Sort using comparable

emp obj
id
name
salary

```
Class Employee implements
Comparable{

public int compareTo(
          Employee obj){
return   this.id - obj.id (Asc)
         obj.id - this.id (desc)
         this.name - obj.name(asc)
th
```

TreeSet → compareTo(obj)
String (ok)
StringBuffer (x)

Sort using comparator

```
class EmpComparator implements
                  comparator<Employee>{
public int compare(Employee e1,
                   Employee e2){
  if( e1.id < e2.id){
     return -1;
  }else if( e1.id > e2.id){
        return 1;
  }
  else{
     return 0;
  }
  return 0;
}
```

```
Collections.sort(empList, new comparatorObj());
list.sort(new ComparatorObj());
```

Sort by ternary operator

→ condition ? v1 : v2
T return v1
F return v2

```
return  e1.id < e2.id ? -1 : e1.id > e2.id ? 1 ; 0 ;
```

If two name is equal then sort the employee using salary in desc

```
int  compare(Emp e1, Emp e2){
     if(e1.name.equals(e2.name)){
     return  e1.salary < e2.salary ? 1 : e1.salary > e2.salary ? -1 : 0;
     }else{
return   e2.compareTo(e1);   // name descending order
  }
  return 0;
}
```

using stream

overriding compare method of comparator using lembda.

```
list.stream().sorted((o1,o2)→ o1.salary < o2.salary ? 1 : o1.salary > o2.salary ?
                                                     -1              -1 : 0).
                                                                     +1
                          collect(Collectors.toList());
```

it will return list of sorted employee using salary desc

ASC→