

## Assignment

Course: **Object-Oriented Programming**

Course ID: **503005**

Subject: **Manage coffee shop module**

*(Students please carefully read all the instructions before starting the assignment)*

### I. Introduction

A coffee shop director needs to manage their products, customers, and orders. The staff of the shop is divided into 2 types: the full-time staff and the part-time staff. In this assignment, students will program some features to manage the business of the shop.

Features that students need to do:

- Read the staff list file.
- Based on the information from the file, perform queries based on requirements.

### II. Provided resources

Source code is provided by default, included in these files:

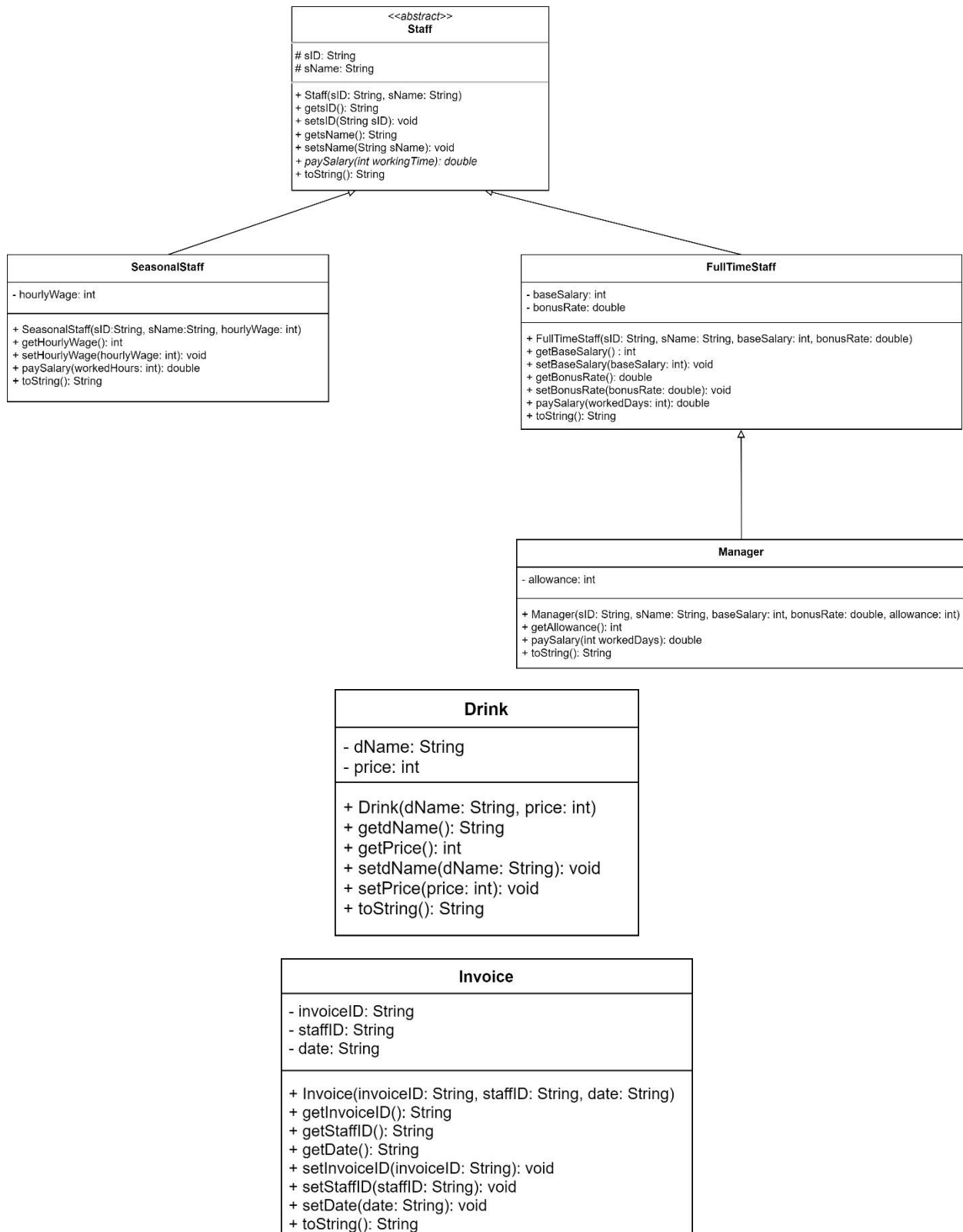
- Input file and expected output:
  - o Folder *input* includes 5 files:
    - *Drinks.txt*: contain the list of drinks
    - *InvoiceDetail.txt*: detail of the invoice
    - *Invoices.txt*: contain invoices information
    - *Staffs.txt*: contain staffs information
    - *Timekeeping.txt*: contain the timekeeping of the staff in a month
  - o Folder *output* includes: 5 files *Req1.txt*, *Req2.txt*, *Req3.txt*, *Req4.txt*, and *Req5.txt* contain the example output of the requirements in the assignment.
- Source code files:
  - o *Test.java*: create objects and call methods that students will define.
  - o *Staff.java*: contain predefined **Staff** class. Students must not edit this file.
  - o *Drink.java*, *Invoice.java*, *InvoiceDetails.java*: contain predefined classes to create corresponding objects. Students must not edit these files.
  - o *StoreManagement.java*: contain class **StoreManagement** with predefined property staffs to contain the list of students, other properties to contain other lists read from

files, constructors, read and write files methods, and some empty methods. Students will implement those empty methods and not edit predefined methods.

### III. The order to do the assignment

- Students download and extract provided resources.
- In the same directory with the 3 provided files, students create 3 files that correspond to these 3 classes **SeasonalStaff**, **FullTimeStaff**, and **Manager**.
- Students implement classes **SeasonalStaff**, **FullTimeStaff**, and **Manager** and add their implementations to the class **StoreManagement** based on the requirements in the following parts
- Implement the read file method to read **Staff** information to list *staffs*.
- Carefully read the provided list of **Drink**, **Invoice**, and **InvoiceDetail**, and combine these lists with the Staff list to implement the assignment requirements.
- After implementing the methods in the **StoreManagement** class, students compile and run the method **main** in the *Test.java* file. Students implement the requirements in the following parts and compare their results with the provided results in the output folder.
- For requirements that students can't implement, please do not remove methods that are related to those requirements and make sure that your program can run with the **main** method in the provided *Test.java* file.
- The Google Drive link will also contain a *version.txt* file. Students should usually check this file. If there are new changes, students should read the changes carefully and re-download the newest file(s). This *version.txt* file is used to notify what was changed and the date when it was changed in case of occurring errors in the assignment.

### IV. Classes description



InvoicesDetails
- invoiceID: String - dName: String - amount: int
+ InvoiceDetails(invoiceID: String, dName: String, amount: int) + getInvoiceID(): String + getDName(): String + getAmount(): int + setInvoiceID(invoiceID: String): void + setDName(dName: String): void + setAmount(amount: int): void + toString(): String

StoreManagement
- staffs: ArrayList<Staff> - workingHours: ArrayList<String> - invoices: ArrayList<Invoice> - invoiceDetails: ArrayList<InvoiceDetails> - drinks: ArrayList<Drink>
+ StoreManagement(staffPath: String, workingHoursPath: String, invoicesPath: String, detailsPath: String, drinkPath: String) + getStaffs(): ArrayList<Staff> + loadDrinks(filePath: String): ArrayList<Drink> + loadInvoices(filePath: String ): ArrayList<Invoice> + loadInvoicesDetails(filePath: String): ArrayList<InvoiceDetails> + loadStaffs(filePath: String): ArrayList<Staff> + getTopFiveSeasonalStaffsHighSalary(): ArrayList<SeasonalStaff> + getFullTimeStaffsHaveSalaryGreaterThan(lowerBound: int): ArrayList<FullTimeStaff> + getStaffHighestBillInMonth(int month): Staff + totalInQuarter(quarter: int): double + loadFile(filePath: String): ArrayList<String> + displayStaffs(): void + writeFile(path: String, list: ArrayList<E>): boolean + writeFile(path: String, object: E): boolean

- **SeasonalStaff** class and **FullTimeStaff** class inherits from **Staff** class, **Manager** class inherits **FullTimeStaff** class.
- Explanation of some of the properties and methods:
  - **Staff**
    - *sID*: staff ID.
    - *sName*: staff name.
    - Abstract method **paySalary(int workingTime)**.
    - **toString()**: return a string of the format: *sID\_sName*
  - **Drink**
    - *dName*: drink name
    - *price*: drink price
    - **toString()**: return a string of the format: *dName\_price*
  - **Invoice**
    - *invoiceID*: invoice ID
    - *staffID*: invoicing staff

- *date*: invoice creation date, save the string in the format dd/mm/yyyy
- **toString()**: return a string of the format: *invoiceID\_staffID\_date*
- **InvoiceDetails**
  - *invoiceID*: invoice ID
  - *dName*: drink name
  - *qty*: quantity of the drink in the invoice
  - **toString()**: return a string of the format: *invoiceID\_dName\_qty*
- **StoreManagement**
  - *staffs*: staffs list
  - *workingTime*: list of working times, this list is provided in the form of a list of String.
  - *invoices*: list of invoices, this list is provided in the form of a list of Invoice.
  - *invoiceDetails*: list of invoices details, this list is provided in the form of a list of InvoiceDetails.
  - *drinks*: list of drinks, this list is provided in the form of a list of Drink.
  - **StoreManagement(String staffPath, String workingTimePath, String invoicesPath, String detailsPath, String drinksPath)**: constructor, make the call to the read file methods to read information to corresponding attributes.
  - **getStaffs(), setStaffs(ArrayList<Staff> staffs)**: use to get Staff list and re-assign Staff list. These two methods are for marking the assignment. Students are not allowed to edit these methods.
  - **loadDrinks(String filePath), loadInvoices(String filePath), loadInvoicesDetails(String filePath)**: methods that are used to read files to lists. Students are not allowed to edit these methods.
  - **public static ArrayList<String> loadFile(String filePath)**: helper method to read information from files. Students can use this method in their implementation.
  - **displayStaffs()**: print the *staffs* list to the command prompt.
  - **writeFile(String path, ArrayList<E> list)**: write to a file the provided list. Students are not allowed to edit this method.
  - **writeFile(String path, E object)**: write a to file the provided object. Students are not allowed to edit this method.
- **FullTimeStaff – Official staff**
  - *baseSalary*: the base salary of the full-time staff
  - *bonusRate*: the bonus rate of the official staff
  - **paySalary(int workedDays)**: return the salary, which will be described in the parts below. With workedDays is the parameter of days that the staff worked in the month.

- **toString():** return a string of the format:

*sID\_sName\_bonusRate\_baseSalary*

- **SeasonalStaff – Part-time staff**

- *hourlyWage*: hourly wage
- **paySalary(int workedHours):** return the salary, which will be described in the parts below. With workedHours is the parameter of hours that the staff worked in the month.
- **toString():** return a string of the format:

*sID\_sName\_hourlyWage*

- **Lớp Manager – Manager:** The manager is an official staff.

- *allowance*: allowance
- **paySalary(int workedDays):** return the salary, which will be described in the parts below. With workedDays is the parameter of days that the staff worked in the month.
- **toString():** return a string of the format:

*sID\_sName\_bonusRate\_baseSalary\_allowance*

- Salary calculation description:

- **FullTimeStaff**

- Salary = Base salary \* bonus rate + **bonus**

**Bonus** calculation:

- If the staff has worked days (workedDays) <= 21: no bonus.
- If the staff has worked days (workedDays) > 21: from day 22 will get 100,000 for each working day.

For example, if an official staff has a base salary of 10,000,000, has a bonus rate of 1.2, and works 24 days a month then the total salary of this staff will be:

$$10,000,000 * 1.2 + (3 * 100,000)$$

- **Manager: FullTimeStaff** là **Manager** will have an additional allowance.

- Salary = FullTimeStaff salary + allowance

- **SeasonalStaff**

- Salary = hourly wage \* worked hours

With worked hours being workedHours.

## V. Input and output files description

- The input file named *Staffs.txt* contains the list of staff with each line corresponding to the properties of one staff separated by the sign “,” in the following format:

- **FullTimeStaff**

Staff ID,Staff name,Base salary,Bonus rate

CT004,Nguyen Thanh Huy,4000000,0.68

CT005,Tran Trung Nghia,3000000,5.89

- **Manager**

Staff ID,Staff name,Base salary,Bonus rate,Allowance

QL003,Nguyen Pham Minh Khoa,10000000,1.5,3000000

- **SeasonalStaff**

Staff ID,Staff name,Hourly wage

TV001,Cao Dang Khoa,12000

TV002,Tran Thanh Hung,25000

TV003,Nguyen Phung Nguyen,10000

- Input files named *Drinks.txt*, *InvoiceDetails.txt*, *Invoices.txt*, and *Timekeeping.txt* contain the corresponding list of items of each class, each line corresponds to the properties of each class separated by the sign “,” in the following format:

- **Drink**

Drink name,Drink price

Tra Sua Phuc Long,50000

- **Invoice**

Invoice ID,Staff ID,Invoice creation date

HD004,TV008,26/01/2022

HD005,CT001,25/03/2022

HD006,TV006,10/04/2022

With staff ID the same as the staff ID in the *staffs* list, the invoice creation date is stored in the format dd/mm/yyyy.

- **InvoiceDetails**

Invoice ID,Drink name ,Quantity

HD029, Latte Macchiato, 2  
HD001, Tra Dao, 1  
HD045, Ca Phe Sua Da, 3

With Invoice ID the same as the invoice ID in the *invoices* list, drink name the same as the drink name in the *drinks* list.

- **Timekeeping:** contains the timekeeping list of a month

Staff ID, Worked time

TV009, 50  
TV010, 8  
CT010, 16  
QL001, 18

If the staff is a full-time one, the worked time is the worked days, if the staff is a part-time one, the worked time is the worked hours.

- The output includes 5 files containing provided example output for the requirements:
  - *Req1.txt*: the result of staff lists read from the file.
  - *Req2.txt*: the result of five part-time staffs that have the highest salary in the month.
  - *Req3.txt*: the result of staffs that have salary > 15,000,000.
  - *Req4.txt*: the result of revenue in the first quarter.
  - *Req5.txt*: the result of staff having the highest number of invoices created in June.
- Each line in the output Req1, Req2, Req3, and Req5 corresponds to one object with the format **toString()** of the **corresponding** class.

**Note:**

- Students can add more data to the input file to test more cases but remember that the added data must follow the format that is defined above.
- Students should carefully read the **main** method to correctly define the order to implement classes and methods.
- Students can add code to the **main** method to test your implementation but make sure that your program **can run on the default provided by the main method**.
- Students can add new methods to help with your implementations but make sure that the student's program can still run with the provided *Test.java* file.
- Students must not make an absolute path when defining methods related to reading files. If students define an absolute path resulting in a cannot read a file during marking, it is equivalent to a compile error.



- Students **must not** in any circumstances edit **the class names and method names** that are defined initially (the naming always follows the class diagram above).
- Students do not make changes to files that are not required to be submitted.

## VI. Requirements

Students must not add additional libraries, only use libraries in provided files.

Students implement your assignment in Java 11 or Java 8. Students must not use the data type *var*. Students' implementation will be scored in Java 11; students will be responsible for all the error that happens if they were to use different Java versions.

### 1. REQUIREMENT 1 (2 POINTS)

Students implement the method **public ArrayList<Staff> loadStaffs(String filePath)** to read staffs from file *Staff.txt*, create **Staff** objects, and import them to attribute *staffs*.

Students implement the method, the statement to call the constructor method of **StoreManagement** in method **main** which will invoke to read file methods, if students correctly implement the read file method for attribute *staffs*, then the predefined write file method will write to the "Req1.txt". Students compare this file to the expected output file *Req1.txt* in the *output* folder.

**Note:** This is the method that students must be able to define to be eligible for scoring, if students can't read the file to be a list of **Staff** then the requirements below will not be scored.

**The requirements below are depending on the list which is read from the files.**

### 2. REQUIREMENT 2 (2 POINTS)

Implement method:

```
public ArrayList<SeasonalStaff> getTopFiveSeasonalStaffsHighSalary()
```

return a list of 05 part-time staffs with the highest salary in the list of staffs.

Students implement the method so that when invoking the **main** method, the correct output will be written to the "Req2.txt". Students compare this file to the expected output file *Req2.txt* in the *output* folder.

### 3. REQUIREMENT 3 (2 POINTS)

Implement method:

```
public ArrayList<FullTimeStaff> getFullTimeStaffsHaveSalaryGreaterThan(int  
lowerBound)
```

return a list of staff that has a higher salary than passed **lowerBound**. For example, provided **lowerBound** is 15000000 then the returned list will contain staffs that have salaries > 15,000,000. Note, the manager (Manager) is full-time staff.

Students implement the method so that when invoking the **main** method, the correct output will be written to the “Req3.txt”. Students compare this file to the expected output file *Req3.txt* in the *output* folder.

#### 4. REQUIREMENT 4 (2 POINTS)

Implement method:

**public double totalInQuarter(int quarter)**

return the total revenue of the passing **quarter**. Knowing that the first quarter includes the months 1, 2, 3, the second quarter includes the months 4, 5, 6, and the same for the third and fourth quarters. The total revenue is the total of invoices for months in that quarter. The total of one invoice is the total cost of drinks in the invoice. The total cost of a drink is the price of that drink multiplied by the quantity in the invoice.

Students implement the method so that when invoking the **main** method, the correct output will be written to the “Req4.txt”. Students compare this file to the expected output file *Req4.txt* in the *output* folder.

#### 5. REQUIREMENT 5 (2 POINTS)

Implement method:

**public Staff getStaffHighestBillInMonth(int month)**

























return the staff had the highest sum of the cost of the invoices in the **month**. The total cost of invoices of staff in the month is calculated by adding the total of each invoice that staff has made in the month.

Students implement the method so that when invoking the **main** method, the correct output will be written to the “Req5.txt”. Students compare this file to the expected output file *Req5.txt* in the *output* folder.

### VII. Check before submission

- If students can't implement some of the requirements please leave the methods related to those requirements intact. Students **MUST NOT DELETE THE METHOD(S) OF THE REQUIREMENTS** which will lead to errors when running the **main** method. Before submitting, students must check that the program is able to run with the provided **main** method.

- All of the files ReqX.txt ( $X = \{1,2,3,4,5\}$ ) are written at the same level as the source code directory. For students that use IDE (Eclipse, Netbeans, ...) make sure that the program can be run with commands in command prompt, make sure that the program does not exist inside a package, the position for written file ReqX.txt must be in the same level as the source code directory.
- Correct structure of written file when running the program will follow this image:

 input	08/05/2022 15:34	File folder	
 Drink.class	08/05/2022 16:25	CLASS File	2 KB
 Drink.java	30/04/2022 22:24	JAVA File	1 KB
 FullTimeStaff.class	08/05/2022 16:25	CLASS File	2 KB
 FullTimeStaff.java	03/05/2022 13:48	JAVA File	1 KB
 Invoice.class	08/05/2022 16:25	CLASS File	2 KB
 Invoice.java	28/04/2022 23:33	JAVA File	1 KB
 InvoiceDetails.class	08/05/2022 16:25	CLASS File	2 KB
 InvoiceDetails.java	30/04/2022 22:40	JAVA File	1 KB
 Manager.class	08/05/2022 16:25	CLASS File	1 KB
 Manager.java	03/05/2022 13:48	JAVA File	1 KB
 Req1.txt	08/05/2022 16:25	Text Document	1 KB
 Req2.txt	08/05/2022 16:25	Text Document	1 KB
 Req3.txt	08/05/2022 16:25	Text Document	1 KB
 Req4.txt	08/05/2022 16:25	Text Document	1 KB
 Req5.txt	08/05/2022 16:25	Text Document	1 KB
 SeasonalStaff.class	08/05/2022 16:25	CLASS File	2 KB
 SeasonalStaff.java	08/05/2022 15:35	JAVA File	1 KB
 Staff.class	08/05/2022 16:25	CLASS File	2 KB
 Staff.java	03/05/2022 13:42	JAVA File	1 KB
 StoreManagement.class	08/05/2022 16:25	CLASS File	9 KB
 StoreManagement.java	08/05/2022 16:20	JAVA File	12 KB
 Test.class	08/05/2022 16:25	CLASS File	2 KB
 Test.java	08/05/2022 16:20	JAVA File	1 KB

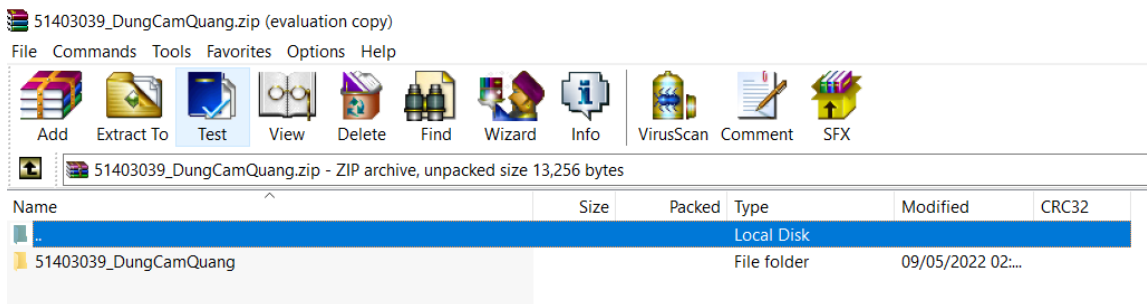
## VIII. Submission guideline

- When submitting, students submit *FullTimeStaff.java*, *Manager.java*, *SeasonalStaff.java*, and file *StoreManagement.java*, **do not include any other files, and must not edit the name of these files.**
- **Students place these 4 files into the folder named MSSV\_HoTen** (Full name is written with no spaces, no signs) compressed with the **.zip** format and submit by the guidance of the practical lecturer.

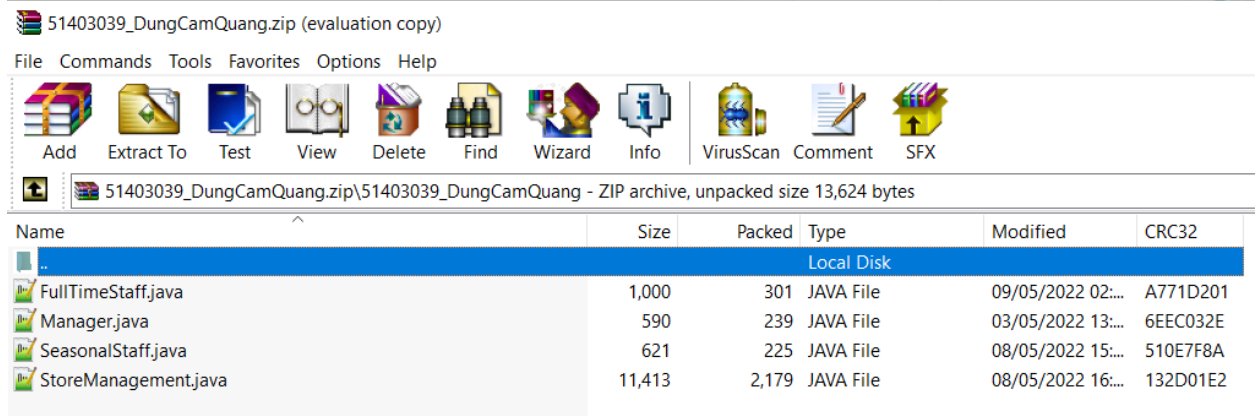
- In case of wrong submission (wrong folder naming, not placing the files into the folder, more files than required, ...) then students will get **0 point**.
- Correct file structure will be:
  - o Compressed file:

51403039\_DungCamQuang.zip 09/05/2022 02:33 WinRAR ZIP archive 4 KB

- o Inside the compressed file:



- o Inside the folder:



## IX. Scoring and regulations

- Your implementation will be scored automatically through testcases (input file and output file will have the format as described above) so students will be responsible for not abiding by the submission guideline or changing the method name which leads to the program not being able to compile.
- ***For all cases that students use absolute file path in the reading files process, the students will get 0 point.***
- Testcases which are used to score the program are files that have the same format as described with different content to the input files provided to students. Students will only receive points for the Requirement if your implementation output is exact completely.

- If the compilation of students' implementation results in an error, you will receive **0 point** for all requirements.
- **All of your code will be checked for plagiarism. Any behaviors of copying code from the Internet, copying your friend's code, or allowing your friends to copy your code if detected you will receive 0 point for the whole Process 2 point or not being able to participate in the final exam.**
- If students implementation have sign of copying code from the Internet or copying each other, students will be call for a code interview to prove that the implementation is indeed their.
- **Submission deadline: 23h00 of May 25<sup>th</sup>, 2022**

-- END --