```
public class Employee {


private int id;

private String name;

private long salary;

private String role;

private int age;

@Override


public String toString() {

return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", role=" + role + ", age=" + age

+ ", year=" + year + ", gender=" + gender + "]";

}

private int year;

private char gender;

public Employee(int id, String name, long salary, String role, int age, int year, char gender) {

super();

this.id = id;

this.name = name;

this.salary = salary;

this.role = role;

this.age = age;

this.year = year;

this.gender = gender;

}

public int getId() {

return id;

}

public void setId(int id) {
```

```
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public long getSalary() {
return salary;
}
public void setSalary(long salary) {
this.salary = salary;
}
public String getRole() {
return role;
}
public void setRole(String role) {
this.role = role;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
public int getYear() {
return year;
}
public void setYear(int year) {
```

```
this.year = year;
}
public char getGender() {
return gender;
}
public void setGender(char gender) {
this.gender = gender;
}
```

} Question :1

```
package employeesJoinedSpecYear;

import java.util.ArrayList;
import java.util.List;
import java.util.stream.Collectors;

public class EmployeesjoinSpecyear {

public static void main(String[] args) {
// TODO Auto-generated method stub

List<Employee> employee = new ArrayList<Employee>();
Employee e1=new Employee(1,"devi",30000,"support",25,2020,'f');
Employee e2=new Employee(2,"hari",10000,"dev",35,2021,'m');
Employee e3=new Employee(3,"suri",45000,"analyst",28,2021,'m');
Employee e4=new Employee(4,"ramya",28000,"admin",32,2022,'f');
employee.add(e1);
employee.add(e2);
employee.add(e3);
```

```
employee.add(e4);
System.out.println(employee);

// Using Lambda expressions
//employee.stream().filter(e -> e.getYear() ==2022).forEach(e -> System.out.println(e));
List< Employee> emp =employee.stream().filter(e -> e.getYear()==
2021).collect(Collectors.toList());
System.out.println("employees having particular age using lambda expressions: "+emp);

// conventional method
for(Employee e: employee) {
if(e.getYear()==2021) {
System.out.println("employees having particular age using conventional method: "+e);
}
}
}
}
```

Question :2

```
public class EmployeesAgeGreaterThan30 {

public static void main(String[] args) {
// TODO Auto-generated method stub
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
```

```
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"admin",32,2022,'f'));
List<Employee> empl=emp.stream().filter(e->e.getAge()>30).collect(Collectors.toList());
System.out.println(empl);
```

Question: 3

```
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(5,"priya",52468,"QA",25,2020,'f'),
new Employee(6,"reshma",97485,"QA",35,2021,'m'),
new Employee(7,"sonali",6475,"dev",28,2021,'m'),
new Employee(8,"veda",825892,"admin",32,2022,'f'));

 Map<Boolean, List<Employee>> empl=emp.stream().collect(Collectors.partitioningBy(e->e.getAge()>30));
for(Map.Entry<Boolean, List<Employee>> map:empl.entrySet()) {
if(Boolean.TRUE.equals(map.getKey())) { // (map.getKey()==true)
System.out.println("employees having age greater than 30 are:"+ map.getValue());
}
else
System.out.println("employees having age less than 30 are:"+ map.getValue());
}
```

Question: 4

```
public class GroupEmployeesBygender {

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

```
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"admin",32,2022,'f'));

//Group employees by Gender
Map<Character,List<Employee>>
employee=emp.stream().collect(Collectors.groupingBy(Employee :: getGender));
System.out.println(employee);

    //Group employees by role
Map<String,List<Employee>> empl=emp.stream().collect(Collectors.groupingBy(Employee ::
getRole));
System.out.println(empl);
}
```

Question :5

```
public class HavingRolesGreaterThan2 {

public static void main(String[] args) {
// TODO Auto-generated method stub

List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
```

```
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"QA",32,2022,'f'));

Entry<String, Long>
empl=emp.stream().collect(Collectors.groupingBy(Employee::getRole,Collectors.counting())).en
trySet().stream().max(Map.Entry.comparingByValue()).get();

System.out.println(empl.getKey());

}
```

Question : 6

```
public class HighestSalaryOfEmployee {
public static void main(String[] args) {
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",10000,"support",25,2020,'f'),
new Employee(2,"hari",25000,"dev",35,2021,'m'),
new Employee(3,"suri",28000,"analyst",28,2021,'m'),
new Employee(4,"ramya",45000,"admin",32,2022,'f'));

//System.out.println(emp);
//Optional<Employee> highestSalary= emp.stream().max(Comparator.comparingLong(e ->
e.getSalary()));
//Optional<Employee> highestSalary=
emp.stream().sorted(Comparator.comparingLong(Employee :: getSalary).reversed()).findFirst();
Optional<Employee>highestSalary =
emp.parallelStream().collect(Collectors.maxBy(Comparator.comparingLong(Employee ::
getSalary)));
```

```
//highestSalary.ifPresent(employee -> System.out.println(employee));

highestSalary.ifPresentOrElse(e -> System.out.println(e),null);

Map<String, Object> empl=emp.stream().collect(Collectors.groupingBy(Employee::getRole,
Collectors.collectingAndThen(Collectors.toList(), list-
>list.stream().max(Comparator.comparingLong(Employee::getSalary)))));

System.out.println(empl);
```

Question :7

```
public class LowestSalary {


public static void main(String[] args) {

List<Employee> emp= Arrays.asList(

new Employee(1,"devi",5000,"support",25,2020,'f'),

new Employee(2,"hari",3000,"dev",35,2021,'m'),

new Employee(3,"suri",5468,"analyst",28,2021,'m'),

new Employee(4,"ramya",54789,"admin",32,2022,'f'));

Optional <Employee> lowestEmp=emp.stream().min((e1,e2)->(int)e1.getSalary()-
(int)e2.getSalary());

//Optional <Employee> lowestEmp=emp.stream().min(Comparator.comparingLong(Employee ::
getSalary));

//Optional <Employee>
lowestEmp=emp.stream().sorted(Comparator.comparingLong(Employee ::
getSalary)).findFirst();


lowestEmp.ifPresent(e -> System.out.println(e));
```

Question:8

```
public class NamesOfAllDepartments {


public static void main(String[] args) {

// TODO Auto-generated method stub

List<Employee> emp= Arrays.asList(
```

```
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"admin",32,2022,'f'));

emp.stream().map(e->e.getRole()).distinct().forEach(e-> System.out.println(e+" "));
System.out.println(emp.stream().count());
```

Question : 9

```
public class RoleHavingHighNumberOfEmployees {

public static void main(String[] args) {
// TODO Auto-generated method stub

List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"QA",32,2022,'f'));
Optional<Entry<String, Long>>
empl=emp.stream().collect(Collectors.groupingBy(Employee::getRole,Collectors.counting())).en
trySet().stream().max(Map.Entry.comparingByValue());
```

```
if(empl.isPresent()){
System.out.println(empl.get());
}
```

Question : 10

```
public class RolesGreaterThan3 {

public static void main(String[] args) {
// TODO Auto-generated method stub
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"QA",32,2022,'f'));
emp.stream().collect(Collectors.groupingBy(Employee::getRole,
Collectors.counting())).entrySet().stream().filter(e-
>e.getValue()>=2).forEach(System.out::println);
```

Question : 11

```
public class SecondHighest {

public static void main(String[] args) {
// TODO Auto-generated method stub
List<Employee> employee = new ArrayList<Employee>();
Employee e1=new Employee(1,"devi",30000,"support",25,2020,'f');
Employee e2=new Employee(2,"hari",10000,"dev",35,2021,'m');
```

```
Employee e3=new Employee(3,"suri",45000,"analyst",28,2021,'m');
Employee e4=new Employee(4,"ramya",28000,"admin",32,2022,'f');
employee.add(e1);
employee.add(e2);
employee.add(e3);
employee.add(e4);
System.out.println(employee);
Optional<Employee>
SecondHighest=employee.stream().sorted(Comparator.comparingLong(Employee::
getSalary).reversed()).skip(1).findFirst();
SecondHighest.ifPresent(e-> System.out.println(e));
```

Question : 12

```
public class SortingByNameAndAge {

public static void main(String[] args) {
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(5,"priya",52468,"QA",25,2020,'f'),
new Employee(6,"reshma",97485,"QA",35,2021,'m'),
new Employee(7,"sonali",6475,"dev",28,2021,'m'),
new Employee(8,"veda",825892,"QA",32,2022,'f'));
Comparator<Employee> emp1=Comparator.comparing(Employee:: getName);
Comparator<Employee> emp2=Comparator.comparing(Employee:: getAge);
emp.stream().sorted(emp1.thenComparing(emp2)).forEach(e->System.out.println(e));
```

Question : 13

```
class SortComparatorSalary implements Comparator<Employee>{

@Override
public int compare(Employee a, Employee b) {
// TODO Auto-generated method stub
return (int)a.getSalary()-(int)b.getSalary();
}

}
class SortComparatorName implements Comparator<Employee>{

@Override
public int compare(Employee a, Employee b) {
// TODO Auto-generated method stub
return a.getName().compareTo(b.getName());
}

}

public class SortingBySalaryConventionalMethod  {

public static void main(String[] args) {
// TODO Auto-generated method stub
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(5,"priya",52468,"QA",25,2020,'f'),
```

```
new Employee(6,"reshma",97485,"QA",35,2021,'m'),
new Employee(7,"sonali",6475,"dev",28,2021,'m'),
new Employee(8,"veda",825892,"QA",32,2022,'f'));

Collections.sort(emp,new SortComparatorSalary());
for(int i=0;i<emp.size();i++) {
System.out.println(emp.get(i));
}
Collections.sort(emp,new SortComparatorName());
System.out.println(emp);
```

Question : 14

```
public class YoungestFemaleEmployee {

public static void main(String[] args) {
 List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"admin",32,2022,'f'));

 Optional<Employee> empl = emp.stream().filter(e-
>e.getGender()=='m').min(Comparator.comparingInt(Employee::getAge));
 empl.ifPresent(System.out::println);
```

Question : 15

```
public class AvgAgeOfMaleAndFemaleEmployees {
```

```
  public static void main(String[] args) {
  List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"analyst",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"admin",32,2022,'f'));


  Map<Character, Double>
empl=emp.stream().collect(Collectors.groupingBy(Employee::getGender,Collectors.averagingL
ong(Employee::getAge)));
  System.out.println(empl);
```

Question :16

```
public class AvgSalaryAndTotalSalary {


public static void main(String[] args) {
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(5,"priya",52468,"QA",25,2020,'f'),
new Employee(6,"reshma",97485,"QA",35,2021,'m'),
new Employee(7,"sonali",6475,"dev",28,2021,'m'),
new Employee(8,"veda",825892,"QA",32,2022,'f'));
```

```
DoubleSummaryStatistics
empl2=emp.stream().collect(Collectors.summarizingDouble(Employee::getSalary));
System.out.println(empl2.getAverage());
System.out.println(empl2.getSum());
```

Question : 17

```
public class AvgSalaryOfRole {

public static void main(String[] args) {
List<Employee> emp= Arrays.asList(
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(5,"priya",52468,"QA",25,2020,'f'),
new Employee(6,"reshma",97485,"QA",35,2021,'m'),
new Employee(7,"sonali",6475,"dev",28,2021,'m'),
new Employee(8,"veda",825892,"QA",32,2022,'f'));

Optional<Entry<String, Double>>
empl2=emp.stream().collect(Collectors.groupingBy(Employee::getRole,Collectors.averagingLon
g(Employee::getSalary))).entrySet().stream().max(Map.Entry.comparingByValue());
    if(empl2.isPresent()) {
        System.out.println(empl2);
```

Question : 18

```
public class CountOfEmployeesByRole {

public static void main(String[] args) {
// TODO Auto-generated method stub
List<Employee> emp= Arrays.asList(
```

```
new Employee(1,"devi",5000,"support",25,2020,'f'),
new Employee(2,"hari",3000,"dev",35,2021,'m'),
new Employee(3,"suri",5468,"admin",28,2021,'m'),
new Employee(4,"ramya",54789,"admin",32,2022,'f'),
new Employee(1,"priya",52468,"QA",25,2020,'f'),
new Employee(2,"reshma",97485,"QA",35,2021,'m'),
new Employee(3,"sonali",6475,"dev",28,2021,'m'),
new Employee(4,"veda",825892,"QA",32,2022,'f'));
Map<String, Long>
empl=emp.stream().collect(Collectors.groupingBy(Employee::getRole,Collectors.counting()));
System.out.println(empl);
for(Map.Entry<String, Long> en : empl.entrySet()) {
System.out.println(en.getKey()+":"+en.getValue());
```