/\*\*

\*

\* @author Steven Glasford

\*

\* @version January 17, 2019

\*

\* A main class to run the program

\*/

//enable the use of the keyboard

import java.util.Scanner;

public class Client {

/\*\*

\* @param args No command line arguments are needed

\*/

public static void main(String[] args) {

//The size of the array and the maximum number of employees contained

//the array

int empNum = 10;

//A variable used often in loops

int i = 0;

//The initialized value for all of the salaried employees

int salary = 0;

//The number of salaried employees

int numSalary = 0;

//the number of hourly employees

int numHourly = 0;

//a conditional used to determine if a value is the correct input

boolean con = false;

//a temporary value used occasionally in different locations

boolean temp = false;

//used to determine whether an employee is a salaried or hourly employee

String payable = null;

//the title of a person, used if the person is salaried

String title = null;

//the rate of which an hourly employee is paid. only used for

//hourly employees

double hourlyRate = 0;

//The position of the employee, used only for an hourly employee

String position = null;

//the id number for an employee, used for both salaried and

//hourly employees

int id = 0;

//the name of the employee, used in both hourly and slaried employees

String name = null;

//used to ask the user if they want to add another employee

String question = null;

//initialize the employee array

Employee[] employeeList = new Employee[empNum];

//initialize the ability to use the keyboard

Scanner keyboard = new Scanner(System.in);

//use a do loop to ask about the employee, use a do-while loop to ask

//for at least one employee

do {

//reset the payable to null, in case it has changed

payable = null;

//reset temp to false, incase it has changed from the previous loop

temp = false;

System.out.println( "Is the employee Salaried (Y/N): " );

//keep asking the user for an acceptable answer

while (temp == false){

payable = keyboard.nextLine();

if ((payable.equals("y") || payable.equals("Y") ||

payable.equals("N") || payable.equals("n"))){

//exit the loop if the answer is acceptable

temp = true;

}

}

//reset id to zero if changed from the previous loop

id = 0;

System.out.println("What is the employee's ID: ");

//convert the entered number into an integer

id = Integer.parseInt(keyboard.nextLine());

//reset name if changed from the previous loop

name = null;

System.out.println("What is the employee's name: ");

//save the name

name = keyboard.nextLine();

//reset values to standard if they have changed from the

//previous loop.

salary = 0;

title = null;

hourlyRate = 0;

position = null;

//if the payable answer was "y", then person is salaried and asks

//for data specific to the salaried class

if (payable.equals("y") || payable.equals("Y")){

System.out.println("What is the Employee's Salary");

salary = Integer.parseInt(keyboard.nextLine());

System.out.println("What is the Employee's Title: ");

title = keyboard.nextLine();

//save the information gained into the employee list

employeeList[i] = new Salaried(id, name, title, salary);

}

else {

System.out.println("What is the Employee's hourly rate");

hourlyRate = Double.parseDouble(keyboard.nextLine());

System.out.println("What is the Employee's position: ");

position = keyboard.nextLine();

//save gained information into the employee list

employeeList[i] = new Hourly(id, name, position, hourlyRate);

}

System.out.println("Do you want to add another employee? (Y/N)");

//question = keyboard.nextLine();

//reset temp to default value

temp = false;

//keep asking the user if they want to add another until they have

//entered an acceptable answer

while (temp == false){

question = keyboard.nextLine();

if ((question.equals("y") || question.equals("Y")

|| question.equals("N") || question.equals("n"))){

temp = true;

}

else

temp = false;

}

//if the person wants to continue adding employees the con variable

//changes to true, other wise it is false, and the loop is not

//allowed to continue

if ((question.equals("n") || question.equals("N")))

con = false;

else

con = true;

//add to the i variable to make sure that the loop doesn't overstep

//the bounds of the array

i++;

//prevent array overstepping, and abide by user commands

} while ((con == true) && (i < 10));

//print out the entire employee array, regardless of actual number

//of employees, meaning null will be printed for unfilled array members

for (i = 0; i < empNum; i++)

System.out.println(employeeList[i]);

//Give everyone a 10% raise to their salary and their hourly rate

for (i=0; i < empNum; i++){

//apply raises for hourly employees

if (employeeList[i] instanceof Hourly){

Hourly h = (Hourly) employeeList[i];

//apply the raise

h.raise();

//save the number of hourly employees

numHourly = h.hourlyNum();

}

//apply raises for salaried employees

else if (employeeList[i] instanceof Salaried) {

Salaried s = (Salaried) employeeList[i];

//apply raises

s.raise();

//save the number of salaried users

numSalary = s.salaryNum();

}

}

//save the total number of employees found

empNum = employeeList[0].empNum();

//print out new information with the raises, and only print

//out the parts of the filled array, aka no nulls

for (i = 0; i < empNum; i++)

System.out.println(employeeList[i]);

//print some of the final information, including the total number of

//employees in the array, the number of salaried and hourly employees

System.out.println("There are a total of " + empNum + " employees. "

+ numSalary + " are salaried, and " + numHourly

+ " are hourly employees.");

}

}

/\*\*

\*

\* @author Steven Glasford

\* @version January 17, 2019

\*

\* The Employee class contains information about an employee

\*

\*/

public class Employee {

private int id = 0;

private String name = null;

private static int empCount = 0;

/\*\*

\* The constructor of the employee class

\* @param id The ID number of the employee

\* @param name The name of the employee

\*/

public Employee(int id, String name){

this.id=id;

this.name=name;

//Count the number of employees

empCount++;

}

/\*\*

\*

\* @return Return the ID number of the employee

\*/

public int getId( ){

return id;

}

/\*\*

\*

\* @return Return the name of the employee

\*/

public String getName( ) {

return name;

}

/\*\*

\*

\* @param o A nondescript object

\* @return Whether or not the object o is an instance of the class employee

\*/

public boolean equals(Object o){

if (!(o instanceof Employee))

return false;

//cast the object o into the employee class if able

Employee e = (Employee) o;

return id == e.id

&& name.equals(e.name);

}

/\*\*

\*

\* @return Return the total number of employees

\*/

public int empNum(){

return empCount;

}

/\*\*

\*

\* @return Return the string of information for the employee, including

\* the name and the ID number of the user

\*/

public String toString(){

return getId() + " : " + getName() + " , " ;

}

}

/\*\*

\*

\* @author Steven Glasford

\*

\* @version January 17, 2019

\*

\* The Hourly Class contains the information about the employee

\* including the name position and hourly rate for an hourly worker

\*/

public class Hourly extends Employee {

private String position;

private double hourlyRate;

private static int hourlyCount;

/\*\*

\*

\* @param id The ID number of the employee

\* @param name The name of the employee

\* @param position The position held by the employee

\* @param hourlyRate The hourly rate of the employee

\*/

public Hourly(int id, String name, String position, double hourlyRate){

super (id, name);

this.position=position;

this.hourlyRate=hourlyRate;

hourlyCount++;

}

/\*\*

\* apply a raise

\* @return false if the employee's income doesn't make sense

\*/

public boolean raise(){

if (hourlyRate < 0)

return false;

//apply the raise

hourlyRate = (double) ((hourlyRate \* .10) + hourlyRate);

return true;

}

/\*\*

\*

\* @return the number of total hourly employees

\*/

public int hourlyNum(){

return hourlyCount;

}

/\*\*

\*

\* @return the string containing the position and hourly rate, as well as

\* information from the super class

\*/

public String toString(){

return super.toString() + " " + getClass().getName() + " @ " + position

+ " : $" + hourlyRate ;

}

}

/\*\*

\*

\* @author Steven Glasford

\*

\* @version January 17, 2019

\*

\* The Salaried class contains information about all of the salaried employees

\* and is a subclass of the superclass of Employee. Each salaried employee has

\* a title and a salary

\*/

public class Salaried extends Employee {

private String title;

public int salary;

private static int salariedCount;

/\*\*

\*

\* @param id The Employee's ID number

\* @param name The name of the Employee

\* @param title The title held by the salaried employee

\* @param salary The salary of the employee

\*/

public Salaried(int id, String name, String title, int salary){

super(id, name);

this.title=title;

this.salary=salary;

salariedCount++;

}

/\*\*

\*

\* @return Return the number of salaried employees

\*/

public int salaryNum(){

return salariedCount;

}

/\*\*

\*

\* @param o a nondescript object

\* @return return whether the object is an instance of a salaried class

\*/

public boolean equals(Object o){

if (!(o instanceof Salaried))

return false;

Salaried s = (Salaried) o;

return true;

}

/\*\*

\* Apply the raise for the salaried employee

\*

\* @return whether or not the employee's salary makes sense (aka makes

\* sure the employee makes a positive income)

\*/

public boolean raise(){

if (salary < 0)

return false;

salary = (int) (((double) salary \* .10) + (double) salary);

return true;

}

/\*\*

\*

\* @return a string containing information from the employee class

\* as well as title and salary information

\*/

public String toString(){

return super.toString() + " " + getClass().getName() + " @ " + title

+ " : $" + salary ;

}

}