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.