**Name: Rutuja Tatewar**

Assessment 1

1. What is metaspace and heap memory?

Metadata is defined as data about the data. It is the documentation about the information which is required by the users. It is used to analyze data usage and performance of particular data.

Heap Memory is created when JVM starts up and used by the application as long as the application is running. Whenever an object is created it occupies the space in the heap memory while reference of that object creates in the stack. It dynamically handles the memory blocks.

1. Generate multiples of 2 until 20 using recursive function

**package** com.pack;

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

**static** **void** multiplies(**int** a,**int** b) {

**if**(b>10)

**return** ;

System.***out***.println("2"+" \* "+b+" = "+a\*b);

*multiplies*(a,b+1);

}

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

**int** a=2;

*multiplies*(a, 1);

}

}

Sol:

2 \* 1 = 2

2 \* 2 = 4

2 \* 3 = 6

2 \* 4 = 8

2 \* 5 = 10

2 \* 6 = 12

2 \* 7 = 14

2 \* 8 = 16

2 \* 9 = 18

2 \* 10 = 20

1. Check if two strings are equal or not

**package** com.pack;

**import** java.util.Scanner;

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

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

String s1;

String s2;

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

System.***out***.println("Enter strings to compare:");

s1=sc.nextLine();

s2=sc.nextLine();

**if**(s1.equals(s2)) {

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

}

**else** {

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

}

}

}

Sol:-

Enter strings to compare:

hello

hello

Strings are equals.

1. Print the character count in a string say, string s ="helloworld" print h-1, e-1, l-3,o-2

**package** com.pack;

**import** java.util.Scanner;

**import** java.util.HashMap;

**import** java.util.Map;

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

**static** **void** characterCount(String s) {

HashMap<Character, Integer> count = **new** HashMap<Character, Integer>();

**char**[] str = s.toCharArray();

**for**(**char** c : str) {

**if**(count.containsKey(c)) {

count.put(c, count.get(c)+1);

}

**else** {

count.put(c, 1);

}

}

**for**(@SuppressWarnings("rawtypes") Map.Entry entry: count.entrySet()) {

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

}

}

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

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

String s;

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

s=sc.nextLine();

*characterCount*(s);

sc.close();

}

}

Sol:

Enter string:

helloworld

r 1

d 1

e 1

w 1

h 1

l 3

o 2

1. Why java is platform independent?

Platform independent means that the compiled code can run on all operating systems. A machine understands code which is in machine level language.So the compiler converts high level language that is human readable language into format of machine level language that machine will understood. So compiler translates the source code from a programming language to a executable code. This executable code is executed by CPU directly or it may be a byte code that is interpreted by a virtual machine.

1. Can we create class as final.

Yes, we can create class as final. But a class which is declared as final cannot be subclassed or it cannot be override.

1. Consider we have employee class with empid, empname and salary and list of employees get the the highest salary paid employee data

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));

}

}

1. Consider a list of duplicate values remove duplicate value and get unique values from the list

**package** com.pack;

**import** java.util.Scanner;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

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

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

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

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

ArrayList<Integer> numlist = **new** ArrayList<Integer>();

**int** num;

System.***out***.println("Enter numbers in list:");

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

num=sc.nextInt();

numlist.add(num);

}

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

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

List<Integer> listwithoutDuplicates = numlist.stream().distinct().collect(Collectors.*toList*());

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

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

sc.close();

}

}

Sol:

Enter numbers in list:

2

3

4

5

2

3

4

5

7

8

Before

[2, 3, 4, 5, 2, 3, 4, 5, 7, 8]

After

[2, 3, 4, 5, 7, 8]

1. Can we write try and finally without catch block what is the use.

Yes, it is possible to use try and finally block without catch block. Because whether the exception occurred in try block, even then finally block will execute.

1. Program to update, insert and remove required data:
2. View table
3. Update data
4. Insert data
5. Remove data
6. View updated table
7. Exit

Sol:

package com.pack;

import java.util.Scanner;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.SQLException;

import java.sql.Statement;

import java.sql.ResultSet;

public class Assessment2 {

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

// TODO Auto-generated method stub

Scanner sc= new Scanner(System.in);

System.out.println("Choose menu:");

System.out.println("1.View Employee Data\r\n"+ "2.Update Employee Data\r\n"

+ "3.Add Employee\r\n"+ "4.Delete Employee Data\r\n"+"5.View Updated Employee Table|r\n"+"6.Exit");

int choice = sc.nextInt();

Class.forName("org.apache.derby.client.ClientAutoloadedDriver");//loading drivers

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

Statement st = conn.createStatement();

ResultSet rs;

PreparedStatement ps;

switch(choice) {

case 1:

rs=st.executeQuery("SELECT \* FROM app.employee");

System.out.println("EmpId\t "+"EmpName\t\t "+"EmpSal\t");

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

while(rs.next()) {

System.out.println(rs.getInt(1)+"\t "+rs.getString(2)+"\t\t "+rs.getInt(3));

}

break;

case 2:

System.out.println("Enter id to update the data:");

int id=sc.nextInt();

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

String nm=sc.next();

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

int sal= sc.nextInt();

ps=conn.prepareStatement("UPDATE app.employee SET emp\_name,emp\_sal WHERE emp\_id=?");

ps.setInt(1, id);

ps.setString(2, nm);

ps.setInt(3, sal);

ps.executeUpdate();

System.out.println("Data updated successfully.");

break;

case 3:

System.out.println("Enter employee id to insert:");

int id1=sc.nextInt();

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

String nm1=sc.nextLine();

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

int sal1=sc.nextInt();

ps=conn.prepareStatement("INSERT INTO app.employee(emp\_id,emp\_name,emp\_sal)VALUES(?,?,?)");

ps.setInt(1, id1);

ps.setString(2, nm1);

ps.setInt(3, sal1);

ps.executeUpdate();

System.out.println("Employee data added successfully.");

break;

case 4:

System.out.println("Enter empid whose data to be removed:");

int id2=sc.nextInt();

ps=conn.prepareStatement("DELETE FROM app.employee WHERE emp\_id=?");

ps.setInt(1, id2);

rs=ps.executeQuery();

int rid=rs.getInt("emp\_id");

if(rid == id2) {

System.out.println("Data deleted successfully.");

}

else {

System.out.println(id2+" id is not found in database.");

}

break;

case 5:

System.out.println("EmpId\t "+"EmpName\t\t "+"EmpSal\t");

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

ps=conn.prepareStatement("SELECT \* FROM app.employee");

rs=ps.executeQuery();

while(rs.next()) {

System.out.println(rs.getInt(1)+"\t "+rs.getString(2)+"\t\t "+rs.getInt(3));

}

break;

case 6:

System.exit(0);

default:

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

break;

}

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

char ch4 = sc.next().charAt(0);

while (ch4 == 'Y' || ch4 == 'y');

System.out.println("Bye");

sc.close();

}

}

1. Create a java application as mentioned above flow:

C: Check Student Result

A: Add student result

X: Exit

**package** com.pack;

**import** java.util.\*;

**public** **class** Student {

Scanner sc1 = **new** Scanner(System.***in***);

**boolean** choice() {

System.***out***.println("Do you want to continue?"+'\n'+"true or false");

**boolean** option = sc1.nextBoolean();

**return** option;

}

**void** display() {

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

System.***out***.println("A: Add Student Result"+'\n'+"C:Check Student Result"+'\n'+"X:Exit");

}

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

Student input = **new** Student();

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

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

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

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

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

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

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

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

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

List<Float> per = **new** ArrayList<Float>();

**boolean** option = input.choice();

**if**(option == **true**) {

input.display();

String choice = sc.nextLine();

**switch**(choice) {

**case** "A":

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

**int** id1 = sc.nextInt();

id.add(id1);

System.***out***.println("Enter student's name:");

String name1 = sc.next();

name.add(name1);

System.***out***.println("Enter marks of English:");

**int** eng1 = sc.nextInt();

eng.add(eng1);

System.***out***.println("Enter marks of Hindi:");

**int** hin1 = sc.nextInt();

hin.add(hin1);

System.***out***.println("Enter marks of Maths:");

**int** math1 = sc.nextInt();

math.add(math1);

System.***out***.println("Enter marks of Science:");

**int** sci1 = sc.nextInt();

sci.add(sci1);

System.***out***.println("Enter marks of Social Studies:");

**int** sst1 = sc.nextInt();

sst.add(sst1);

**int** total=0;

total=eng1+hin1+math1+sci1+sst1;

tot.add(total);

**float** percentage=0;

percentage = (total\*100) / 500;

per.add(percentage);

**break**;

**case** "C":

System.***out***.println("Enter id to check result:");

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

**if**(idc <= id.size()) {

System.***out***.println("id="+id.get(idc)+'\n'+"Name="+name.get(idc));

System.***out***.println("English="+eng.get(idc));

System.***out***.println("Hindi="+hin.get(idc));

System.***out***.println("Maths="+math.get(idc));

System.***out***.println("Science="+sci.get(idc));

System.***out***.println("Social Studies="+sst.get(idc));

System.***out***.println("Total="+tot.get(idc));

System.***out***.println("Percentage="+per.get(idc));

**if**(per.get(idc) >= 50) {

System.***out***.println("Result= PASS");

}

**else** {

System.***out***.println("Result= FAIL");

}

}

**else** {

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

}

**break**;

**case** "X":

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

**break**;

**default**:

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

input.choice();

}

}

**else** {

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

}

}

}

Sol: Do you want to continue?

true or false

true

Welcome to College Management

A: Add Student Result

C:Check Student Result

X:Exit

A

Enter the student id:

01

Enter student's name:

Raju

Enter marks of English:

45

Enter marks of Hindi:

78

Enter marks of Maths:

89

Enter marks of Science:

89

Enter marks of Social Studies:

90

1. What is Garbage Collector and how it works?

Garbage Collection is the process of reclaiming the runtime unused memory by destroying the unused objects.

Java Garbage Collection is the process by which Java programs perform automatic memory management. Java programs compile into bytecode that can be run on a Java Virtual Machine (JVM).

When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program.

Over the lifetime of a Java application, new objects are created and released. Eventually, some objects are no longer needed. You can say that at any point in time, the heap memory consists of two types of objects:

* *Live* - these objects are being used and referenced from somewhere else
* *Dead* - these objects are no longer used or referenced from anywhere

The garbage collector finds these unused objects and deletes them to free up memory.

Java garbage collection is an automatic process. The programmer does not need to explicitly mark objects to be deleted.

The garbage collection implementation lives in the JVM. Each JVM can implement its own version of garbage collection. However, it should meet the standard JVM specification of working with the objects present in the heap memory, marking or identifying the unreachable objects, and destroying them with compaction.

1. What is java memory model?

The java memory model specifies how the Java virtual machine works with the computer’s memory. 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.

1. What is young and old generations?

Young Generation:

From a high level, the young generation is where all new objects start out. Once they’re allocated in the Java code, they go specifically to this subsection called the eden space.Eventually, the eden space fills up with objects. At this point, a minor garbage collection event occurs.

That’s where the marking algorithm I described earlier comes into play. Some objects (those that are referenced) are marked, and some (those that are unreferenced) are not. Those that had been marked then move onto another subsection of the young generation called S0 of the survivor space (note that the survivor space itself is split into two parts, S0 and S1). Those left unmarked are cleared out by Java’s automatic garbage collection.

The Old Generation:

The old generation can be thought of as where long-lived objects lie. Basically, if objects reach a certain age threshold after multiple garbage collection events in the young generation, then they can then be moved to the old generation.

When objects get garbage collected from the old generation, a major garbage collection event occurs. The old generation is comprised of only one section called the tenured generation. This is why in conversation or in reading sometimes that the two terms have come to be mostly interchangeable.

1. What is Eden and survivor space?

The Young generation or new space is divided into two portions called Eden space and Survivor space:

* Eden space : When we create a object, the memory will be allocated from eden space.
* Survivor space: This contains the object , that have survived from Young garbage Collection or Minor garbage collection. We have two equally divided survivor spaces called S0 and S1.