COP3503 Project 3 – Work Order Generator

Objectives:

- Show an understanding of how to
    - Create class files that serve as templates for creating objects
    - Setup inheritance and implement interfaces
    - Read UML diagrams and implement the classes and methods outlined in the diagram

Submission Requirements:

- Submit your project via the Canvas course
    - Upload a zip file containing your 9 .java files specified in the UML diagram
    - The zip file should be named Project3_N#.zip
        - For Example: Project3_N00123456.zip
    - The zip file should contain the 9 .java files and only the 9 .java files
        - Any additional files added to the submission will result in a loss of points
- Project requires 1 zip file to be submitted

Design Specification Requirements: 100 points

Use the Project 3 UML Diagram as an outline for the project.

1. Display the project title followed by a blank line
2. 20 points: Display “Loading Employee Data” to console
    a. Use the readEmployee method in the FileHandler class to load the data from the employee file
        i. Ignore the header in the csv file
        ii. Use the logger method to write the String “Loading Employee Data” to the log file
    b. The employee_data.csv file contains 2 types of employees tier1 and tier2
    c. Read in each line and determine the type of employee
        i. If employee is tier2 create a Tier2Employee object with the data given
        ii. If the employee is tier1 create an Employee object with the data given
    d. All Employee and Tier2Employee objects must be stored in the Employee ArrayList in the Project3 class file.

3. 25 points: Display “Loading Ticket Data” to console
    a. Use the readTicket method in the FileHandler class to load the data from the 2 ticket files
        i. tier1_ticket_data.csv & tier2_ticket_data.csv
        ii. Ignore the header in the csv file
        iii. Use the logger method to write the String “Loading Ticket Data” to the log file
    b. Create Ticket objects along with their respective Customer objects for each row in the 2 ticket files.
        i. The readTicket method needs to return an LinkedList with the Ticket objects for the given ticket file
        ii. The returned LinkedList needs to be stored in the Ticket Queue in the Project3 class file
            1. Tier1 tickets need to be saved in the tier1 ticket queue
            2. Tier2 tickets need to be saved in the tier2 ticket queue

4. 25 points: Display "Creating Work Orders" to console
    a. Use the createWorkOrders method to iterate over the lists of Tickets and Employees to create WorkOrders for each Ticket in the 2 queues
        i. Call the constructor method in the WorkOrder class to create work orders for each ticket in the list
            1. The WorkOrder objects created need to be loaded into the ArrayList in the Project 3 class file
            2. The createdAt variable for the WorkOrder objects must be assigned the current date and time for when the work order was created.
        ii. Ticket assignment requirements:
            1. Tier1 employees should only be assigned tier1 tickets and Tier2 employees should only be assigned tier2 tickets
            2. A ticket should only ever be assigned to 1 employee
            3. Every employee should have at least 1 ticket assigned to them

5. 30 points: Display "Writing Work Order Data to File" to console
    a. Create a new csv file called workorder_data.csv and write out the data for each WorkOrder object in the workOrderList to the file
        i. Use the writeDate method in the FileHandler class to iterate over the list of WorkOrders
        ii. Use the logger method to write the String "Writing Work Order Data to File" to the log file
        iii. Create a header for the csv file denoting what each row is storing
        iv. Use the getFileData method from the WorkOrder class to get the String containing all the data for the work order then write that String to the csv file
            1. The returned String should be comma separated
            2. The getFileData method in WorkOrder should rely on the getFileData methods of the other classes (Employee, Ticket, & Customer)
        v. Use the logger method to write out the data to the log file for each WorkOrder object created
            1. Use the string returned from the call to getFileData
6. Display "Work Orders created. Program Exiting" to the console
    a. Use the logger method in FileHandler to write the String "Work Orders created. Program Exiting" to the log file

Note: Refer to the sample output in the **Example Output** section below.

Style Requirements: Required

Javadoc's: -10 points

- Each class and method declaration must have a correct Javadoc comment placed above it
    - Add your Javadoc comments last
    - Generate the Javadoc's using an IDE and verify no errors or warning are produced

Naming Conventions: -10 points

- Each class, variable and method must be named appropriately
    - Variables and methods always start with a lowercase letter
    - Classes always start with an uppercase letter

Minimum Requirements: - 100 points

- All 9 class files need to be created and all methods and variables shown in the UML diagram must be present
- The main method cannot be set to throw exceptions
- All class files must be set to inherit or implement the correct classes/interfaces as shown in the UML diagram

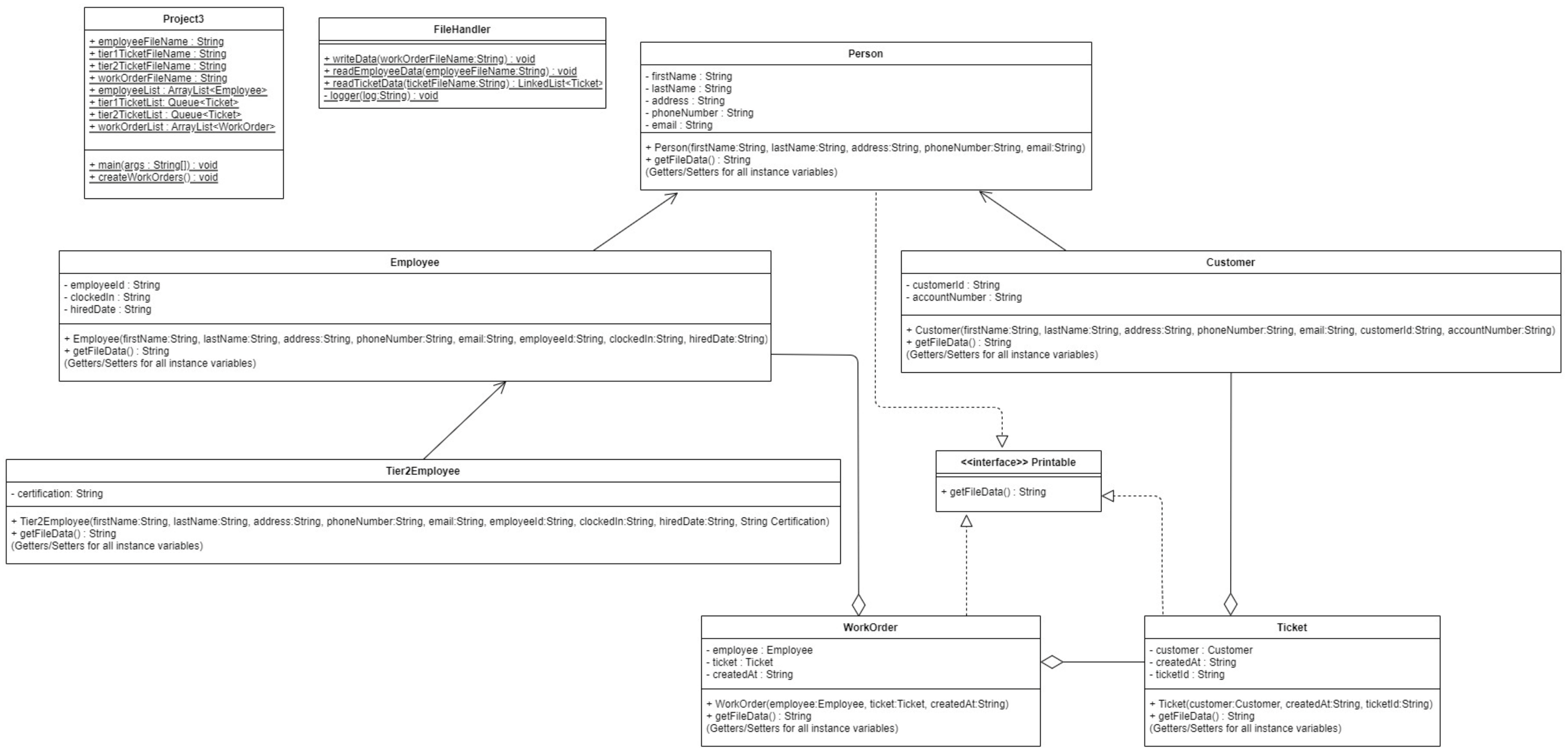
Additional Notes:

- Each class should have getter and setter methods for each class variable. This is not shown in the UML diagram but is required for the project
- Each method and variable outlined in the UML diagram must be implemented in the project. Additional methods and variables can be added if needed.
- Surround each input and output stream with a try catch. No error checking of file data required.
- The String class variables in Project3 need to be used to store the name of the files to be read in and created (employee_data.csv, tier1_ticket_data.csv, tier2_ticket_data.csv, workorder_data.csv)
    - You can assume the files will be placed in the working directory
    - /eclipse-workspace/Project3/"csv file location"
- The FileHandler method logger must be used to write to the log.txt file for the project
    - This method should be set to append to the log file, not overwrite it every time it is executed
    - Each entry to the log file needs to include the date and time the entry was written to the log file
    - Refer to the example log file to determine correct format of the log file
- When outputting the work orders refer to the example workorder_data.csv for the correct format of the data and the required header
- Refer to the getFileData output file to see an example String of what should be returned when calling those methods
    - Each class's getFileData excluding Person must use another class's getFileData method.
    - Failure to leverage inheritance properly when setting up getFileData will result in a lose of points for each class not configured correctly
- Use the @Override notation when implementing the interface methods in the classes
- I suggest using the FileReader class wrapped with the Scanner class to read data in from file
- I suggest using the FileWriter class wrapped with the PrintWriter class to output data to the new file
    - Use the println() method to write to the file line by line
- You do not need to verify the files exist before reading in the data
- Use the SimpleDateFormat class along with the Date class to get the current date & time

Example Output:

```
Project 3 Work Order Generator

Loading Employee Data
Loading Ticket Data
Creating Work Orders
Writing Work Order Data to File
Work Orders Created. Program Exiting
```


Project3
+ employeeFileName : String
+ tier1TicketFileName : String
+ tier2TicketFileName : String
+ workOrderFileName : String
+ employeeList : ArrayList<Employee>
+ tier1TicketList: Queue<Ticket>
+ tier2TicketList : Queue<Ticket>
+ workOrderList : ArrayList<WorkOrder>
+ main(args : String[]) : void
+ createWorkOrders() : void
FileHandler
+ writeData(workOrderFileName:String) : void
+ readEmployeeData(employeeFileName:String) : void
+ readTicketData(ticketFileName:String) : LinkedList<Ticket>
- logger(log:String) : void
Person
- firstName : String
- lastName : String
- address : String
- phoneNumber : String
- email : String
+ Person(firstName:String, lastName:String, address:String, phoneNumber:String, email:String)
+ getFileData() : String
(Getters/Setters for all instance variables)
Employee
- employeeId : String
- clockedIn : String
- hiredDate : String
+ Employee(firstName:String, lastName:String, address:String, phoneNumber:String, email:String, employeeId:String, clockedIn:String, hiredDate:String)
+ getFileData() : String
(Getters/Setters for all instance variables)
Customer
- customerId : String
- accountNumber : String
+ Customer(firstName:String, lastName:String, address:String, phoneNumber:String, email:String, customerId:String, accountNumber:String)
+ getFileData() : String
(Getters/Setters for all instance variables)
Tier2Employee
- certification: String
+ Tier2Employee(firstName:String, lastName:String, address:String, phoneNumber:String, email:String, employeeId:String, clockedIn:String, hiredDate:String, String Certification)
+ getFileData() : String
(Getters/Setters for all instance variables)
<<interface>> Printable
+ getFileData() : String
WorkOrder
- employee : Employee
- ticket : Ticket
- createdAt : String
+ WorkOrder(employee:Employee, ticket:Ticket, createdAt:String)
+ getFileData() : String
(Getters/Setters for all instance variables)
Ticket
- customer : Customer
- createdAt : String
- ticketId : String
+ Ticket(customer:Customer, createdAt:String, ticketId:String)
+ getFileData() : String
(Getters/Setters for all instance variables)