**Capgemini Assignment-1**

1. **What is Metaspace and heap memory?**

The metaspace is the new memory space. It has been replaced by “permgen” space.

Dynamic allocation of native memory space during class & memory leak.

It is used to manage memory for class metadata .Class metadata are the runtime representation of java classes within a JVM process - basically any information the JVM needs to work with a Java class.

Heap memory is the main working memory, lowest address is the starting address. It's a chunk of memory allocated from the operating system by the memory manager in use by a process.

* The Heap memory is divided into many generations:

1. Young Generation.

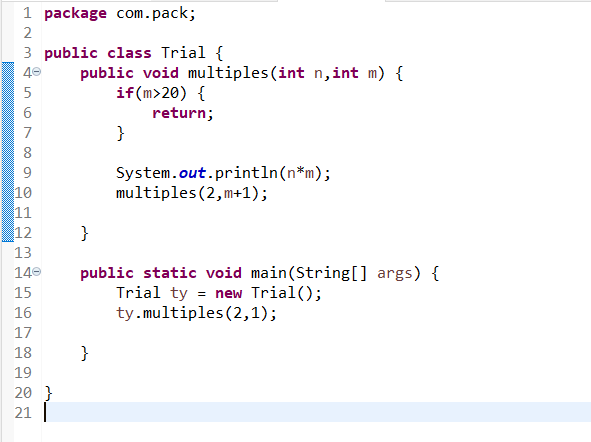
a.   Eden.

b.  Survivor (S0).

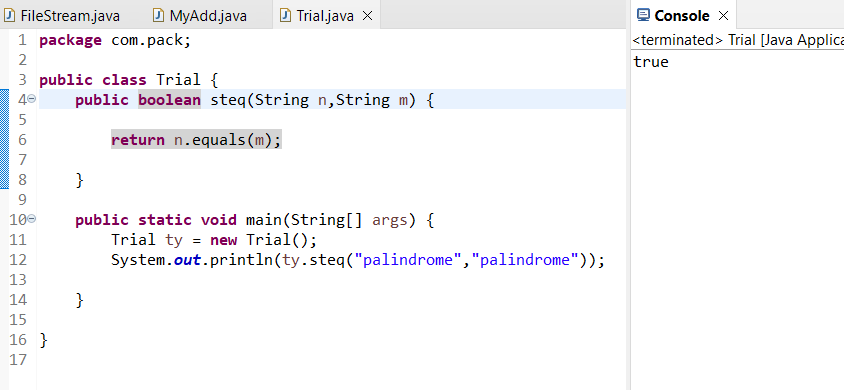
c.  Survivor (S1).

2. Old Generation (Tenured).  
3. Permanent generation.

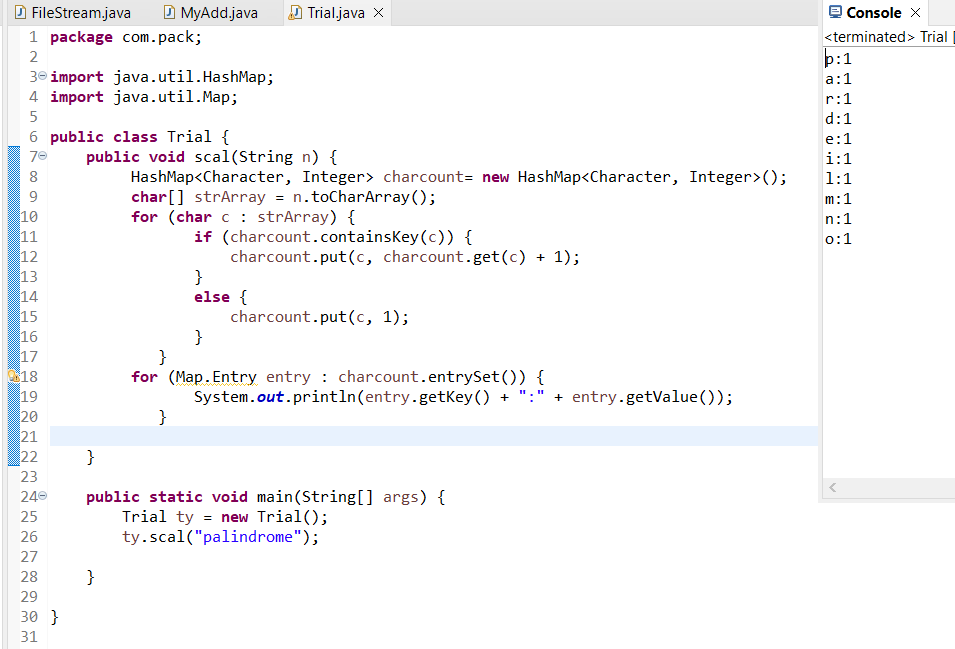
1. **Generate multiples of 2 until 20 using recursive function.**

* ****

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?**

Platform independent language means once compiled you can execute the program on any platform (OS). Java is platform independent. Because the Java compiler converts the source code to bytecode, which is Intermediate Language. Bytecode can be executed on any platform (OS) using Java Virtual Machine. the Java Virtual Machine is different for each platform; that is why it is known as platform-dependent. Using the Java Virtual Machine we can make the byte code understandable to any platform. That is why the byte code is known as platform-independent.

1. **Can we create class as final?**

Yes, we can make a class as final. A final class is a class that cannot be inherited or subclassed.

Once a class is declared final in Java, it is a sort of constant and it becomes immutable. Hence to make a class immutable, we have to declare it as final. A final variable cannot be reassigned another value A final method cannot be overridden

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

Program:

**package** com.pack;

**import** java.util.Arrays;

**import** java.util.Collection;

**import** java.util.Collections;

**import** java.util.List;

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

**int** empid;

String empname;

**int** empsalary;

**public** Salary(**int** empid, String empname, **int** empsalary) {

**this**.empid = empid;

**this**.empname = empname;

**this**.empsalary = empsalary;

}

**public** **int** getEmpid() {

**return** empid;

}

**public** **void** setEmpid(**int** empid) {

**this**.empid = empid;

}

**public** String getEmpname() {

**return** empname;

}

**public** **void** setEmpname(String empname) {

**this**.empname = empname;

}

**public** **int** getEmpsalary() {

**return** empsalary;

}

**public** **void** setEmpsalary(**int** empsalary) {

**this**.empsalary = empsalary;

}

@Override

**public** String toString() {

**return** "Salary [empid=" + empid + ", empname=" + empname + ", empsalary=" + empsalary + "]";

}

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

List<Salary> emp = Arrays.*asList*(

**new** Salary(1001,"Yoshita",10000),

**new** Salary(1002,"Akanksha",20000),

**new** Salary(1003,"Samiksha",30000),

**new** Salary(1004,"Rashmi",40000),

**new** Salary(1005,"Priyanka",50000)

);

Salary max = Collections.*max*(emp);

System.***out***.println("max salary employee : id=" + max.getEmpid() + " emp name= "+ max.getEmpname() + " salary = " + max.getEmpsalary());

}

@Override

**public** **int** compareTo(Salary s) {

**if** (**this**.getEmpsalary() > s.getEmpsalary()) {

**return** 1;

} **else** **if** (**this**.getEmpsalary() < s.getEmpsalary()) {

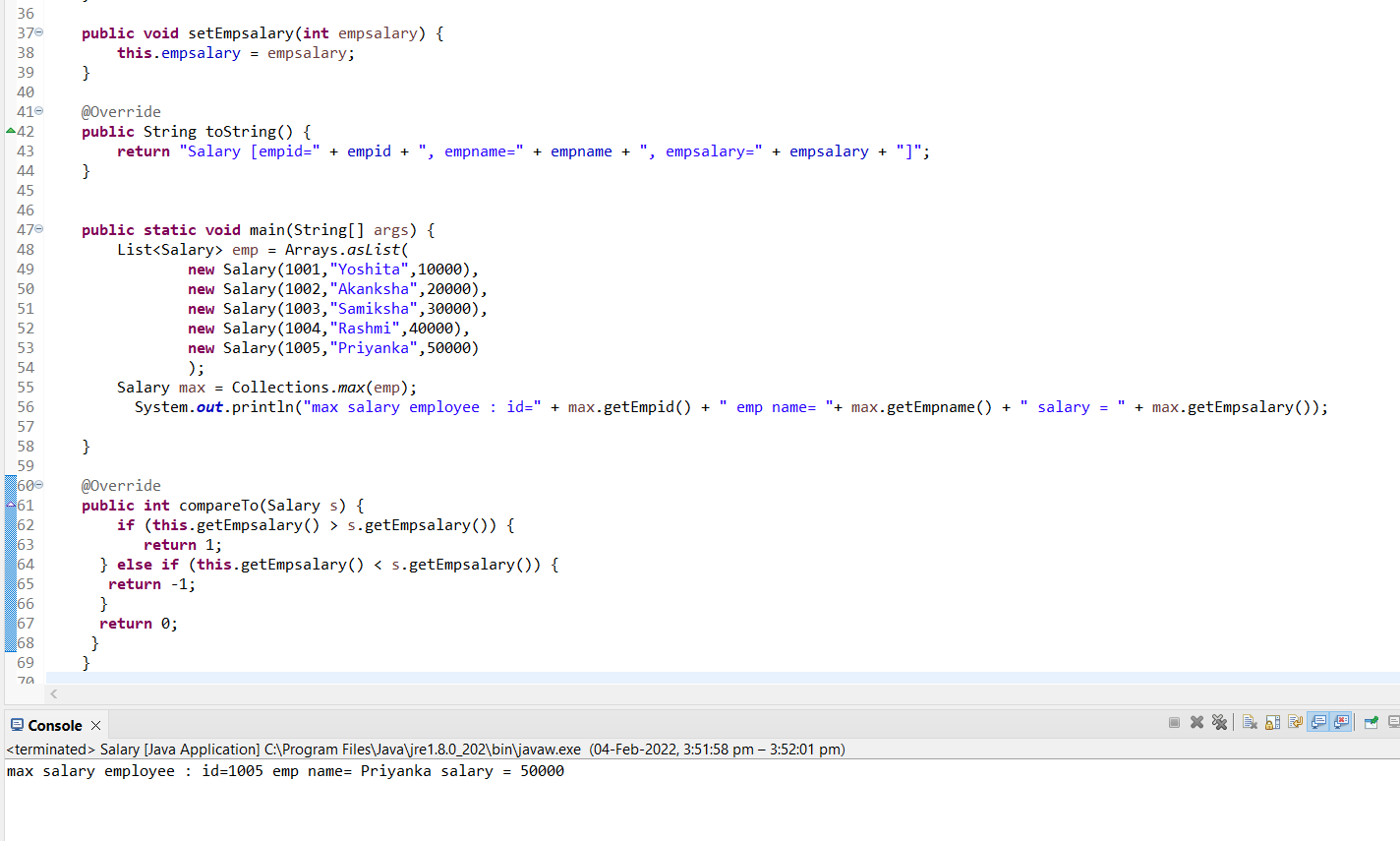
**return** -1;

}

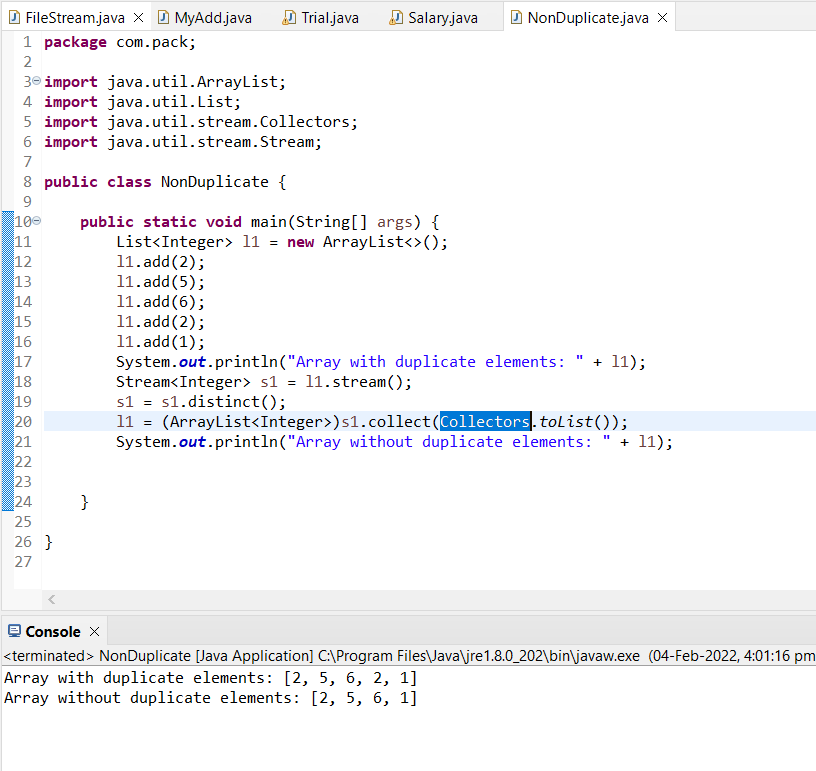
**return** 0;

}

}



1. **consider a list of duplicate values remove duplicate value and get unique values from the list**
2. **package** com.pack;
3. **import** java.util.ArrayList;
4. **import** java.util.List;
5. **import** java.util.stream.Collectors;
6. **import** java.util.stream.Stream;
7. **public** **class** NonDuplicate {
8. **public** **static** **void** main(String[] args) {
9. List<Integer> l1 = **new** ArrayList<>();
10. l1.add(2);
11. l1.add(5);
12. l1.add(6);
13. l1.add(2);
14. l1.add(1);
15. System.***out***.println("Array with duplicate elements: " + l1);
16. Stream<Integer> s1 = l1.stream();
17. s1 = s1.distinct();
18. l1 = (ArrayList<Integer>)s1.collect(Collectors.*toList*());
19. System.***out***.println("Array without duplicate elements: " + l1);
20. }
21. }



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

Yes we can write try and finally without catch.

The finally block is executed regardless of how the try block is written with catch or not. We can use try-finally block when we want to do nothing with the exception. if there is no catch block, the exception won't be handled.

1. **Create a java application for College Management.**
2. **package** com.pack;
3. **import** java.util.ArrayList;
4. **import** java.util.List;
5. **import** java.util.Scanner;
6. **import** java.util.stream.Collectors;
7. **public** **class** CollegeManagement {
9. **int** empId;
10. String empNm;
11. **int** m1, m2, m3, m4, m5;
12. String r;
13. **float** per;
14. **int** t;
15. Scanner sc = **new** Scanner(System.***in***);
17. **public** CollegeManagement(**int** empId, String empNm, **int** m1, **int** m2, **int** m3, **int** m4, **int** m5, String r, **float** per) {
18. **super**();
19. **this**.empId = empId;
20. **this**.empNm = empNm;
21. **this**.m1 = m1;
22. **this**.m2 = m2;
23. **this**.m3 = m3;
24. **this**.m4 = m4;
25. **this**.m5 = m5;
26. **this**.r = r;
27. **this**.per = per;
28. }
30. **public** **int** getEmpId() {
31. **return** empId;
32. }
33. **public** **void** setEmpId(**int** empId) {
34. **this**.empId = empId;
35. }
36. **public** String getEmpNm() {
37. **return** empNm;
38. }
39. **public** **void** setEmpNm(String empNm) {
40. **this**.empNm = empNm;
41. }
43. **public** **int** getM1() {
44. **return** m1;
45. }
46. **public** **void** setM1(**int** m1) {
47. **this**.m1 = m1;
48. }
49. **public** **int** getM2() {
50. **return** m2;
51. }
52. **public** **void** setM2(**int** m2) {
53. **this**.m2 = m2;
54. }
55. **public** **int** getM3() {
56. **return** m3;
57. }
58. **public** **void** setM3(**int** m3) {
59. **this**.m3 = m3;
60. }
61. **public** **int** getM4() {
62. **return** m4;
63. }
64. **public** **void** setM4(**int** m4) {
65. **this**.m4 = m4;
66. }
67. **public** **int** getM5() {
68. **return** m5;
69. }
70. **public** **void** setM5(**int** m5) {
71. **this**.m5 = m5;
72. }
73. **public** String getR() {
74. **return** r;
75. }
76. **public** **void** setR(String r) {
77. **this**.r = r;
78. }
79. **public** **float** getPer() {
80. **return** per;
81. }
82. **public** **void** setPer(**float** per) {
83. **this**.per = per;
84. }
86. @Override
87. **public** String toString() {
88. **return** "CollegeManagement [empId=" + empId + ", Name=" + empNm + ", Hindi=" + m1 + ", English=" + m2 + ", Maths=" + m3
89. + ", science=" + m4 + ", social=" + m5 + ", result=" + r + ", percent=" + per + "]";
90. }
91. **public** **static** **void** main(String[] args) {
92. **int** count = 0;
93. String result;
94. **char** choice;
95. List<CollegeManagement> stu = **new** ArrayList<>();
96. List<CollegeManagement> stu1 = stu.stream().distinct().collect(Collectors.*toList*());
97. System.***out***.println("Welcome to College Management");
99. **do** {
100. **if** (count > 0) {
101. System.***out***.println("C : Check Student Result");
102. }
103. System.***out***.println("A : Add Student Result");
104. System.***out***.println("X : Exit");
105. Scanner sc = **new** Scanner(System.***in***);
106. String c = sc.next();
108. **switch** (c.toUpperCase()) {
109. **case** "A":
110. System.***out***.println("Enter student id");
111. **int** id = sc.nextInt();
112. System.***out***.println("Enter student Name");
113. String name = sc.next();
114. System.***out***.println("Enter marks in hindi");
115. **int** hindi = sc.nextInt();
116. System.***out***.println("Enter marks in english");
117. **int** eng = sc.nextInt();
118. System.***out***.println("Enter marks in maths");
119. **int** maths = sc.nextInt();
120. System.***out***.println("Enter marks in science");
121. **int** science = sc.nextInt();
122. System.***out***.println("Enter marks in social");
123. **int** social = sc.nextInt();
124. **int** total = hindi+eng+maths+science+social;
125. **float** per = (**float**) total / 5;
126. per = per \* 100;
127. **if** (per > 50) {
128. result="Pass";
129. } **else** {
130. result="Fail";
131. }
132. stu1.add(**new** CollegeManagement (id, name,hindi,eng,maths,science,social,result,per));
133. System.***out***.println("Student added successfully");
134. count++;
135. **break**;
136. **case** "C":
137. System.***out***.println("Enter Student Id to Check result: ");
138. **int** in=sc.nextInt();
139. **for**(CollegeManagement e:stu1)
140. {
141. **if**(e.getEmpId()==(in))
142. {
143. stu1.stream().filter(emp->emp.getEmpId()==in).forEach(System.***out***::println);
144. System.*exit*(0);
145. }
146. }
147. System.***out***.println("No such data found");
148. **break**;
149. **case** "X":
150. System.*exit*(1);
151. **default**:
152. System.***out***.println("Invalid choice");
153. **break**;
154. }
155. System.***out***.println("Do you want to continue (Y/N)");
156. choice = sc.next().charAt(0);
157. } **while** (choice == 'Y' || choice == 'y');
158. System.***out***.println("Bye");
159. System.*exit*(1);
160. }
161. }

