//  CompanyManagement.java

//=====================================================

import java.io.BufferedReader;

import java.io.File;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Map;

public class CompanyManagement {

/\*\*

\* list of employees

\*/

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

/\*\*

\* Constructor for CompanyManagement Initializes constructor method by calling

\* file read method

\*

\* @throws IOException

\* @throws FileNotFoundException

\*/

public CompanyManagement(String path, String path1) {

this.empList = getEmployeeFromFile(path, path1);

}

public ArrayList<Employee> getEmployeeFromFile(String path, String path1) {

ArrayList<Employee> eList = new ArrayList();

try {

BufferedReader br = new BufferedReader(new FileReader(path));

BufferedReader br1 = new BufferedReader(new FileReader(path1));

String line;

ArrayList<String> programmingLanguages;

HashMap<String, ArrayList<String>> mp = new HashMap();

while ((line = br1.readLine()) != null) {

String[] ls = line.split(",");

ArrayList<String> list = new ArrayList<String>(Arrays.asList(ls));

list.remove(0);

mp.put(ls[0], list);

}

while ((line = br.readLine()) != null) {

String[] ls = line.split(",");

if (ls[1].startsWith("D")) {

if (ls[5].compareToIgnoreCase("L") == 0) {

eList.add(new TeamLeader(ls[1], ls[2], Integer.parseInt(ls[7]), ls[3], mp.get(ls[2]),

Integer.parseInt(ls[4]), Double.parseDouble(ls[6])));

} else {

eList.add(new Developer(ls[1], ls[2], Integer.parseInt(ls[5]), ls[3], mp.get(ls[2]),

Integer.parseInt(ls[4])));

}

} else if (ls[1].startsWith("T")) {

eList.add(new Tester(ls[1], ls[2], Integer.parseInt(ls[5]), Double.parseDouble(ls[3]), ls[4]));

}

}

} catch (Exception e) {

e.printStackTrace();

}

return eList;

}

public ArrayList<Developer> getDeveloperByProgrammingLanguage(String pl) {

ArrayList<Developer> dList = new ArrayList();

for (Employee emp : this.empList) {

if (emp instanceof Developer && ((Developer) emp).getProgrammingLanguages().contains(pl)) {

dList.add((Developer) emp);

}

}

return dList;

}

public ArrayList<Tester> getTestersHaveSalaryGreaterThan(double value) {

ArrayList<Tester> tList = new ArrayList();

for (Employee emp : this.empList) {

if (emp instanceof Tester && ((Tester) emp).getSalary() > value) {

tList.add((Tester) emp);

}

}

return tList;

}

public ArrayList<Employee> getEmployeesHaveSalaryGreaterThan(double value) {

ArrayList<Employee> tList = new ArrayList();

for (Employee emp : this.empList) {

if (emp.getSalary() > value) {

tList.add(emp);

}

}

return tList;

}

public Employee getEmployeeWithHighestSalary() {

if (empList == null || empList.size() == 0) {

return null;

}

Employee retEmp = empList.get(0);

for (Employee emp : this.empList) {

if (retEmp.getSalary() < emp.getSalary())

retEmp = emp;

}

return retEmp;

}

public TeamLeader getLeaderWithMostEmployees() {

Map<String, ArrayList<Employee>> team = new HashMap();

Map<String, TeamLeader> tLeader = new HashMap();

ArrayList<Employee> list;

String teamName = "";

for (Employee emp : this.empList) {

if (emp instanceof TeamLeader) {

teamName = ((TeamLeader) emp).getTeamName();

tLeader.put(teamName, (TeamLeader) emp);

} else if (emp instanceof Developer) {

teamName = ((Developer) emp).getTeamName();

if (team.containsKey(teamName)) {

team.get(teamName).add(emp);

} else {

list = new ArrayList();

list.add(emp);

team.put(teamName, list);

}

}

}

int cnt = 0;

for (Map.Entry<String, ArrayList<Employee>> entry : team.entrySet()) {

if (entry.getValue().size() > cnt) {

cnt = entry.getValue().size();

teamName = entry.getKey();

}

}

return tLeader.get(teamName);

}

public ArrayList<Employee> sorted() {

ArrayList<Employee> sortedList = (ArrayList<Employee>) this.empList.clone();

Collections.sort(sortedList, new Comparator<Employee>() {

@Override

public int compare(Employee e1, Employee e2) {

if (e2.getSalary() - e1.getSalary() > 0)

return 1;

else if (e1.getSalary() == e2.getSalary()) {

String[] e1Name = e1.getEmpName().split(" ");

String[] e2Name = e2.getEmpName().split(" ");

return e1Name[e1Name.length].compareTo(e2Name[e2Name.length]);

} else

return -1;

}

});

return sortedList;

}

public void printEmpList() {

this.empList.forEach(employee -> {

System.out.println(employee.toString());

});

}

public void printEmpList(ArrayList<Employee> list) {

list.forEach(employee -> {

System.out.println(employee.toString());

});

}

public <E> boolean writeFile(String path, ArrayList<E> list) {

try {

File fileObj = new File(path);

fileObj.createNewFile();

FileWriter myWriter = new FileWriter(fileObj);

for (E employee : list) {

myWriter.write(employee.toString());

}

myWriter.close();

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

return false;

}

return true;

}

public boolean writeFile(String path, Employee employee) {

try {

File fileObj = new File(path);

fileObj.createNewFile();

FileWriter myWriter = new FileWriter(fileObj);

myWriter.write(employee.toString());

myWriter.close();

} catch (IOException e) {

System.out.println("An error occurred.");

e.printStackTrace();

return false;

}

return true;

}

}

//==============================================================================================

//Developer.java

//==========================================

/\*\*

 \*

 \*/

import java.util.ArrayList;

/\*\*

 \* @author PSL3

 \*

 \*/

public class Developer extends Employee {

private String teamName;

/\*\*

\* @return the teamName

\*/

public String getTeamName() {

return teamName;

}

/\*\*

\* @param teamName the teamName to set

\*/

public void setTeamName(String teamName) {

this.teamName = teamName;

}

/\*\*

\* @return the programmingLanguages

\*/

public ArrayList<String> getProgrammingLanguages() {

return programmingLanguages;

}

/\*\*

\* @param programmingLanguages the programmingLanguages to set

\*/

public void setProgrammingLanguages(ArrayList<String> programmingLanguages) {

this.programmingLanguages = programmingLanguages;

}

/\*\*

\* @return the expYear

\*/

public int getExpYear() {

return expYear;

}

/\*\*

\* @param expYear the expYear to set

\*/

public void setExpYear(int expYear) {

this.expYear = expYear;

}

private ArrayList<String> programmingLanguages;

private int expYear;

/\*\*

\* @param empID

\* @param empName

\* @param baseSal

\* @param teamName

\* @param programmingLanguages

\* @param expYear

\*/

public Developer(String empID, String empName, int baseSal, String teamName, ArrayList<String> programmingLanguages,

int expYear) {

super(empID, empName, baseSal);

this.teamName = teamName;

this.programmingLanguages = programmingLanguages;

this.expYear = expYear;

}

@Override

public double getSalary() {

if (getExpYear() >= 5) {

return getBaseSal() + getExpYear() \* 2000000;

} else if (5 > getExpYear() && getExpYear() >= 3) {

return getBaseSal() + getExpYear() \* 1000000;

} else {

return getBaseSal();

}

}

@Override

public String toString() {

return getEmpID() + "\_" + getEmpName() + "\_" + getBaseSal() + "\_" + getTeamName() + "\_"

+ getProgrammingLanguages() + "\_" + getExpYear();

}

}

//  Tester.java

//=====================================================

public class Tester extends Employee {

private double bonusRate;

private String type;

/\*\*

\* @return the bonusRate

\*/

public double getBonusRate() {

return bonusRate;

}

/\*\*

\* @param bonusRate the bonusRate to set

\*/

public void setBonusRate(double bonusRate) {

this.bonusRate = bonusRate;

}

/\*\*

\* @return the type

\*/

public String getType() {

return type;

}

/\*\*

\* @param type the type to set

\*/

public void setType(String type) {

this.type = type;

}

/\*\*

\* @param empID

\* @param empName

\* @param baseSal

\* @param bonusRate

\* @param type

\*/

public Tester(String empID, String empName, int baseSal, double bonusRate, String type) {

super(empID, empName, baseSal);

this.bonusRate = bonusRate;

this.type = type;

}

// Implemented getSalary method for Tester

@Override

public double getSalary() {

return getBaseSal() + getBonusRate() \* getBaseSal();

}

}

//  TeamLeader.java

//=====================================================

import java.util.ArrayList;

public class TeamLeader extends Developer {

private double bonus\_rate;

/\*\*

\* @return the bonus\_rate

\*/

public double getBonusRate() {

return bonus\_rate;

}

/\*\*

\* @param bonus\_rate the bonus\_rate to set

\*/

public void setBonusRate(double bonus\_rate) {

this.bonus\_rate = bonus\_rate;

}

/\*\*

\* @param empID

\* @param empName

\* @param baseSal

\* @param teamName

\* @param programmingLanguages

\* @param expYear

\*/

public TeamLeader(String empID, String empName, int baseSal, String teamName,

ArrayList<String> programmingLanguages, int expYear, double bonus\_rate) {

super(empID, empName, baseSal, teamName, programmingLanguages, expYear);

this.bonus\_rate = bonus\_rate;

}

// getSalary implementation for TeamLeader

@Override

public double getSalary() {

return super.getSalary() + super.getSalary() \* getBonusRate();

}

}

//Employee.java

//==========================================

/\*\*

 \* Abstract class Employee

 \*/

public abstract class Employee {

private String empID;

private String empName;

private int baseSal;

/\*\*

\* constructor for Employee

\*

\* @param empID

\* @param empName

\* @param baseSal

\*

\*/

Employee(String empID, String empName, int baseSal) {

this.empID = empID;

this.empName = empName;

this.baseSal = baseSal;

}

/\*\*

\* @return the employee ID

\*/

public String getEmpID() {

return empID;

}

/\*\*

\* @return the empName

\*/

public String getEmpName() {

return empName;

}

/\*\*

\* @param empName the empName to set

\*/

public void setEmpName(String empName) {

this.empName = empName;

}

/\*\*

\* @return the baseSal

\*/

public int getBaseSal() {

return baseSal;

}

/\*\*

\* @param baseSal the baseSal to set

\*/

public void setBaseSal(int baseSal) {

this.baseSal = baseSal;

}

// Abstract getSalary method for abstract class employee

public abstract double getSalary();

/\*\*

\* Overrride default toString method for class Employee

\*/

@Override

public String toString() {

return getEmpID() + "\_" + getEmpName() + "\_" + getBaseSal();

}

}

//  TestCM.java

//=====================================================

import java.io.File;

import java.util.ArrayList;

import java.util.Scanner;

public class TestCM {

/\*\*

\* @param args

\*/

public static void main(String[] args) {

StringBuilder sb = new StringBuilder();

sb.append("\n1.Read all Employees and print to screen");

sb.append("\n2.Show staff proficient in a Programming Language");

sb.append("\n3.Show Tester has a height salary");

sb.append("\n4.Show Employee's highest salary");

sb.append("\n5.Show Leader of the Team has most Employees");

sb.append("\n6.Sort Employees as descending salary");

sb.append("\n7.Write file");

sb.append("\n8.Exit");

sb.append("\nYour option from 1 - 8 : ");

int choice = 0;

Scanner sc = new Scanner(System.in);

CompanyManagement cm = new CompanyManagement(System.getProperty("user.dir") + "/src/input/ListOfEmployees.txt",

System.getProperty("user.dir") + "/src/input/PLInfo.txt");

ArrayList<Employee> empList;

while (choice != 8) {

System.out.println(sb.toString());

choice = sc.nextInt();

switch (choice) {

case 1:

empList = cm.getEmployeeFromFile(System.getProperty("user.dir") + "/src/input/ListOfEmployees.txt",

System.getProperty("user.dir") + "/src/input/PLInfo.txt");

empList.forEach(employee -> {

System.out.println(employee.toString());

});

break;

case 2:

System.out.println("Input Programming Language: ");

ArrayList<Developer> dList = cm.getDeveloperByProgrammingLanguage(sc.next());

dList.forEach(employee -> {

System.out.println(employee.toString());

});

break;

case 3:

System.out.println("Input Salary: ");

ArrayList<Tester> tList = cm.getTestersHaveSalaryGreaterThan(sc.nextDouble());

tList.forEach(employee -> {

System.out.println(employee.toString());

});

break;

case 4:

System.out.println(cm.getEmployeeWithHighestSalary());

break;

case 5:

System.out.println(cm.getLeaderWithMostEmployees());

break;

case 6:

empList = cm.sorted();

empList.forEach(employee -> {

System.out.println(employee.toString());

});

break;

case 7:

File directory = new File(System.getProperty("user.dir") + "/src/output");

if (!directory.exists()) {

directory.mkdir();

}

cm.writeFile(System.getProperty("user.dir") + "/src/output/Req2.txt",

cm.getDeveloperByProgrammingLanguage("C++"));

cm.writeFile(System.getProperty("user.dir") + "/src/output/Req3.txt",

cm.getEmployeesHaveSalaryGreaterThan(4700000));

break;

default:

break;

}

}

}

}

ListofEmployees.txt

1,D01,Nguyen Dinh Minh Khoi,Run,1,5000000

2,D02,Pham Le Anh Khoa,Fly,3,6000000

6,D04,To Quoc Bao,Walk,3,L,0.3,10000000

3,T01,Vu Bao Dang Khoa,0.2,AT,3000000

4,T01,Truong Pham Thao Mi,0,MT,2000000

PLInfo.txt

Nguyen Dinh Minh Khoi,C,C#,C++

Pham Le Anh Khoa,C,Java

To Quoc Bao,C

Vu Bao Dang Khoa,C++

Truong Pham Thao Mi,Java

Directory Strucutre:

Chegg
src
(default package)
> CompanyManag
> Developerjava
> Employee.java
> > TeamLeader.java
> TestCM.java
> Tester.java
Em

Output :

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

1

D01\_Nguyen Dinh Minh Khoi\_5000000\_Run\_[C, C#, C++]\_1

D02\_Pham Le Anh Khoa\_6000000\_Fly\_[C, Java]\_3

D03\_Lorene lee\_5000000\_Walk\_[Java]\_2

D04\_To Quoc Bao\_10000000\_Walk\_[C]\_3

T01\_Vu Bao Dang Khoa\_3000000

T01\_Truong Pham Thao Mi\_2000000

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

2

Input Programming Language:

Java

D02\_Pham Le Anh Khoa\_6000000\_Fly\_[C, Java]\_3

D03\_Lorene lee\_5000000\_Walk\_[Java]\_2

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

3

Input Salary:

1000000

T01\_Vu Bao Dang Khoa\_3000000

T01\_Truong Pham Thao Mi\_2000000

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

4

D04\_To Quoc Bao\_10000000\_Walk\_[C]\_3

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

5

D04\_To Quoc Bao\_10000000\_Walk\_[C]\_3

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

6

D04\_To Quoc Bao\_10000000\_Walk\_[C]\_3

D02\_Pham Le Anh Khoa\_6000000\_Fly\_[C, Java]\_3

D01\_Nguyen Dinh Minh Khoi\_5000000\_Run\_[C, C#, C++]\_1

D03\_Lorene lee\_5000000\_Walk\_[Java]\_2

T01\_Vu Bao Dang Khoa\_3000000

T01\_Truong Pham Thao Mi\_2000000

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

7

1.Read all Employees and print to screen

2.Show staff proficient in a Programming Language

3.Show Tester has a height salary

4.Show Employee's highest salary

5.Show Leader of the Team has most Employees

6.Sort Employees as descending salary

7.Write file

8.Exit

Your option from 1 - 8 :

Please find the attached , input files and sample output.

I have created sample input data for testing , you can try with actual input data.

If you have any queries or need any changes , feel free to put comments.