2 Database SQL :- Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

=> Thymeleaf (Template Engine) :- A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

=> Spring for RabbitMQ Messaging :- Gives your applications a common platform to send and receive messages, and your messages a safe place to live until received.

=> Validation I/O :- Bean Validation with Hibernate validator.

=> Spring Boot DevTool :-

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

=> Spring Security Security :- Highly customizable authentication and access-control framework for Spring applications.

=> Spring Boot Actuator :- Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

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

=> Simple Calculator App in Spring Boot:

>> Create controller: MyController.java in src

package com.calculater.CalculatorApp;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class MyController {

@RequestMapping("u")

public static String MyMethod() {

return "<p>Hello world</p>";

}

}

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

>>> Calculator App data:

>> MyController.java:

package com.calculater.CalculatorApp;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;

@RestController

public class MyController {

@RequestMapping("u")

public static String MyMethod() {

return "<p>Hello world</p>";

}

@GetMapping("cal-form")

public static String getData(@RequestParam int num1, @RequestParam int num2) {

int add = num1 + num2;

return "Result of Arithmatic Op :" + add;

}

}

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

>>> resources >> static >> index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Calculator Page</title>

</head>

<body>

<form action="cal-form" method="get">

<input type="text" name="num1">

<input type="text" name="num2">

<button type="submit">Submit</button>

</form>

</body>

</html>

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

Thymeleaf:- It is a Java-based template engine used in Spring Boot applications to build dynamic web pages. Allows the template to be viewed as static HTML when opened in a browser.

=> Add the Thymeleaf dependency

=> build pages in resource/templates

=> Create a Controller to render a template

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

Spring Security:- to secure web application and REST APIs.

It provides authentication, authorization, and protection.

1.) Authentication:- Verifies the identity of users trying to access the application.

> Configure an in-memory database or external identity provider, ex:- OAuth2, LDAP

Ex- using username and password for login

>Default login page is auto configured and we can customize it

2) Authorization :- Specify role based access using annotations like @PreAuthorize @Secured

Ex:- Allowing only users with ADMIN role to access a particular endpoint.

@PreAuthorize("hasRole('ADMIN')")

@GetMapping("/admin")

public String adminPage() {

return "Admin page";

}

3) Password Management :- Supports password encoding (Ex:- BCrypt) to securely store passwords in databases and on page protection.

Ex:-

@Bean

public PasswordEncoder passwordEncoder(){

return new BCryptPasswordEncoder();

}

4) Securing REST APIs :-

> Protect REST endpoints with token-based authentication, such as JWT (JSON Web Token)

> Example :- Configuring JWT in spring security to secure API Calls.

5) OAuth2 and OpenID Connect :-

> Support third party authentication providers (ex:- Google, Facebook, Linkdin)

> Example :- Config OAuth2 for soical login

6) CSRF (cross site request forgery) :-

> Enabled by default to protect against csrf attack.

> for REST APIs we can disable it when required

Ex:-

http.csrf().disable()