**ASSIGNMENT**

1. What is Metaspace and heap memory?

* Metaspace is a new memory space – starting from the Java 8 version; it has replaced the older PermGen memory space. The most significant difference is how it handles memory allocation.

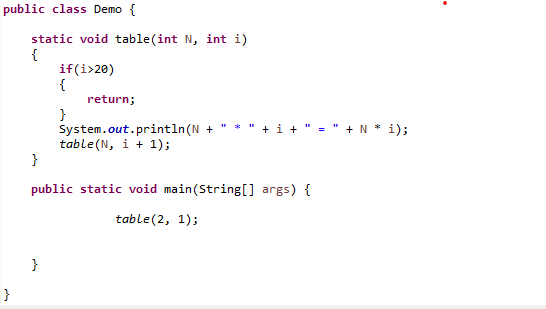
Specifically, this native memory region grows automatically by default

* JVM Memory Structure is divided into multiple memory area like heap area, stack area, method area, PC Registers etc.

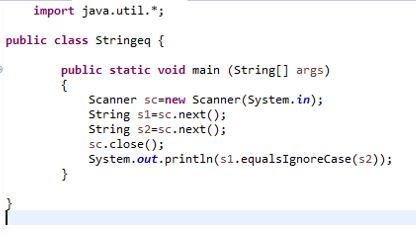
JVM Memory area parts

Here, the heap area is one of the most important memory areas of JVM. Here, all the [java objects](https://www.geeksforgeeks.org/classes-objects-java/) are stored. The heap is created when the JVM starts. The heap is generally divided into two parts. That is:

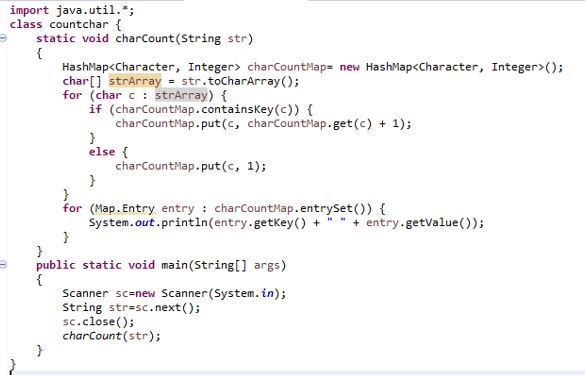
1. Young Generation(Nursery): All the new objects are allocated in this memory. Whenever this memory gets filled, the [garbage collection](https://www.geeksforgeeks.org/garbage-collection-java/) is performed. This is called as Minor Garbage Collection.
2. Old Generation: All the long lived objects which have survived many rounds of minor garbage collection is stored in this area. Whenever this memory gets filled, the garbage collection is performed. This is called as Major Garbage Collection.
3. Generate multiples of 2 until 20 using recursive function



1. Check if two strings are equal or not



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



1. Why java is platform independent?

Java compiler produces a unique type of code called bytecode unlike c compiler where compiler produces only natively executable code for a particular machine. When the Java program runs in a particular machine it is sent to java compiler, which converts this code into intermediate code called bytecode. This bytecode is sent to Java virtual machine (JVM) which resides in the RAM of any operating system. JVM recognizes the platform it is on and converts the bytecodes into native machine code. Hence java is called platform independent language.

1. Can we create class as final

Yes, you can declare that your class is final; that is, that your class cannot be subclassed. There are (at least) two reasons why you might want to do this: security reasons and design reasons.

Security: One mechanism that hackers use to subvert systems is to create subclasses of a class and then substitute their class for the original. The subclass looks and feels like the original class but does vastly different things possibly causing damage or getting into private information. To prevent this kind of subversion, you can declare your class to be final and prevent any subclasses from being created. The String class in the java.lang package is a final class for just this reason. The String class is so vital to the operation of the compiler and the interpreter that the Java system must guarantee that whenever a method or object uses a String they get exactly a java.lang.String and not some other string. This ensures that all strings have no strange, inconsistent, undesirable, or unpredictable properties.

If you try to compile a subclass of a final class, the compiler will print an error message and refuse to compile your program. In addition, the bytecode verifier ensures that the subversion is not taking place at the bytecode level by checking to make sure that a class is not a subclass of a final class.

Design: Another reason you may wish to declare a class as final are for object-oriented design reasons. You may think that your class is "perfect" or that, conceptually, your class should have no subclasses.

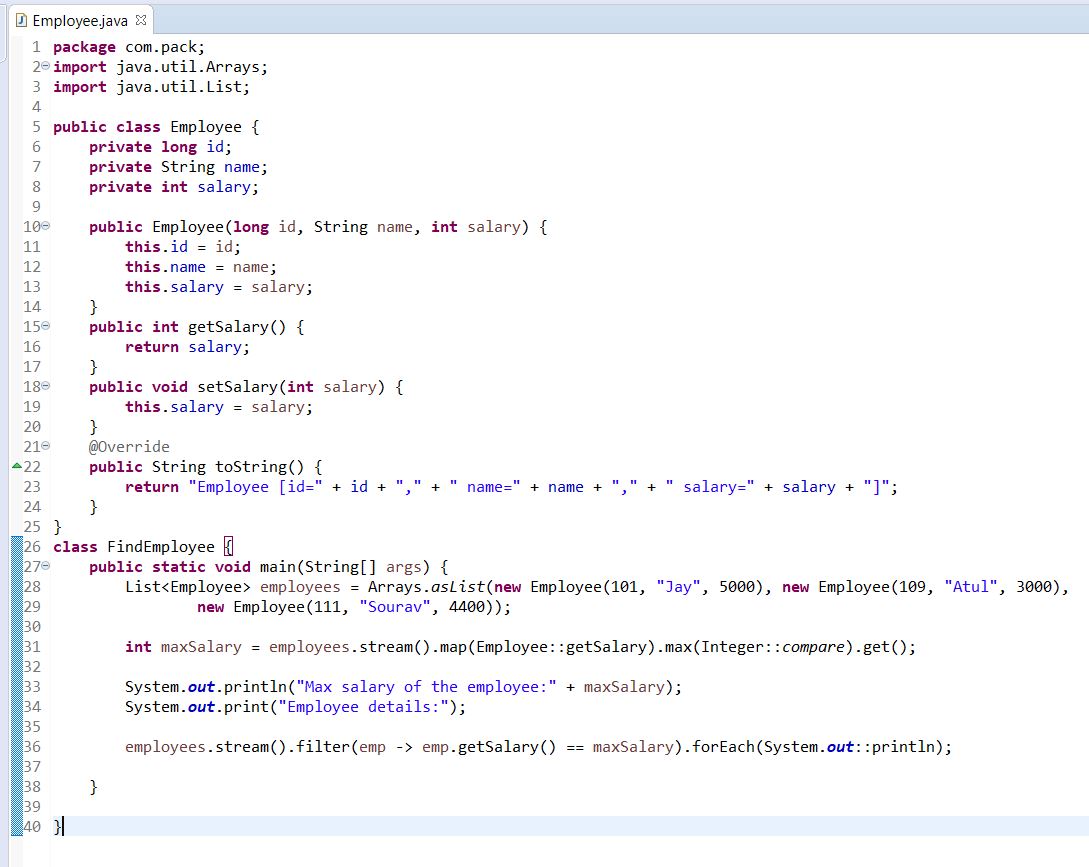
To specify that your class is a final class, use the keyword final before the class keyword in your class declaration. For example, if you wanted to declare your (perfect) ChessAlgorithm class as final, its declaration would look like this:

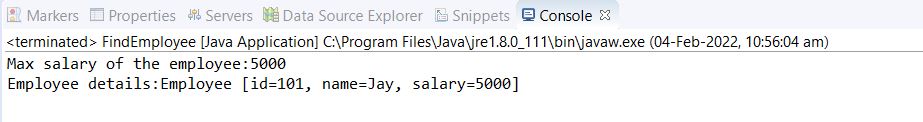
final class ChessAlgorithm {

. . .

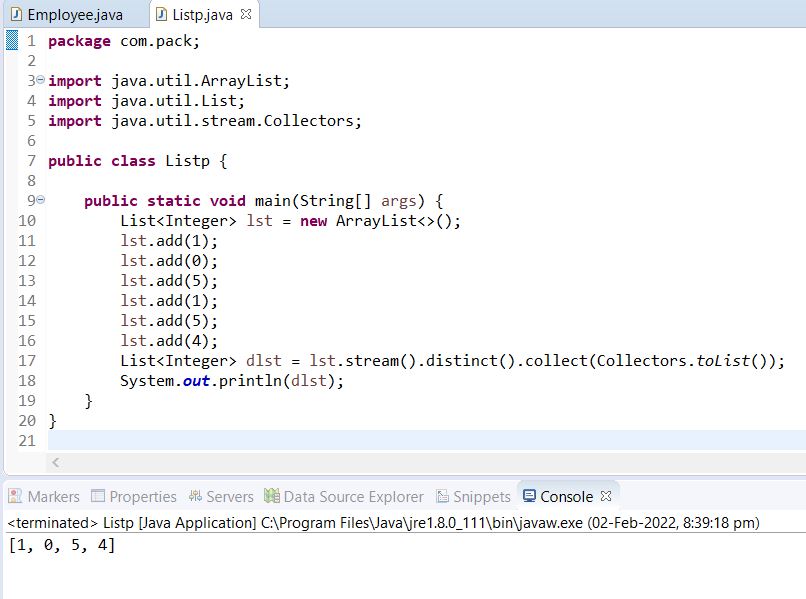
}

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





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



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

Yes, we can have try without catch block by using finally block.

You can use try with finally. As you know finally block always executes even if you have exception or return statement in try block except in case of System.exit().

1. Create a java application for College Management.

package l.pack;

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

import java.util.stream.Collectors;

public class StudentD {

public static void main(String[] args) {

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

List<Employee> dlst = empList.stream().distinct().collect(Collectors.toList());

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

char choice;

String result;

int count = 0;

do {

if (count > 0) {

System.out.println("C : Check Student Result");

}

System.out.println("A : Add Student Result");

System.out.println("X : Exit");

Scanner sc = new Scanner(System.in);

String c = sc.next();

switch (c.toUpperCase()) {

case "A":

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

int id = sc.nextInt();

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

String name = sc.next();

System.out.println("Enter marks in hindi");

int hindi = sc.nextInt();

System.out.println("Enter marks in english");

int eng = sc.nextInt();

System.out.println("Enter marks in maths");

int maths = sc.nextInt();

System.out.println("Enter marks in science");

int science = sc.nextInt();

System.out.println("Enter marks in social");

int social = sc.nextInt();

int total = hindi+eng+maths+science+social;

float per = (float) total / 5;

per = per \* 100;

if (per > 50) {

result="Pass";

} else {

result="Fail";

}

dlst.add(new Employee(id, name,hindi,eng,maths,science,social,result,total,per));

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

count++;

break;

case "C":

System.out.println("Enter Student Id to Check result: ");

int in=sc.nextInt();

for(Employee e:dlst)

{

if(e.getEmpId()==(in))

{

dlst.stream().filter(emp->emp.getEmpId()==in).forEach(System.out::println);

System.exit(0);

}

}

System.out.println("No such data found");

break;

case "X":

System.exit(1);

default:

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

break;

}

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

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

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

System.out.println("Bye");

System.exit(1);

}

@Override

public String toString() {

return "MyClass []";

}

}

class Employee {

private int empId;

private String empNm;

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

private String r;

private float per;

private int t;

Scanner sc = new Scanner(System.in);

public Employee() {

}

public Employee(int empId, String empNm, int m1, int m2, int m3, int m4, int m5, String r,int t, float per ) {

super();

this.empId = empId;

this.empNm = empNm;

this.m1 = m1;

this.m2 = m2;

this.m3 = m3;

this.m4 = m4;

this.m5 = m5;

this.per=per;

this.r=r;

this.t=t;

}

public int getEmpId() {

return empId;

}

public void setEmpId(int empId) {

this.empId = empId;

}

public String getEmpNm() {

return empNm;

}

public void setEmpNm(String empNm) {

this.empNm = empNm;

}

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 getR() {

return r;

}

public void setR(String r) {

this.r = r;

}

public float getPer() {

return per;

}

public void setPer(float per) {

this.per = per;

}

public int getT() {

return t;

}

public void setT(int t) {

this.t = t;

}

@Override

public String toString() {

return "Student Result {Id=" + empId + ", name=" + empNm + ",marks=Subject Marks{ Hindi=" + m1 + ", English=" + m2 + ", maths=" + m3 + ", Science="

+ m4 + ", Social=" + m5 + "}, result=" + r +", total=" + t + ", percentage=" + per + "}";

}

}

* + **.Class File of assignment:**



------------------------------------------------------------------------------------------------

1. what is garbage collector and how it works

Garbage collection in Java is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. 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.

#### What is Garbage Collection?

In C/C++, a programmer is responsible for both the creation and destruction of objects. Usually, programmer neglects the destruction of useless objects. Due to this negligence, at a certain point, sufficient memory may not be available to create new objects, and the entire program will terminate abnormally, causing **OutOfMemoryErrors**.

But in Java, the programmer need not care for all those objects which are no longer in use. Garbage collector destroys these objects. The main objective of Garbage Collector is to free heap memory by destroying **unreachable objects**. The garbage collector is the best example of the Daemon thread as it is always running in the background.

#### How Does Garbage Collection in Java works?

Java garbage collection is an automatic process. Automatic garbage collection is the process of looking at heap memory, identifying which objects are in use and which are not, and deleting the unused objects. An in-use object, or a referenced object, means that some part of your program still maintains a pointer to that object. An unused or unreferenced object is no longer referenced by any part of your program. So the memory used by an unreferenced object can be reclaimed. The programmer does not need to mark objects to be deleted explicitly. The garbage collection implementation lives in the JVM.

1. what is heap space

**The Java heap 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. The size of the heap can vary, so many users restrict the Java heap size to 2-8 GB in order to minimize garbage collection pauses.

1. what is metaspace

Metaspace is a new memory space – starting from the Java 8 version; it has replaced the older PermGen memory space. The most significant difference is how it handles memory allocation.

Specifically, this native memory region grows automatically by default

1. ​ what is young and old generations

The Young Generation is where all new objects are allocated and aged. When the young generation fills up, this causes a minor garbage collection. A young generation full of dead objects is collected very quickly. Some survived objects are aged and eventually move to the old generation.

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

The heap memory is the runtime data area from which the Java VM allocates memory for all class instances and arrays. The heap may be of a fixed or variable size. The garbage collector is an automatic memory management system that reclaims heap memory for objects.

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

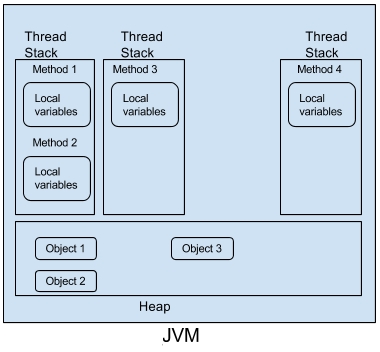
1. what is java memory model

Java memory model is divided between Thread Stacks (One for each thread) and a heap area.

Thread Stack: It is a thread specific memory area and contains local variables, methods call information etc. JVM stacks could be of fixed size or variable size. If computation in a thread exceeds its stack size limit then JVM throws StackOverflowError and exits.

* Heap

It contains all the objects created during application lifecycle. The heap is created when the virtual machine starts up. Garbage collector reclaims heap storage for objects and objects are never explicitly deallocated. The JVM is not using any automatic storage management system, and it may vary as per the system requirements. The heap may be of a fixed size or may vary as per requirement. The memory for the heap does not need to be contiguous.



Static variables are stored on heap area and object stored on the heap can be referred by references stored in thread stack.