1. How many male and female employees are there in the organization?

Map<String, Long> noOfMaleAndFemaleEmployees = employeeList.stream().collect(Collectors.groupingBy(Employee::getGender, Collectors.counting()));

System.out.println(noOfMaleAndFemaleEmployees);

1. Print the name of all departments in the organization?

employeeList.stream().map(Employee::getDepartment).distinct().forEach(System.out::println);

1. What is the average age of male and female employees?

Map<String, Double> averageAgeOfMaleAndFemaleEmp = employeeList.stream().collect(Collectors.groupingBy(Employee::getGender, Collectors.averagingInt(Employee::getAge)));

System.out.println(averageAgeOfMaleAndFemaleEmp);

1. Get the details of highest paid employee in the organization?

|  |
| --- |
| Optional<Employee> maxSal = employeeList.stream().collect(Collectors.maxBy(Comparator.comparingDouble(Employee::getSalary))); |
|  | System.out.println(maxSal.get()); |

1. Get the names of all employees who have joined after 2015?

employeeList.stream().filter(emp -> emp.getYearOfJoining() > 2015).map(Employee::getName).forEach(System.out::println);

1. Count the number of employees in each department?

Map<String, Long> noEmployeesInDept = employeeList.stream().collect(Collectors.groupingBy(Employee::getDepartment, Collectors.counting()));

Set<Map.Entry<String, Long>> entrySet = noEmployeesInDept.entrySet();

for (Map.Entry<String, Long> entry: entrySet){

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

}

1. What is the average salary of each department?

Map<String, Double > avgSalDept = employeeList.stream().collect(Collectors.groupingBy(Employee::getDepartment, Collectors.averagingDouble(Employee::getSalary)));

Set<Map.Entry<String, Double>> entries = avgSalDept.entrySet();

for (Map.Entry<String, Double> entry : entries)

{

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

}

1. Get the details of youngest male employee in the product development department?

Optional<Employee> youngestEmployee = employeeList.stream()

.filter(emp -> emp.getGender().equals("Male") && emp.getDepartment().equals("Product Development"))

.min(Comparator.comparing(Employee::getAge));

System.out.println(youngestEmployee);

1. Who has the most working experience in the organization?

Optional<Employee> mostWorkingExpEmp = employeeList.stream().sorted(lator.comparing(Employee::getYearOfJoining)).findFirst();

System.out.println(mostWorkingExpEmp.get());

1. How many male and female employees are there in the sales and marketing team?

Map<String, Long> maleAndFemaleInSaleMarketTeam = employeeList.stream()

.filter(emp -> emp.getDepartment().equals("Sales And Marketing"))

.collect(Collectors.groupingBy(Employee::getGender, Collectors.counting()));

System.out.println(maleAndFemaleInSaleMarketTeam);

1. What is the average salary of male and female employees?

Map<String, Double> avgSalaryOfMaleAndFemale = employeeList.stream()

.collect(Collectors.groupingBy(Employee::getGender, Collectors.averagingDouble(Employee::getSalary)));

System.out.println(avgSalaryOfMaleAndFemale);

1. List down the names of all employees in each department?

Map<String, List<Employee>> employeesDepartmentList = employeeList.stream()

.collect(Collectors.groupingBy(Employee::getDepartment));

Set<Map.Entry<String, List<Employee>>> entriess = employeesDepartmentList.entrySet();

for(Map.Entry<String, List<Employee>> entry : entriess)

{

System.out.println("Department : "+ entry.getKey());

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

for(Employee emp : entry.getValue())

{

System.out.println(emp.getName());

}

}

1. What is the average salary and total salary of the whole organization?

DoubleSummaryStatistics employeeLStats = employeeList.stream().collect(Collectors.summarizingDouble(Employee::getSalary));

System.out.println("Average Salary : "+employeeLStats.getAverage());

System.out.println("Total Salary : "+employeeLStats.getSum());

1. Separate the employees who are younger or equal to 25 years from those employees who are older than 25 years.

Map<Boolean, List<Employee>> partitionEmployees = employeeList.stream()

.collect(Collectors.partitioningBy( emp -> emp.getAge() > 25));

Set<Map.Entry<Boolean, List<Employee>>> entriesss = partitionEmployees.entrySet();

for (Map.Entry<Boolean, List<Employee>> entry: entriesss)

{

if(entry.getKey())

{

System.out.println("Employee older than 25 years");

}else {

System.out.println("Employee younger than or equals to 25 years");

}

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

for(Employee emp : entry.getValue())

{

System.out.println(emp.getName());

}

}

1. Who is the oldest employee in the organization? What is his age and which department he belongs to?

Optional<Employee> oldEmployee = employeeList.stream().max(Comparator.comparing(Employee::getAge));

System.out.println("Name: "+oldEmployee.get().getName());

System.out.println("Age: "+oldEmployee.get().getAge());

System.out.println("Department: "+oldEmployee.get().getDepartment());

1. Get the details of 2nd highest paid employee in the organization?
2. Get the details of 1st 3 highest paid employee in the organization?
3. Get the details of the highest salary in each department?
4. Get the details of 2nd highest-paid employee in the organization?

2. Get the details of 1st three highest-paid employees in the organization?

3. Get the details of the highest salary in each department?

Text

Description automatically generated

|  |
| --- |
| //Get students with a total score greater than 500 |
|  | students |
|  | .stream() |
|  | .filter(student -> student.getTotalScore() > 500) |
|  | .collect(Collectors.toList()); |
|  |  |
|  | //Get the 100 students having the highest total score |
|  | students |
|  | .stream() |
|  | .sorted( |
|  | Comparator.comparing(Student::getTotalScore, |
|  | Comparator.reverseOrder())) |
|  | .limit(100); |
|  |  |
|  | //Get students having a similar total score |
|  | Map<Integer, List<Student>> studentsWithSimilarMarks = students |
|  | .stream() |
|  | .collect( |
|  | Collectors.groupingBy(student -> student.getTotalScore())); |
|  |  |
|  | //Get passed and failed students |
|  | Map<Boolean, List<Student>> passingFailing = students |
|  | .stream() |
|  | .collect( |
|  | Collectors.partitioningBy( |
|  | student -> student.getPercentage() >= PASSING\_PERCENTAGE)); |
|  |  |
|  | //Get the student with the longest name |
|  | Optional<String> longestName = students |
|  | .stream() |
|  | .map(Student::getName) |
|  | .reduce((name1, name2) |
|  | -> name1.length() > name2.length() |
|  | ? name1 : name2); |
|  |  |

name | age | num of played | num of wins --------------------------------------------- \*/

**Player** kai = **new** **Player**( "Kai", 26, 28, 7);

**Player** eric = **new** **Player**( "Eric", 28, 30, 11);

**Player** saajan = **new** **Player**("Saajan", 30, 100, 66);

**Player** kevin = **new** **Player**( "Kevin", 24, 50, 49);

**String** kotlin = "Kotlin"; **String** java = "Java"; **String** python = "Python"; **String** ruby = "Ruby"; TreeSet<String> myTreeSet = Stream.of(ruby, java, kotlin, python).collect(Collectors.toCollection(TreeSet::**new**)); assertThat(myTreeSet).containsExactly(java, kotlin, python, ruby);

TreeSet<Player> myTreeSet = Stream.of(saajan, eric, kai, kevin) .collect(Collectors.toCollection(() -> **new** **TreeSet**<>(Comparator.comparingInt(Player::getNumberOfWins)) )); assertThat(myTreeSet).containsExactly(kai, eric, kevin, saajan);

TreeSet<Player> myTreeSet = Stream.of(saajan, eric, kai, kevin) .collect(Collectors.toCollection(() -> **new** **TreeSet**<>(Comparator.comparing(player -> BigDecimal.valueOf(player.getNumberOfWins()) .divide(BigDecimal.valueOf(player.getNumberOfPlayed()), 2, RoundingMode.HALF\_UP))))); assertThat(myTreeSet).containsExactly(kai, eric, saajan, kevin);