**Exercise1:**



***Income*** interface***:***

* + Methods:
* ***calculateBasicIncome ()***:This method calculates and returns the basic income of a Role. The basic income of a role is calculated as follows:
  + ***PrincipalRole***:

*Basic income = basic salary + 2000.*

* + ***SecondaryRole***:

*Basic income = hourly rate \* 10*.

* ***calculateExtraIncome(y: int*):** this method calculates and returns the extra income of a role. The extra income of a role is equal to the sum of cash of the *Awards* won by the role in the year ***y***.

***Award*** class***:***

* + Attributes:
    - ***name***: the name of the award.
    - ***awardYear:*** the year of the award.
    - ***cash:*** the amount of money obtained if the award is won.
  + Methods:
* ***Award (name: String, year: int, cash: double)***: constructor
* ***getAwardYear(*):** this method returns the year of the award.
* ***getCash(*):** this method returns the cash of the award.

***Role*** class

* + Attributes:
    - ***roleName***: the name of the role.
  + Methods:
* ***Role (name: String)***: constructor.
* ***Split(y: int, equalAwards: Award[], greaterAwards: Award[])***: this method splits the awards obtained by the role into two arrays. The Awards obtained in the year ***y*** are stored into the array ***equalAwards***. The Awards obtained later than the year ***y*** are stored into the array ***greaterAwards***.

***PrincipalRole*** class:

* + Attributes:
    - ***basicSalary***: the basic salary allocated for the role.
  + Methods:
* ***PrincipalRole (name: String, basicSalary: double):*** constructor.
* ***getBasicSalary():*** This method returns the basic salary of the Principal Role.

***SecondaryRole*** class:

* + Attributes:
    - ***hourlyRate***: the hourly rate allocated for the role.
  + Methods:
* ***SecondaryRole (name: String, hourlyRate: double):*** constructor.
* ***getHourlyRate():*** This method returns the hourly rate of the Secondary Role.

**QUESTION**: Translate into Java code the following:

* The interface ***Income.***
* The class ***Role,***
* The class ***PrincipalRole*** *(****do not implement*** *the method* ***getBasicSalary****)****.***

**Exercise 2:**

Let’s consider the same class ***Role*** described in exercise 1.



***Movie class***

* + Attributes:
    - ***name***: the name of the movie.
    - ***year:*** the year of the movie.
  + Methods:
* ***Movie (name: String, year: int, size: int)***: constructor.
* ***addRole(r: Role*):**this method adds the *Role* ***r*** to the Movie. It returns *true* if the *Role* ***r*** is successfully added. Otherwise, it returns *false*.
* ***addPrincipalRolesFromFile(fileName: String, basic: double, extra: double):*** this method reads from the file ***fileName*** objects of type ***Role***. It adds into the array ***arrRoles*** the *PrincipalRole* objects having a basic salary equal to the parameter ***basic*** and having an extra income equal to the parameter ***extra***.

**QUESTION**: Translate into Java code the class ***Movie*** *(do not implement the getters)****.***

**Exercise 3:**

Let’s consider the same class ***Movie*** described in exercise 2.



***LinkedListOfMovies class***

* + Methods:
* ***LinkedListOfMovies ()***: constructor.
* ***addMovieAtFirst(m: Movie*):** this method adds the *Movie* ***m*** at the beginning of the linked list.
* ***addMovieAtLast(m: Movie*):** this method adds the *Movie* ***m*** at the end of the linked list.
* ***getMovies(y: int)***: This method returns a linked list containing the movies that were produced in the year y. The movies of the obtained linked list are ordered in the inverse order than their original order.
* ***getLastMovie():*** this method returns the last movie in the linked list.
* ***insertMovieAfter(newM: Movie, name: String)***: This method inserts the movie ***newM*** after the first movie called ***name***.

**QUESTION**: Translate into Java code the class ***LinkedListOfMovies*** *(****do not implement*** *the methods* ***addMovieAtFirst*** *and* ***addMovieAtLast****)****.***

**public** **interface** Income { **……………… 1 ……………… total = 3**

**public** **double** calculateBasicIncome();**……………… 1**

**public** **double** calculateExtraIncome(**int** y); **……………… 1**

}

**public** **abstract** **class** Role **implements** Income { **……………… 1 + 1 ……………… total = 20**

**private** String roleName;

**private** Award arrAwards[];**……………… 1**

**private** **int** nbAwards; **……………… 1**

**public** Role(String name) { **……………… total = 2**

roleName = name;

arrAwards = **new** Award[10]; **……………… 1**

nbAwards = 0; **……………… 1**

}

**public** **double** calculateExtraIncome(**int** y) { **……………… total = 5**

**double** total = 0.0; **……………… 1**

**for** (**int** i = 0; i < nbAwards; i++) **……………… 1**

**if** (arrAwards[i].getAwardYear() == y) **……………… 1**

total += arrAwards[i].getCash();**……………… 1**

**return** total; **……………… 1**

}

**public** **void** split(**int** y, Award equalAw[], Award greaterAw[]) {**…………… total = 9**

**int** j = 0, k = 0; **……………… 1 + 1**

**for** (**int** i =0; i < nbAwards; i++) { **……………… 1**

**if** (arrAwards[i].getAwardYear() == y) **……………… 1**{

equalAw[j] = arrAwards[i]; **……………… 1**

j++; **……………… 1**

}

**else** {

**if** (arrAwards[i].getAwardYear() > y) { **……………… 1**

greaterAw[k] = arrAwards[i]; **……………… 1**

k++; **……………… 1**

}

}

}

}

}

**public** **class** PrincipalRole **extends** Role { **……………… 1 ……………… total = 5**

**private** **double** basicSalary;

**public** PrincipalRole(String name, **double** bs) { **……………… total = 2**

**super**(name); **……………… 1**

basicSalary = bs; **……………… 1**

}

**public** **double** calculateBasicIncome() { **……………… 1 ……………… total = 2**

**return** (basicSalary + 2000); **……………… 1**

}

}

**public** **class** Movie { **……………… total = 23**

**private** String name;

**private** **int** year;

**private** Role arrRoles[];**……………… 1**

**private** **int** nbRo; **……………… 1**

**public** Movie(String s, **int** y, **int** size) { **……………… total = 2**

name = s;

year = y;

arrRoles = **new** Role[size]; **……………… 1**

nbRo = 0; **……………… 1**

}

**public** **boolean** addRole(Role r) { **……………… total = 4**

**if** (nbRo < arrRoles.length) { **……………… 1**

arrRoles [nbRo] = r; **……………… 1**

nbRo ++; **……………… 1**

**return** **true**; **……………… 0.5**

}

**else**

**return** **false**; **……………… 0.5**

}

**public** **void** addPrincipalRolesFromFile(String fileName, **double** basic,

**double** extra) **throws** IOException { **……………… total = 13**

File f1 = **new** File(fileName); **……………… 1**

FileInputStream fo1 = **new** FileInputStream(f1); **……………… 1**

ObjectInputStream pf = **new** ObjectInputStream(fo1); **……………… 1**

Role r;

**try** { **……………… 1**

**while** (**true**) { **……………… 1**

**if** (r **instanceof** PrincipalRole) { **……………… 1**

**if** (((PrincipalRole)r).getBasicSalary()==basic **…1 + 1**

&& r.calculateExtraIncome(year) == extra ){ **…… 1**

arrRoles[nbRo] = r; **…… 1**

nbRo ++; **…… 1**

}

}

}

}

**catch**(EOFException eof) { **…… 1**

}

**catch**(ArrayIndexOutOfBoundsException e) { **…… 1**

}

**catch** (ClassNotFoundException e) {

}

pf.close();

}

**public** **void** addPrincipalRolesFromFileV2(String fileName, **double** basic,

**double** extra) **throws** IOException { **……………… total = 13**

File f1 = **new** File(fileName); **…… 1**

FileInputStream fo1 = **new** FileInputStream(f1); **…… 1**

ObjectInputStream pf = **new** ObjectInputStream(fo1); **…… 1**

Role r;

**try** { **…… 1**

**while** (**true**) { **…… 1**

**if** (r **instanceof** PrincipalRole) { **…… 1**

**if** (((PrincipalRole)r).getBasicSalary()==basic **…1 + 1**

&& r.calculateExtraIncome(year) == extra ) { **…… 1**

**if** (nbRo < arrRoles.length) { **…… 1**

arrRoles[nbRo] = r; **…… 1**

nbRo ++; **…… 1**

}

}

}

}

}

**catch**(EOFException eof) { **…… 1**

}

**catch** (ClassNotFoundException e) {

}

pf.close();

}

**public** **void** addPrincipalRolesFromFileV3(String fileName, **double** basic,

**double** extra) **throws** IOException { **……………… total = 13**

File f1 = **new** File(fileName); **…… 1**

FileInputStream fo1 = **new** FileInputStream(f1); **…… 1**

ObjectInputStream pf = **new** ObjectInputStream(fo1); **…… 1**

Role r;

**try** { **…… 1**

**while** (**true**) { **…… 1**

**if** (r **instanceof** PrincipalRole) { **…… 1**

**if** (((PrincipalRole)r).getBasicSalary()==basic **…1 + 1**

&& r.calculateExtraIncome(year) == extra ) { **…… 1**

addRole(r); **…… 3**

}

}

}

}

**catch**(EOFException eof) { **…… 1**

}

**catch** (ClassNotFoundException e) {

}

pf.close();

}

}

**public** **class** LinkedListOfMovies { **……………… total = 24**

**private** Node head; **…… 1**

**public** LinkedListOfMovies() {

head = **null**; **…… 1**

}

**public** LinkedListOfMovies getMovies(**int** y) { **…… total = 6**

LinkedListOfMovies res = **new** LinkedListOfMovies();**…… 1**

Node current = head; **…… 1**

**while** ( current != **null** ) { **…… 1**

**if** (current.getData().getYear() == y) { **…… 1**

res.addMovieAtFirst(current.getData());**…… 1**

}

}

**return** res; **…… 1**

}

**public** Movie getLastMovie() { **…… total = 7**

Node current = head; **…… 1**

**while** (current != **null** && current.getNext() != **null**) { **…… 1 + 1**

current = current.getNext();**…… 1**

}

**if** (current != **null**) **…… 1**

**return** current.getData();**…… 1**

**return** **null**; **…… 1**

}

**public** Movie getLastMovieV2() { **…… total = 7**

Node current = head; **…… 1**

**if** (head == **null**) **return** **null**; **…… 1 + 1**

**while** (current.getNext() != **null**) { **…… 1**

current = current.getNext(); **…… 1**

}

**return** current.getData();**…… 2**

}

**public** **boolean** insertAfter(Movie newM, String name) { **…… total = 9**

Node newNode = **new** Node(newM); **…… 1**

Node current = head; **…… 1**

**while** (current != **null** **…… 1**

&& !current.getData().getName().equals(name)) { **…… 1**

current = current.getNext();**…… 1**

}

**if** (current == **null**) **…… 1**

**return** **false**; **…… 0.5**

**else** {

newNode.setNext(current.getNext());**…… 1**

current.setNext(newNode); **…… 1**

**return** **true**; **…… 0.5**

}

}

}