**Assignment-1 (dated-30/1/2022)**

1. **Create a java application as mentioned below flow:-**

**Welcome to COllege Management**

**A: Add Student Result**

**X: Exit**

**a**

**Enter student id:**

**10**

**Enter student Name:**

**durga**

**Enter marks in hindi:**

**1**

**Enter marks in english:**

**1**

**Enter marks in maths:**

**1**

**Enter marks in science:**

**1**

**Enter marks in social:**

**1**

**Student added successfully..**

**Do you want to continue (Y/N):**

**C: Check student Result**

**A: Add Student Result**

**X: Exit**

**c**

**Enter student id to check result:**

**1**

**student id not found**

**Do you want to continue (Y/N):**

**y**

**C: Check student Result**

**A: Add Student Result**

**X: Exit**

**c**

**Enter student id to check result:**

**10**

**Student Result{id='10', name='null', marks=Subject Marks{hindi=1, english=1, maths=1, science=1, social=1}, result='Pass', total=5, percentage=100.0}**

**Do you want to continue (Y/N):**

**n**

**Bye**

**You need to show on startup menu to add student result and check student result,** **if there are no students present then only add student result should appear once you add student result then you need to enable to check the student result until you press X or don’t want to continue then only you should exit the program.**

**Code:-**

**package** com.pack;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**import** java.util.stream.Collectors;

**public** **class** Application {

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

List<Employee> stdList = **new** ArrayList<>();

List<Employee> lst = stdList.stream().distinct().collect(Collectors.*toList*());

System.***out***.println("Welcome to College Management");

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

**int** studentId;

**boolean** n=**true**;

String name;

String result;

**int** pick;

**int** hindi;

**int** english;

**int** maths;

**int** science;

**int** social;

**int** total,percentage;

**do** {

System.***out***.println("enter valid choice");

System.***out***.println("c - check student result");

System.***out***.println("a - add student result");

System.***out***.println("x - exit");

**char** choice=sc.next().charAt(0);

**switch**(choice) {

**case** 'a':System.***out***.println("Add student result");

System.***out***.println("StudentId-");

studentId=sc.nextInt();

System.***out***.println("Enter student Name");

name = sc.next();

System.***out***.println("marks in hindi-");

hindi=sc.nextInt();

System.***out***.println("marks in english-");

english=sc.nextInt();

System.***out***.println("marks in maths-");

maths=sc.nextInt();

System.***out***.println("marks in science-");

science=sc.nextInt();

System.***out***.println("marks in social-");

social=sc.nextInt();

total=hindi+english+maths+science+social;

percentage=(total/5)\*100;

**if**(percentage>=50){

result="pass";

}**else** {

result="fail";

}

lst.add(**new** Employee(studentId, name,hindi,english,maths,science,social,result,total,percentage));

System.***out***.println("Student added successfully");

**break**;

**case** 'c':System.***out***.println("Check student result");

System.***out***.println("Enter Student ID=");

**int** in=sc.nextInt();

**for**(Employee e:lst) {

**if**(e.getStdId()==in) {

lst.stream().filter(emp->emp.getStdId()==in).forEach(System.***out***::println);

System.*exit*(0);

}

**else** {

System.***out***.println("Student id not found");

}

}

**break**;

**case** 'x':System.*exit*(1);

**default**:System.***out***.println("Invalid Input");

}

System.***out***.println("Do you want to continue (Y/N)");

pick = sc.next().charAt(0);

} **while** (pick == 'Y' || pick == 'y');

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

System.*exit*(1);

}

@Override

**public** String toString() {

**return** "MyClass []";

}

}

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

**int** stdId,salary;

String stdNm;

**int** m1,m2,m3,m4,m5;

String result;

**float** percentage;

**int** total;

**public** Employee() {

}

**public** Employee(**int** stdId,String stdNm,**int** m1,**int** m2,**int** m3,**int** m4,**int** m5,String result,**float** percentage,**int** total) {

**super**();

**this**.stdId = stdId;

**this**.stdNm = stdNm;

**this**.m1=m1;

**this**.m2=m2;

**this**.m3=m3;

**this**.m4=m4;

**this**.m5=m5;

**this**.result=result;

**this**.percentage=percentage;

**this**.total=total;

}

**public** **int** getStdId() {

**return** stdId;

}

**public** **void** setStdId(**int** stdId) {

**this**.stdId = stdId;

}

**public** String getStdNm() {

**return** stdNm;

}

**public** **void** setStdNm(String stdNm) {

**this**.stdNm = stdNm;

}

**public** **int** getM1() {

**return** m1;

}

**public** **void** setM1(**int** m1) {

**this**.m1 = m1;

}

**public** **int** getM2() {

**return** m2;

}

**public** **void** setM2(**int** m2) {

**this**.m2 = m2;

}

**public** **int** getM3() {

**return** m3;

}

**public** **void** setM3(**int** m3) {

**this**.m3 = m3;

}

**public** **int** getM4() {

**return** m4;

}

**public** **void** setM4(**int** m4) {

**this**.m4 = m4;

}

**public** **int** getM5() {

**return** m5;

}

**public** **void** setM5(**int** m5) {

**this**.m5 = m5;

}

**public** String getResult() {

**return** result;

}

**public** **void** setResult(String result) {

**this**.result = result;

}

**public** **float** getPercentage() {

**return** percentage;

}

**public** **void** setPercentage(**float** percentage) {

**this**.percentage = percentage;

}

**public** **int** getTotal() {

**return** total;

}

**public** **void** setTotal(**int** total) {

**this**.total = total;

}

@Override

**public** String toString() {

**return** "Student Details [stdId=" + stdId + ",stdNm=" + stdNm + ", m1=" + m1 + ", m2=" + m2

+ ", m3=" + m3 + ", m4=" + m4 + ", m5=" + m5 + ", result=" + result + ", percentage=" + percentage

+ ", total=" + total + "]";

}

}

**Assignment-2 (dated-2/2/2022)**

1. **What is Metaspace and Heap Memory?:-**

**Ans:-**

* Metaspace is a memory which by default auto increases its size depending on the underlying Operating system. In other words It is memory that VM uses to store the class Metadata. Garbage collection is automatically triggered when the class metadata usage reaches its maximum Metaspace size.
* Heap Memory - Heap memory is a part of memory allocated to JVM, which is shared by all executing threads in the application. It is the part of JVM in which all class instances and are allocated.

1. **Generate Multiples of 2 Until 20 using recursive function:-**

**Code-**

**package** com.pack;

**public** **class** Assignment {

**static** **void** multiple(**int** a,**int** i) {

**if**(i<=10) {

System.***out***.println(a+"x"+i+"="+a\*i);

*multiple*(a,i+1);

}

}

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

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

**int** a=2;

Assignment.*multiple*(a,1);

}

}

**Output-**

2x1=2

2x2=4

2x3=6

2x4=8

2x5=10

2x6=12

2x7=14

2x8=16

2x9=18

2x10=20

1. **Check If Two strings are equal or not:-**

**Code:-**

**package** com.pack;

**import** java.util.Scanner;

**public** **class** Assignment {

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

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

String s1=sc.next();

String s2=sc.next();

**boolean** a=s1.equals(s2);

**if**(a==**true**) {

System.***out***.println("Strings are equal");

}

**else** {

System.***out***.println("Strings are not equal");

}

}

}

**Input-**

durga

vedant

**Output-**

Strings are not equal

1. **Print Character count in String say string s=”helloworld”:-**

**Code-**

**package** com.pack;

import java.util.Scanner;

**public** **class** Assignment {

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

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

**int**[] count=**new** **int**[30];

System.***out***.println("Enter a string-");

String s1=sc.next();

**int** n=s1.length();

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

count[s1.charAt(i)-'a']++;

}

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

**if**(count[s1.charAt(i)-'a']!=0) {

System.***out***.println(s1.charAt(i)+"-"+count[s1.charAt(i)-'a']+" ");

count[s1.charAt(i)-'a']=0;

}

}

}

}

**Output:-**

Enter a string-

helloworld

h-1

e-1

l-3

o-2

w-1

r-1

d-1

**5.Why Java is platform Independent?:-**

**Ans:-**

Java is platform-independent because it does not depend on any type of platform. Java compiled code can run on all operating systems that support Java. In Java, programs are compiled into byte code and that byte code is platform-independent and any machine can execute that byte code.

**6. Can we create class as Final:-**

**Ans:-**

Yes , We can create a class as Final. A class can be made final by using the final keyword. The final class cannot be inherited and so the final keyword is commonly used with a class to prevent inheritance.

**7. Consider we have a employee class with empid, empname, salary and list of employees, get highest salary paid employee data.**

**Code:-**

**package** com.pack;

**public** **class** Employee **implements** Comparable<Employee>{

**int** empId,salary;

String empNm;

**public** Employee() {

}

**public** Employee(**int** empId,**int** salary,String empNm) {

**super**();

**this**.empId = empId;

**this**.empNm = empNm;

**this**.salary=salary;

}

**public** **int** getEmpId() {

**return** empId;

}

**public** **void** setEmpId(**int** empId) {

**this**.empId = empId;

}

**public** **int** getSalary() {

**return** salary;

}

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

**this**.salary = salary;

}

**public** String getEmpNm() {

**return** empNm;

}

**public** **void** setEmpNm(String empNm) {

**this**.empNm = empNm;

}

@Override

**public** String toString() {

**return** "Employee [empId=" + empId + ",salary=" + salary + ", empNm=" + empNm + "]";

}

@Override

**public** **int** hashCode() {

**final** **int** prime = 31;

**int** result = 1;

result = prime \* result + empId;

result = prime \* result + ((empNm == **null**) ? 0 : empNm.hashCode());

result = prime \* result + salary;

**return** result;

}

@Override

**public** **boolean** equals(Object obj) {

**if** (**this** == obj)

**return** **true**;

**if** (obj == **null**)

**return** **false**;

**if** (getClass() != obj.getClass())

**return** **false**;

Employee other = (Employee) obj;

**if** (empId != other.empId)

**return** **false**;

**if** (empNm == **null**) {

**if** (other.empNm != **null**)

**return** **false**;

} **else** **if** (!empNm.equals(other.empNm))

**return** **false**;

**if** (salary != other.salary)

**return** **false**;

**return** **true**;

}

@Override

**public** **int** compareTo(Employee o) {

**if**(**this**.salary>o.salary) {

**return** -1;

}

**else** **if**(**this**.salary<o.salary){

**return** 1;

}

**else** {

**return** 0;

}

}

}

**package** com.pack;

**import** java.util.ArrayList;

**import** java.util.Collections;

**import** java.util.List;

**public** **class** Assignment {

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

List<Employee> empList = **new** ArrayList<>();

empList.add(**new** Employee(1,2000,"a"));

empList.add(**new** Employee(7,2700,"b"));

empList.add(**new** Employee(6,2500,"d"));

empList.add(**new** Employee(5,3000,"b"));

empList.add(**new** Employee(2,1900,"e"));

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

Collections.*sort*(empList);

System.***out***.println(empList.get(0));

}

}

**Output:-**

[Employee [empId=1,salary=2000, empNm=a], Employee [empId=7,salary=2700, empNm=b], Employee [empId=6,salary=2500, empNm=d], Employee [empId=5,salary=3000, empNm=b], Employee [empId=2,salary=1900, empNm=e]]

Highest salary employee details=

Employee [empId=5,salary=3000, empNm=b]

**8. Consider a list of Duplicate Values remove duplicate value and get unique values from list:-**

**Code:-**

**package** com.pack;

**import** java.util.ArrayList;

**import** java.util.LinkedHashSet;

**import** java.util.List;

**import** java.util.Set;

**public** **class** Assignment {

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

List<Integer> empList = **new** ArrayList<>();

empList.add(1);

empList.add(7);

empList.add(6);

empList.add(7);

empList.add(2);

empList.add(5);

empList.add(1);

System.***out***.println("Before Removal-");

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

Set<Integer> empSet=**new** LinkedHashSet<Integer>(empList);

System.***out***.println("After Removal-");

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

}

}

**Output:-**

Before Removal-

[1, 7, 6, 7, 2, 5, 1]

After Removal-

[1, 7, 6, 2, 5]

**9. Can we write try and finally without catch block what is the use. What is its use?:-**

**Ans:-**

Yes, we can have try without catch block by using finally block. Finally Block is used for running cleanup code after execution of try block. Finally block always executes even if you have exception or return statement in try block except.

**Assignment-3 (dated-3/2/2022)**

1. **Create an application to insert, update and delete the database(emp\_id, emp\_nm, emp\_sal):-**

**Code:-**

package com.pack;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Scanner;

public class JdbcDemo {

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

Scanner sc=new Scanner(System.in);

Class.forName("org.apache.derby.jdbc.ClientDriver");//loading drivers

Connection conn=DriverManager.getConnection("jdbc:derby://localhost:1527/training;create=true","derby","derby");

try{

Details e1=new Details();

char pick;

do {

System.out.println("Enter a choice");

System.out.println("1. View Employee Data");

System.out.println("2. Update Employee Data");

System.out.println("3. Insert Employee Data");

System.out.println("4. Delete Employee Data ");

int choice=sc.nextInt();

switch(choice) {

case 1:

e1.getEmployeeDetails();

break;

case 2:

e1.updateEmployee();

break;

case 3:

e1.insertEmployee();

break;

case 4:

e1.deleteEmployee();

break;

default:

System.out.println("INvalid choice");

}

System.out.println("Do you want to continue (Y/N)");

pick = sc.next().charAt(0);

}while (pick == 'Y' || pick == 'y');

System.out.println("Bye");

}catch(Exception e) {

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

}

}

}

class Details

{

private String emp\_nm;

private int emp\_id;

private int emp\_sal;

public void getEmployeeDetails() throws InstantiationException, IllegalAccessException, ClassNotFoundException, SQLException {

Class.forName("org.apache.derby.jdbc.ClientDriver");//loading drivers

Connection conn=DriverManager.getConnection("jdbc:derby://localhost:1527/training;create=true","derby","derby");

Scanner input = new Scanner(System.in);

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

int emp\_id = input.nextInt();

PreparedStatement sm = conn.prepareStatement("SELECT \* FROM app.employee WHERE emp\_id=?");

sm.setInt(1,emp\_id);

ResultSet rs = sm.executeQuery();

if(rs.next()==false)

{

System.out.println("No such record found in the database");

}

else

{

System.out.println("emp\_id-"+rs.getInt(1)+" emp\_nm-"+rs.getString(2)+" emp\_sal-"+rs.getInt(3));

}

}

public void insertEmployee() throws InstantiationException, IllegalAccessException, ClassNotFoundException, SQLException {

Class.forName("org.apache.derby.jdbc.ClientDriver");//loading drivers

Connection conn=DriverManager.getConnection("jdbc:derby://localhost:1527/training;create=true","derby","derby");

Scanner input = new Scanner(System.in);

System.out.println("Enter employee Name");

emp\_nm = input.nextLine();

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

emp\_id = input.nextInt();

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

emp\_sal= input.nextInt();

PreparedStatement sm = conn.prepareStatement("INSERT INTO app.employee VALUES(?,?,?)");

sm.setInt(1,emp\_id);

sm.setString(2,emp\_nm);

sm.setInt(3,emp\_sal);

int rm = sm.executeUpdate();

if(rm>0) {

System.out.println("Record inserted");

}

}

public void updateEmployee() throws InstantiationException, IllegalAccessException, ClassNotFoundException, SQLException {

Class.forName("org.apache.derby.jdbc.ClientDriver");//loading drivers

Connection conn=DriverManager.getConnection("jdbc:derby://localhost:1527/training;create=true","derby","derby");

Scanner input = new Scanner(System.in);

System.out.println("Enter employee name-");

String inname=input.next();

System.out.println("Enter employee salary-");

int insal=input.nextInt();

PreparedStatement sm = conn.prepareStatement("UPDATE app.employee SET emp\_sal=? WHERE emp\_nm=?");

sm.setInt(1,insal);

sm.setString(2,inname);

int rm = sm.executeUpdate();

if(rm>0) {

System.out.println("Record Updated");

}

else {

System.out.println("No such record");

}

}

public void deleteEmployee() throws InstantiationException, IllegalAccessException, ClassNotFoundException, SQLException {

Class.forName("org.apache.derby.jdbc.ClientDriver");//loading drivers

Connection conn=DriverManager.getConnection("jdbc:derby://localhost:1527/training;create=true","derby","derby");

System.out.println("Enter the Employee name to be deleted");

Scanner input = new Scanner(System.in);

String inputname=input.nextLine();

PreparedStatement sm = conn.prepareStatement("DELETE FROM app.employee WHERE emp\_nm=?");

sm.setString(1,inputname);

int rm = sm.executeUpdate();

if(rm>0) {

System.out.println("Record deleted");

}else {

System.out.println("No such record");

}

}

}

**Assignment-4 (dated-4/2/2022)**

1. **Garbage Collector and how it works:-**

**Ans:-**

The task of garbage collection (GC) in the Java virtual machine (JVM) is to automatically determine what memory is no longer being used by a Java application and to recycle this memory for other uses. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory.

1. **What is Heap Space?:-**

**Ans:-**

Heap Space is the area of memory used to store objects instantiated by applications running on the JVM. When the JVM is started, heap memory is created and any objects in the heap can be shared between threads as long as the application is running.

1. **What is Metaspace?:-**

**Ans:-**

* Metaspace is a memory which by default auto increases its size depending on the underlying Operating system. In other words It is memory that VM uses to store the class Metadata. Garbage collection is automatically triggered when the class metadata usage reaches its maximum Metaspace size.

1. **What is Java Memory Model?:-**

**Ans:-**

The Java memory model specifies how and when different threads can see values written to shared variables by other threads, and how to synchronize access to shared variables when necessary. The Java memory model used internally in the JVM divides memory between thread stacks and the heap.

1. **What is young and old generation?:-**

**Ans:-**

* The young generation is the place where all the new objects are created. When the young generation is filled, garbage collection is performed.
* The Old Generation is used to store long surviving objects. Typically, a threshold is set for young generation object and when that age is met, the object gets moved to the old generation. Eventually the old generation needs to be collected. This event is called a major garbage collection.

1. **What is Eden and Survivor Space?:-**

**Ans:-**

* Eden Space is the pool from which memory is initially allocated for most objects.
* Survivor Space is the pool containing objects that have survived the garbage collection of the Eden space.