Mia Harang

Doyle Chism

Sam Gumm

Kenny Jia Hui Leong

Mani Raj Rejinthala

Computer Science 362

Project topic: Fashion

Iteration 03

Departments [Use Cases]

HQ	Design	Manufacturing	Marketing
collateGlobalReport	designProduct	createProduct	advertiseDesign
	approveDesign	transferProudct	advertiseEvent
	transferDesign	receiveDesign	manageCollaborations
	createMarketingDesign	createCustomDesign	

Inventory	Modeling	Treasury	HR
updateInventory	scheduleEvent	processPayroll	handleInterview
registerProduct	scheduleFittings	adjustSalary	handleResume
registerStorage	manageStorage		

Committee	Sales	Public Relations	Security
createCatalog	processNewOrder	manageSocialPresence	assignSecurityPersonnel
	handlingOrderReturn	managePublicityActivities	manageSecurityAudits
	registerRetailer		
	GenerateSaleReport		
	TrackRetailerRewards		
	PriceManipulation		

Modeling Department

[Mia Harang]

Use Case #1:

manageStorage

Actor(s): [Storage Team], Team Managers

Preconditions:

1. The modeling department has a warehouse for storing custom garments, camera equipment, props, etc.
2. A team in the modeling department is responsible for delegating items to events/fittings.
3. There is a transportation team in the storage management team whose sole responsibility is transporting items to events/fittings.
4. When an item is requested for an event, that item is something stored in the warehouse.

Main Success Scenario:

1. When an item (Clothing, Technical, Prop, etc.) that only the modeling department uses needs to be stored, it goes to the modeling warehouse.
    a. Modeling warehouse: a warehouse owned and operated by the modeling department
2. The item is created or bought and given to the storage team for storage.
    a. The storage team categorizes each item to which the item belongs and then subcategorizes it into types and is recorded on the system.
3. Any team can request an item from their category, or request for an item to be ordered for an upcoming event.
    a. The storage team will then lookup said item, and retrieve it from the section where it is placed.
4. When a team indicates that an item needs a monthly/daily/yearly recurring order, the storage team can place a flag on the item that reminds them of the set ordering date.
5. Every day, the storage team performs a sweep of each item to make sure that it is not damaged or depleted, and flagged as damaged in the system.
    a. If there is a recurring order flagged, a new one will be ordered.
    b. If there isn't a recurring order flagged, the storage team will contact the team associated with the item to possibly order a new one.

c. Either way, the depleted/damaged item will be discarded and removed from the system.

Alternative Flows:

- An item is requested, but it is not stored in the system.
    - A storage team member will be sent to see if the item is placed in the requesting department's section but not recorded. If it is found, record the item and location in the system and mark it as checked out. If it is not, place an order for the item and flag it as expedited.
- An item is requested for a date, but cannot be ordered quickly enough.
    - The date is requested to be pushed back or continue without the item.
- An item is damaged while checked out.
    - The item is returned by the team so it can be marked as damaged, then the normal damage routine takes place.
- An item is stored in the system, but not found in the warehouse.
    - If there is a recurring order, the item will be reordered. Otherwise, the team associated with the item will be contacted.

Marketing Department

[Mia Harang]

Use Case #2:

manageCollaborations

Actor(s): [HOD], Team Managers

Preconditions:

1. The marketing department has the budget to afford to collaborate with celebrities/brands.
2. There exists a list of pre-approved celebrities/brands that the company is ok with collaborating with.

Main Success Scenario:

1. The Company decides that they would like to move forward with a collaboration with a specific celebrity/brand.
    a. If the celebrity/brand is not on the pre-approved list, the HOD will put in a request to have them added.

2. The HOD will meet with the Team Managers to find a celebrity/brand that they would like to collaborate with.
3. When a celebrity/brand is found, the HOD will be tasked with contacting the celebrity/brand for collaboration.
4. A meeting with the celebrity/brand and the Team Managers will take place to figure out what kind of collaboration they would like to be involved with.
    a. A modeling event, advertisement, etc.
5. Once a collaboration is decided, that kind of event/advertisement is requested through the necessary departments with the celebrity/brand that is associated with it.
    a. If the celebrity/brand wants to jump in on an event that is already in the works, a request for them to be added will be issued.
    b. A celebrity will be stored as a temporary model to be deleted afterwards
6. Any complaints or changes made to the schedule of the celebrity/brand or the event that they are participating in, need to be made through the marketing department where it will be handled.
7. Once the collaborations have been finished, any temp employees will be deleted.

Alternative Flows:

- The celebrity/brand that is contacted rejects the brand collaboration request.
    - A new celebrity/brand will be researched and a new request will be made. If no celebrity/brand is willing, the collaboration will be thrown out the door.
- A celebrity/brand contacts the marketing department first.
    - If they are on the pre-approved list, the collaboration will take place. Otherwise, the marketing department will request for the celebrity/brand to be put onto the pre-approved list. If not approved, the request will be denied.
- A request is put in to remove a celebrity/brand from the pre-approved list.
    - If there is a collaboration with the celebrity/brand in the process, the collaboration will be swiftly terminated. The celebrity/brand will be removed from the pre-approved list.
- A celebrity/brand initially agrees to go through with collaboration but backs out.
    - A new celebrity/brand will be researched and a new request will be made to jump in as a replacement. If no celebrity/brand is willing, the collaboration will be thrown out the door.

Manufacturing Department

[Doyle Chism]

Use Case #1:

createManufacturingReport

Actor(s): Head of Manufacturing, Manufacturing Manager
Preconditions: All the designs have been approved and sent to manufacturing so the manufacturing can begin processing and creating the product. The actors have full access and control to input the timeframe for which they are collecting materials and creating products.
Main Success Scenario:

1. The Manufacturing Department receives a Final Design or a custom Design from the Design Department.
2. The Head of Manufacturing receives the design and evaluates the specifications.
3. The Head of Manufacturing alerts the Manufacturing Manager to collect the raw materials for the product.
4. The Manufacturing manager receives the specifications and retrieves the materials needed for the product and then enters the time at which they collected the rawMaterials to track progress in the system.
5. The Head of Manufacturing verifies the raw materials retrieved by the manufacturing manager and then enters the time at which he verified the raw materials in the system.
6. The Manufacturing manager begins creating the product with the raw materials and also logs the time at which they started production of the product.
7. The product is created and collected by the manufacturing manager and is given to the Head of Manufacturing for processing.
8. The manufacturing manager then logs the time at which the product was finished being created.
9. The Head of Manufacturing can view the Manufacturing report to understand how long it took to collect raw materials and create a product.
10. The Head of Manufacturing sends the Custom Design to the Modelling Department.
11. The product is created and it is stored in the Inventory Department.

Alternative Flows:

6A: The machine is not on, so it has to be turned on to begin production.
7A: The product does not meet quality standards, is the wrong size, or is the wrong

color so it has to be made again repeating steps 4-8.

9A. The Manufacturing Report shows the time it takes to do Scenarios 4, 5, 6, 7.

9B. The Head of Manufacturing can then be able to see how to optimize the manufacturing process based on the data collected from the timeframe it takes to complete tasks.

Design Department

[Doyle Chism]

Use Case #2:

organizeDesignCreation

Actor(s): Head of Design Department

Preconditions: The sketches have been created and are waiting to be approved by the Head of Design to begin working on the Final Designs.

Main Success Scenario:

1. The Head of Design sets a Final Design for Manufacturing based on the sketches.
2. The Head of Design then sets the marketing specifications based on the Final Design that was selected for Manufacturing.
3. After the Marketing design is set, the design team beings creating the specifications needed for Marketing the product.
4. The Head of Design receives a custom design from the modelling department.
5. The Head of Design and the Design Team customizes and creates the custom design that the modelling team needs.
6. After the Head of Design sets the Final Design and Custom Design for Manufacturing then they will send the specifications to the Manufacturing Department to create the product.
7. After the Head of Design sets the Marketing Design based on the Final Design they will send the specifications to the Marketing Department to begin marketing the product.
8. After the Final Designs have been created you have the option to remove them or keep them stored in the file system.

Alternative Flows:

1A. The Head of Design can select a design sketch and remove it from the file.

1B. Only one Final Design will be selected for Manufacturing at a time.

1C. The Head of design can select a final design and remove it from the file.

2-3A: The Head of Design either verifies the sketch or makes changes to the current sketch to meet the requirements they see fit.

7A: The marketing design is set based on the Final Design for Manufacturing

7B: The Marketing Design is set based on targetAudience, price, seasonType, and description. Given a brief explanation on why these are marketable items.

7C: The head of design will receive the manufacturing product and either change modifications of the final design or tell them to make the product again.

HR Department

[Sam Gumm]

Use Case #1:

handleInterview

Actor(s):

- Recruiting Coordinator (RC)
- Hiring Manager (HM)
- HR Assistant (HA)

Preconditions:

1. Candidate has been selected for an interview by Hiring Manager
2. Candidate’s information is in record
3. Available interviewers and time slots are known.

Main Success Scenario:

1. RC accesses the interview scheduling module via terminal.
2. RC inputs candidates' unique ID to retrieve records.
3. RC selects available interview time and records the assigned interviewer.
4. System records time, unique ID, interviewer, candidate, and interviewer notes into a text file.
5. System flags candidates record with a “Scheduled” tag.
6. After the interview is complete, the interviewer records their notes into the candidates interview file.
7. Candidate is moved from the Applied folder to the Approved to await processing by HM.

Alternative Flows:

- If the chosen time slot is already booked, the system prompts the Coordinator to try another time.
- If the interviewer is unavailable, the Coordinator either selects a different interviewer or reschedules.
- If candidate data is incomplete or not found, the system notifies the RC for correction.

Use Case #2:

handleResume

Actor(s):

- Candidate
- HR Assistant
- Hiring Manager

Preconditions:

1. Candidate is known to the organization
2. Candidate's physical resume has been received and reviewed by an internal staff member.

Main Success Scenario:

1. HA receives a paper resume via mail or hand delivery.
2. HA opens the candidate's record in the mainframe database by entering in their unique ID or exact name.
3. HA manually enters key resume data, (i.e. prior positions) into the system's text fields.
4. System updates candidate records to include new details.
5. Physical resume is stored onsite for perusal by the interviewer before meeting with the candidate.
6. HM is able to query the system later to view the candidate's file.

Alternative Flows:

- If the candidate is not yet in the system, the Clerk creates a new candidate record first, then inputs the resume data.
- If resume details are unclear or incorrect, HA seeks out correct information and re-enters the data into the system.
- If a candidate has previously applied, the system marks the existing record and queries the HA on how to handle the conflict.

Public Relations Department

[Kenny Jia Hui Leong]

Use Case: manageSocialPresence
Level:
Primary Actor: Head of Public Relations (HoPR)
Stakeholders & Interests:

- HoPR: Wants fast and accurate scheduling of social media content, wants no mistakes in content assignments that fail to follow SOP or miss deadlines, can result in controversy towards the fashion company and its brand image. This leads to reputational and financial losses.
- HoDepartments: Wants to propose new social media posts or make changes to existing account information. Wants content of posts and account information to fall in line with SOP.
- CEO/Headquarters: Wants brand to have consistent social media presence to build stronger relationships with customers, increasing brand loyalty and attracting investors.
- Public relations employees: Wants to receive fast and accurate schedules for social media related assignments.
- Customers: Wants to see consistent and engaging posts with up-to-date content from the fashion brand.

Preconditions: public relations employee data and social posting requests are complete and up-to-date

Main success scenario:

1. Departments submit new social content requests and the system records it.
2. HoPR enters updated PR employee information and retrieves existing lists from the system.
3. System records new personnel information if applicable.
4. HoPR receives and views details about social content requests from departments in the system, e.g. category of content, deadline, chosen platform, content development stage, etc.
5. HoPR goes through social content requests and assigns PR Planning Team employees based on content category or specific needs.
6. HoPR goes through developed social content requests and assigns PR Review Team employees based on content type or specific needs.
7. System records data and presents PR employee schedules with complete information, e.g. assigned employees, deadlines, tasks, etc.
8. HoPR repeats 5-7 until done.
9. HoPR verifies assignment schedules and confirms in system.
10. System stores assignment schedules to repository.
11. System sends assignment schedules with statuses to departments.
12. System sends assignment schedules to PR employees.
13. HoPR closes system.

Extensions:
4a. Receives developed social content that requires revision

1. PR Review Team marks social content as needing revision.
2. System flags and presents developed social content needing revision.
3. HoPR modifies schedule assignment detailing revisions required after reviewing flagged social content.
4. HoPR reassigns and reschedules PR Review Team employees who authored the social content review to assist PR Planning Team.
5. System records the created revision schedule and sends it to the authors from the PR Planning Team.
6. System sends pending social content notification to specific department.
7. System receives new data and updates PR Review Team employee schedules accordingly.
8. System sends updated social content review schedules to PR Review Team employees.

4b. Receives emergency content post from department

1. System flags and presents emergency for specific content post.
2. HoPR performs immediate schedule override.
3. System presents all available or assigned PR employees.
4. HoPR reschedules available or assigned PR Planning and Review employees to specific emergency content request.
5. System receives new data and updates PR employee schedules accordingly.
6. System sends changed assignment schedules to PR Planning Team employees.
7. System sends changed assignment schedules to PR Review Team employees.
8. System sends changed assignments schedules to specific department.

Use Case: managePublicityActivities
Level:
Primary Actor: Head of Public Relations (HoPR)
Stakeholders & Interests:

- HoPR: Wants fast and accurate scheduling of publicity events, wants events to meet fashion brand standards and follow SOP, can lead to reputational damage and financial losses on failures.
- CEO/Headquarters: Wants brand to have consistent social events to generate good publicity for the fashion brand. Wants these events to maintain a positive image of fashion brand, increasing brand loyalty and attracting new investors.
- Public relations employees: Wants to receive fast and accurate schedules for publicity event related assignments.
- Customers: Interested in seeing the fashion brand involved in events that showcase good values.

Precondition: public relations employee data is complete and up-to-date

Main success scenario:

1. HoPR enters updated list of PR employees.
2. System records new PR employees if applicable and displays all PR employee data.
3. HoPR receives and views planned or reviewed publicity events from the system, e.g. publicity event planning status, publicity events to be reviewed, publicity events that need revision, etc.
4. HoPR goes through scheduled publicity events and assigns PR Planning Team employees based on types of events, deadline, priority level.
5. HoPR goes through developed publicity events and assigns PR Review Team employees based on event type or specific needs.
6. System records data and presents publicity event schedules with complete information, e.g. assigned employees, type of event, specific instructions, and etc.
7. HoPR repeats 4-6 until done.
8. HoPR verifies publicity event schedules and confirms in system.
9. System stores publicity event schedules to repository.
10. System sends publicity event schedules to PR employees.
11. HoPR closes system.

Extensions:

3a. Damage control for publicity event

1. System flags and presents negative incident for specific publicity event.
2. HoPR creates damage control event for negative incident after reviewing failed publicity event.
3. System records the created damage control event and sends it to event authors of the publicity event.
4. HoPR assigns new employees or reassigns existing ones to assist in the damage control event.
5. System receives new data and updates PR employee schedules accordingly.
6. System sends damage control event schedules to PR employees.

3b. Receives planned schedule event that requires revision

1. PR Review Team marks publicity event as needing revision.
2. System flags and presents developed publicity event needing revision.
3. HoPR modifies schedule assignment detailing revisions required after reviewing flagged publicity event.
4. HoPR reassigns and reschedules PR Review Team employees who authored the publicity event review to assist PR Planning Team.
5. System records the created revision schedule and sends it to the authors from the PR Planning Team.
6. System receives new data and updates PR Review Team employee schedules accordingly.
7. Systems sends updated publicity event review schedules to PR Review Team employees.

Sale Department

[Mani Raj Rejinthala]

Use Case #1:

Use Case: GenerateSaleReport
Actor: Sales Executive

Pre-Conditions:

1) System has access to past sales information

Main Success Scenario:

1) The Sales Representative logs into the system and select sales department
2) Sales Representative enter command to generate sale report.

3) The system provides three options to choose for report like sale analysis by day, month, or year.

4) Sale representative enters a date in specified format to generate a day report.

5) System acknowledges the selection and proceed to analysis all sales happen during the requested date.

6) System analyses each sale in a loop and fetches count of each product sold in that sale

7) The sale report will be printed and saved to file with unique title for the sales on specified date.

8) The sale report specifies the highest demand product and lowest demand product.

9) The sale representative analyses the generated report.

10) The system prompts whether interested to continue analysis on other scales

11) Sale representative enters no and logs out the sale management system.

Extensions:

a) If the sale representative wants to customise sale report

1. Before entering the sale generate command the system lists commands to customise sale report by product name
2. Sale representative enters the customise sale report command
3. The system prompts to enter the products name to analyse its sales
4. The Sale representative enters the product name
5. The system prompts to enter the range of sales to analyse for that specified product.
6. The sales representative enters the available range options and enters in requested format
7. The system generates the slae report for the request product which will be printed and saved to files.
8. The sale representative analyses the report and logs out the sale management system.

b) System prompts to select the range for report

1. If the sale representative selects the year and month scale
2. The system acknowledges the selection.
3. The system prompts to enter the year to analyse or month and year to analyse the monthly sales.
4. The sale report will be printed and saved to file with unique title for the sales on during specified range
5. The sale report specifies the highest demand product and lowest demand product for the entire term.
6. The sale representative analyses the generated report and logs out the sales system.

Use Case #2:

Use Case: ProcessNewOrder

Actor: Sales Executive

Department: Sales

Precondition:

1. Sales department can access the inventory records.

Main Success Scenario:

1. A Retailer meets the sales representative at the office or contacts to place an order.
2. The Sales Representative logs into the system and select sales department
3. Then selects the “New Order Request” option.
4. The Retailer provides their retailer ID and the sales representative enters the id.
5. The system retrieves the retailer's details, such as name and location.
6. The Sales Representative verifies the retailer information
7. Retailer requests to add certain quantity of a list items to the order
8. Sales representative requests the product list at a particular storage
9. System displays the product list of a particular storage
10. Sales representative checks the product id for each of the requested item
11. Sales representative enters the requested fashion items ID’s along with their quantities.
12. The system confirms the availability of the requested items and asks for the confirmation to proceed.
13. The system prompts the available reward points of the Retailer and asks if they want to use them with the current order.
14. the Sales Representative enters yes and the system acknowledges the responses.
15. The system prompts enter the points how much that a retailer desired to apply with the current order.
16. the Sales Representative enters the points the Retailer wishes to use.
17. The system deducts the used reward points from the Retailer’s account and recalculates the order total.
18. The system calculates the reward points the Retailer earns for the current order and prompts the earned points.
19. The Sales Representative confirms and submits the order.

20. The system generates a unique order ID, and sends a request to update the inventory.
21. The System prompts do you need receipt and provides two options yes/no
22. Sales representative asks the retailer and enters the input.
23. Sales representative finishes the order request and logs out the system.

Extensions:

1. If reward points exceed the available amount
   a. The system prompts: “Entered points exceed available points. Please enter within the limit.”
   b. The Sales Representative re-enters the points within the valid range.
   c. The system deducts the valid reward points and recalculates the order total.
2. If the retailer requests the receipt:
   a. The System prompts do you need receipt and provides two options yes/no
   b. Sales representative enters Yes.
   c. System prints the receipt.
   d. Sales representative provides the receipt to the retailer.
3. If the Retailer ID is not valid:
   a. The Sales Representative enters the retailer ID
   b. The system prompts the ID is Invalid.
   c. System prompts to re-enter correct retailer ID
      i. Sales representative verifies the ID with the retailer and renters the ID
      ii. System validates the ID and fetches the stored retailer information
      iii. Sales representative verifies the fetched details by enquiring with retailer.
      iv. Sales representative confirms the check.

Use Case #3:

Use Case: PriceManipulation

Actor: Sales Executive

Pre-Conditions:

1. System has access to past sales information
2. The sales system is integrated with the inventory system to access product details.

Main Success Scenarios:

1. The Sales Executive logs into the sales system.
2. The sales executive enters the command price manipulation

3. The system Requests the product name whose price need to be manipulated
4. The sales executive enters the product name
5. The system acknowledges the input and analyzes the entire sales history of the selected product and compares the total sales count against the default threshold count.
6. Based on the comparison:

a. If the total sales count meets or exceeds the threshold, the system recommends increasing the product price.

b. If the total sales count is below the threshold, the system recommends decreasing the product price.

7. The system displays a detailed explanation of the recommendation, including the sales range and threshold used for the decision.
8. The system provides a prompt for customized sales analysis, asking if the Sales Executive wants to specify a sales range and a custom threshold.
9. Sales executive responds no and logs out the system.

Extensions:

1) Custom Sales Analysis
    a. The system prompts the Sales Executive to specify a sales range (day, month, or year) and a custom threshold count.
    b. Based on the selected range:
        1. Day Analysis:
        a) The system prompts the Sales Executive to enter a date in DD-MM-YYYY format.
        b) The sales representative provides the date
        c) The system calculates the total product sales for the specified day and compares it against the custom threshold.
        d) A recommendation is printed
        2. Month Analysis:
        a) The system prompts the Sales Executive to enter a month and year (MM and YYYY).
        b) The sales representative provides the month and year
        c) The system calculates the total product sales for the specified month and compares it against the custom threshold.
        d) A recommendation is printed
        3. Year Analysis:

a) The system prompts the Sales Executive to enter a year (YYYY).

b) The sales representative provides year

c) The system calculates the total product sales for the specified year and compares it against the custom threshold.

d) A recommendation is provided based on the analysis.

Use Case #4:

Use Case: TrackRetailerRewards

Actor: Sales Executive

Pre-conditions:

1. Each retailer is uniquely identified by a Retailer ID in the system.
2. The sale system tracks details such as points earned, points used, and order totals for every order.

Main Success Scenario:

1. The Sales Executive logs into the sales system.
2. The sales executive enters the command to track rewards
3. The system prompts to enter the retailer id
4. The sales representative enters the retailer id
5. The system retrieves the retailer's details and displays options to analyze rewards such as by order, month, or year.
6. Based on selection:

a. Specific Order Analysis:

1) The system prompts to enter the order id
2) Sales executive enters the order id
3) The system retrieves the order details, calculates the order total, points earned, and points used.
4) The reward details are displayed

b. Monthly Analysis:

1) The system prompts to enter month and year
2) Sales executive enters month and year
3) For each order during the month, the system calculates and displays the order total, points earned, and points used.

4) If no orders are found, a message indicates no orders were placed during the specified month.

c. Yearly Analysis:

1) The system prompts to enter year

2) Sales executive year

3) For each order during the year, the system calculates and displays the order total, points earned, and points used.

4) If no orders are found, a message indicates no orders were placed during the specified month.

7. The sales representative rewards the particular retailer and logs out the sale system.

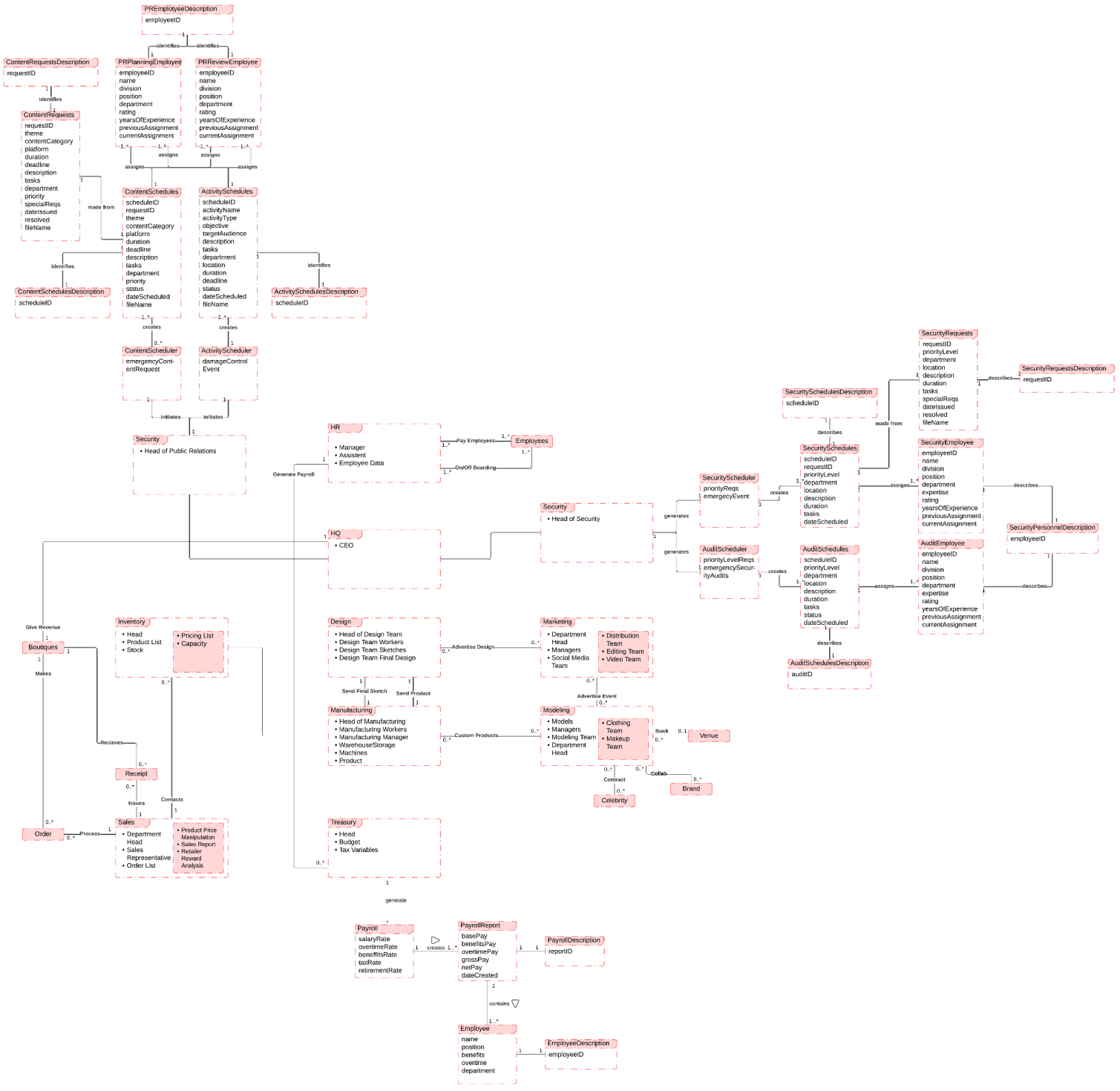
Extensions:

1) If the system finds no orders associated with provided retailer id for the selected date, month, or year

a) The system informs the Sales Representative that no data is available for the specified range.

b) The Sales Representative can either select a new range to analyze or exit the sales system.

Domain Model



Information Expert

Modeling Department:

- **manageStorage:** The main expert for this use case will be the Storage Team Manager. Any manager, team member, etc. in the modeling department can go to this manager to make and manage requests but the main workings will be managed by the Storage Team Manager. In detail, the manager can send members to find items and can add, update, and remove items from the system.

Marketing Department:

- **manageCollaborations:** The main expert for this use case will be the HOD for the marketing department. The HOD is the main communicator between the celebrity/brand and the only person who can request advertisements, events, etc. from other departments. They act as a temporary manager for the celebrity/brand.

Sale Department:

- **GenerateSaleReport:** The Information expert for generating sales reports is SalesManage as it has order details which have methods to fetch product count from each order.

- **TrackRetailerRewards:** The Information expert for Tracking rewards is salesManage as it has order details and retailer details.

- **PriceManipulation:** The Information expert for Price Manipulation is salesManage as it has order details and inventory communicator which provides the product information.

- **PlaceNewOrder:** The Information expert for PlaceNewOrder is saleManage as it has order and retailer details. In detail, salesMange creates the instances of order and retailer objects and calls the necessary methods of those to fulfill the order.

Manufacturing Department[Doyle Chism]:

1. **createManufacturingReport:** The information expert for manufacturing is createManufacturingTimeReport. The manufacturing department is now responsible for tracking what they do in order to optimize the efficiency of their process. They are now obligated to put a timestamp of when they collect raw materials and verify raw materials to track how long it takes from collection to production. The department is also responsible for tracking the start and end times of production to see how long a product

takes to create. These data entries will then create a report making it easy for the department to see the timeframe it takes to complete tasks.

Design Department[Doyle Chism]:

1. **organizeDesignCreation:** The information expert for design is organizeDesignCreation. The Design Department can have many sketches and designs that they are creating and working with throughout a work schedule, so I created a way to organize the amount of work they are dealing with. Once the design department creates a sketch and if they do not need it anymore, they can now remove it from their files. This also applies for final or custom deigns and they can be removed if the design is already created or if they want to remove it to free up space for new designs.

Public Relations Department [Kenny]:

1. **manageSocialPresence**
   The information expert for manageSocialPresence is ContentScheduler. It knows PRPlanningEmployee, PRReviewEmployee, ContentRequests, and ContentSchedules. The method assigned to ContentScheduler will initiate the creation of a social content schedule by retrieving PRPlanningEmployee, PRReviewEmployee, and ContentRequests instances from the repository, then putting them into ContentSchedules.

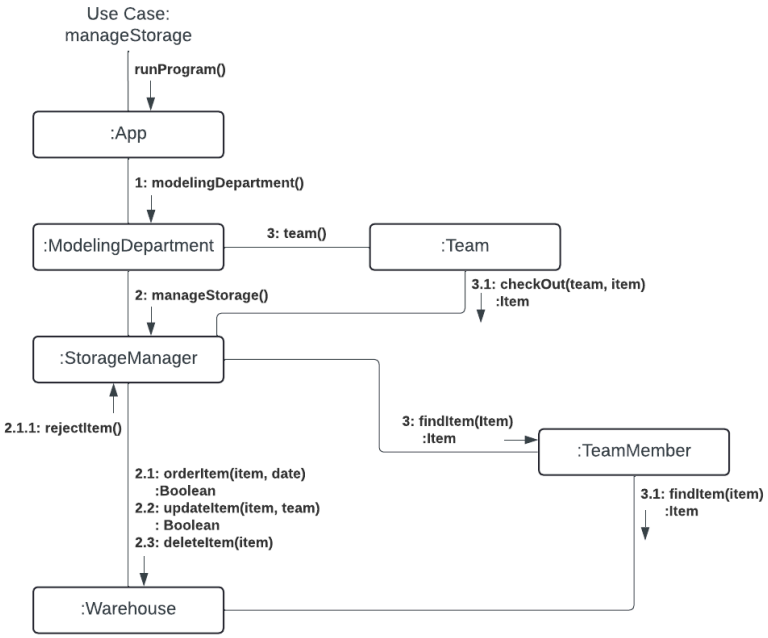
2. **managePublicityActivities**
   The information expert for managePublicityActivities is ActivityScheduler. It knows PRPlanningEmployee, PRReviewEmployee, and ActivitySchedules. The method assigned to ActivityScheduler will initiate the creation of a social publicity activity schedule by retrieving PRPlanningEmployee and PRReviewEmployee instances from the repository, then putting them into ActivitySchedules.

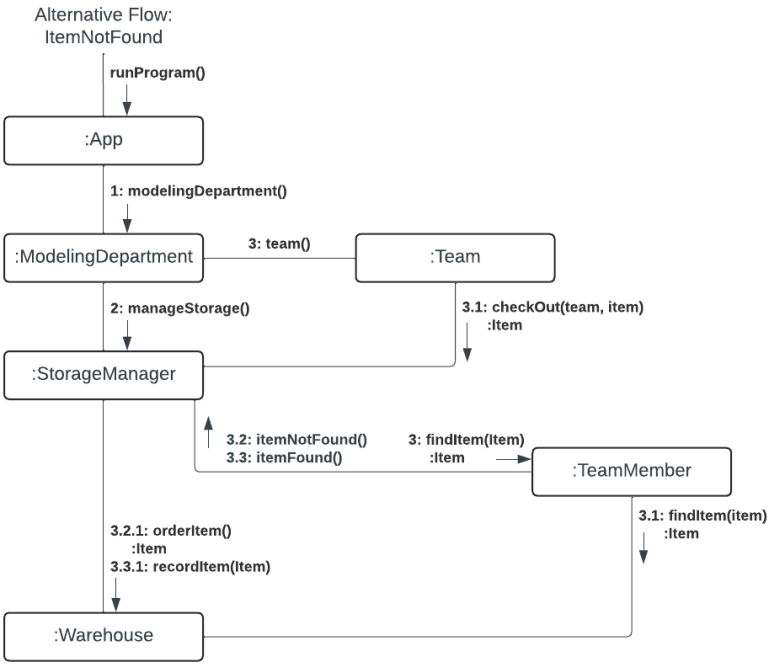
HR Department:

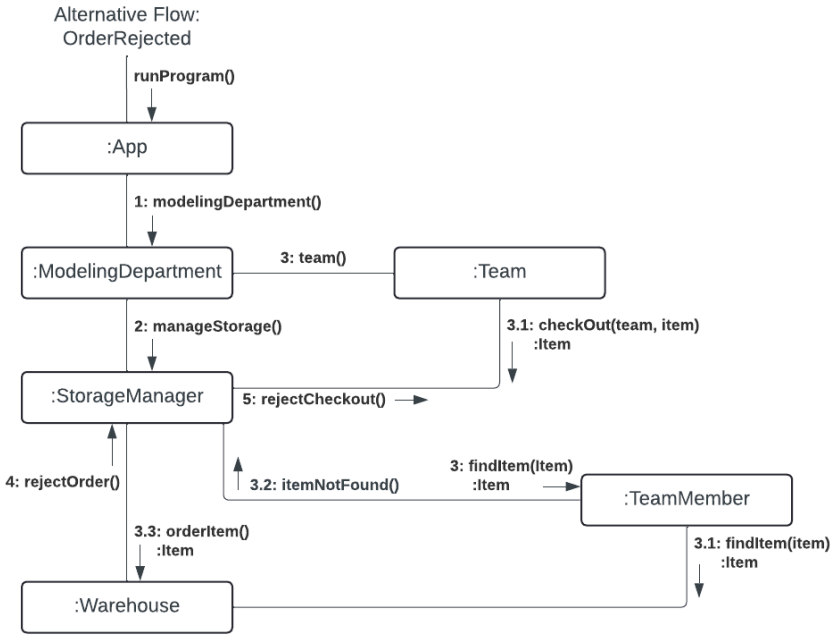
1. employeeRecordManager
   a. This is the information expert for all methods related to CRUDL operations on an employee object. While it uses fileStorageHR for actual file operations, the employeeRecordManager maintains what constitutes a valid employee record, as well as how data should be processed and stored.
2. candidateRecordManager
   a. Similar to employeeRecordManager, candidateRecordManager maintains what a valid candidate object is like, as well as how to interact with it.
3. fileStorageHR
   a. fileStorageHR is the information expert for actual file handling, leaving the other handlers free to focus on domain logic as opposed to low-level file manipulation.

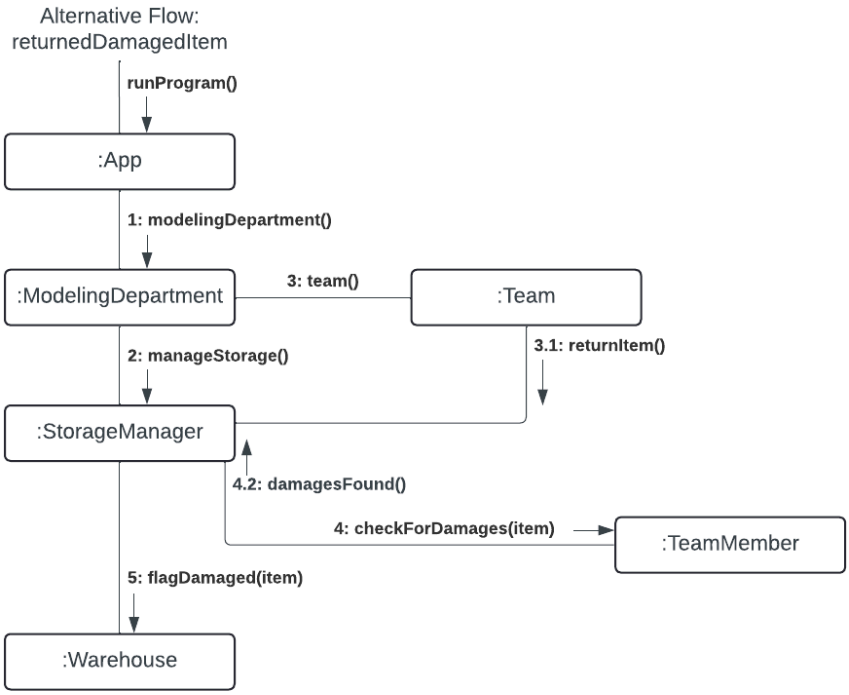
Interaction Diagrams

Modeling Department:

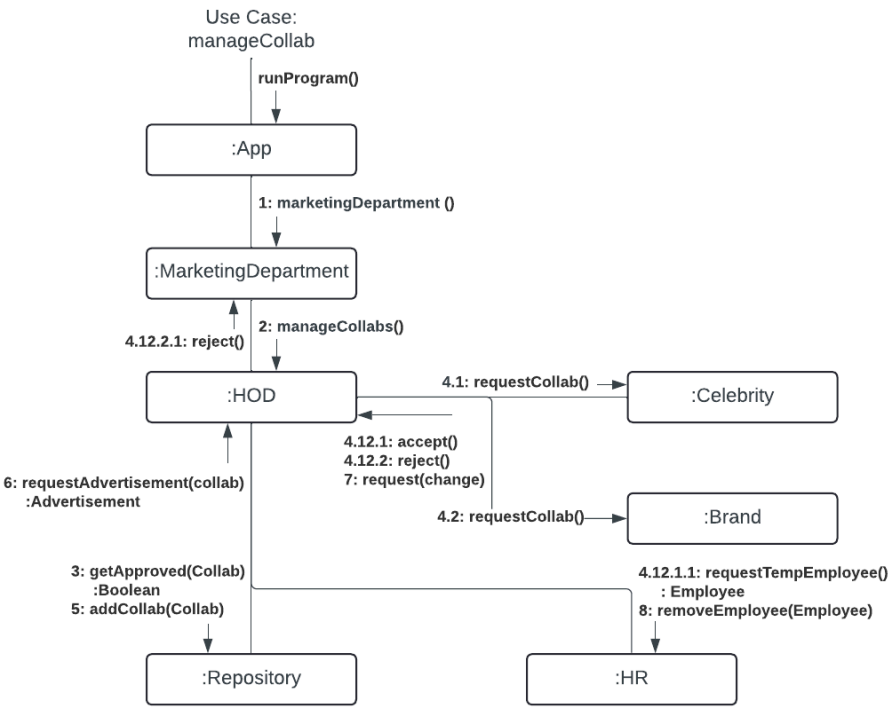


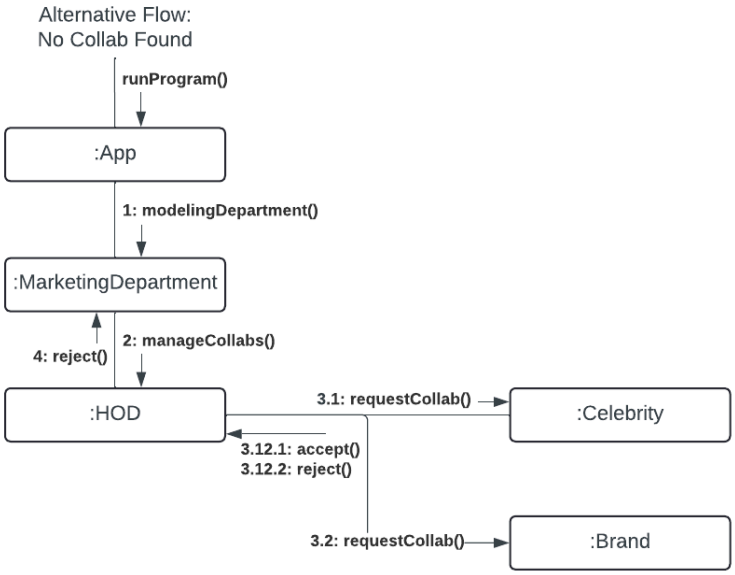


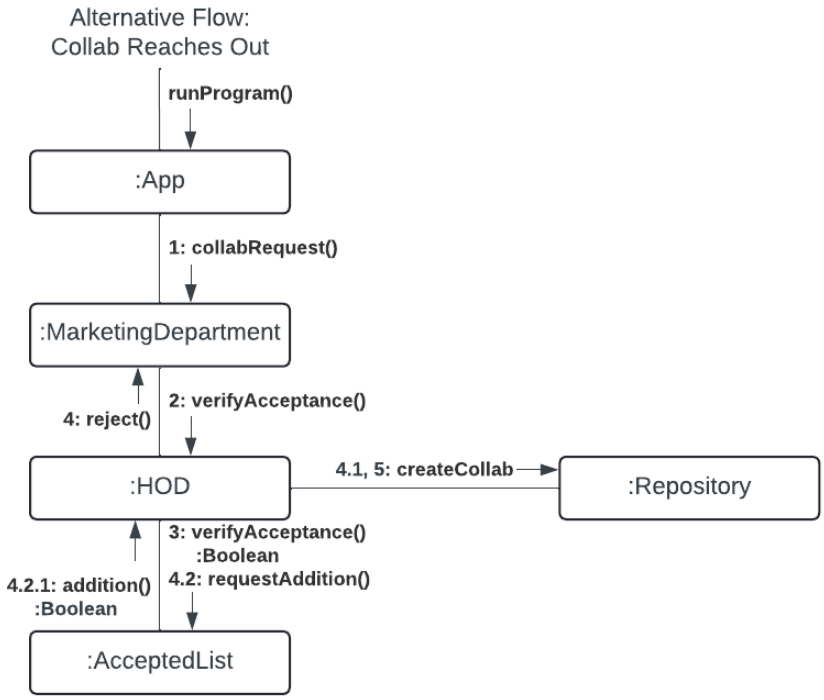

Alternative Flow:
OrderRejected
runProgram()
:App
1: modelingDepartment()
:ModelingDepartment
3: team()
:Team
2: manageStorage()
3.1: checkOut(team, item)
:Item
:StorageManager
5: rejectCheckout()
4: rejectOrder()
3.2: itemNotFound()
3: findItem(Item)
:Item
:TeamMember
3.3: orderItem()
:Item
3.1: findItem(item)
:Item
:Warehouse

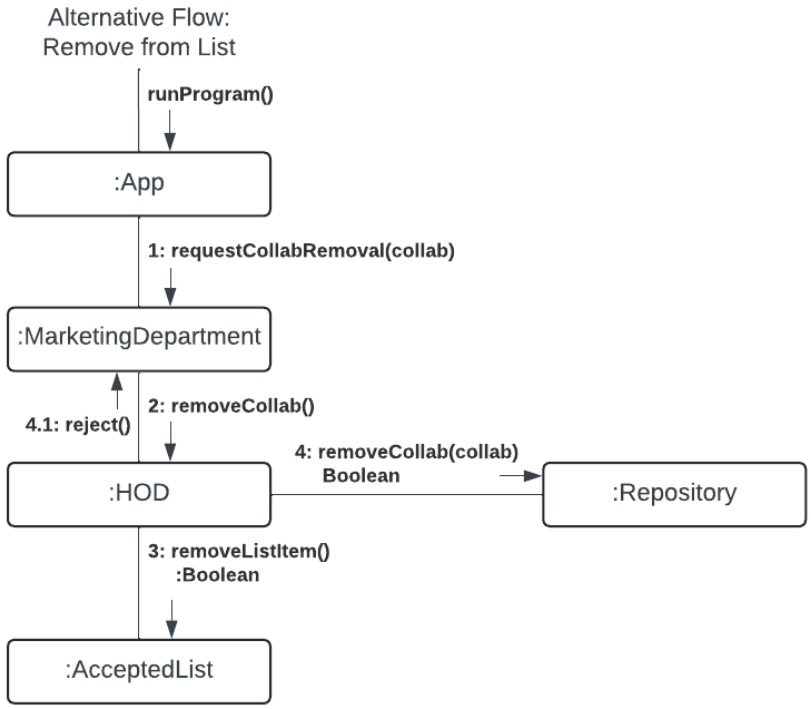

Alternative Flow:
returnedDamagedItem
runProgram()
:App
1: modelingDepartment()
:ModelingDepartment
3: team()
:Team
2: manageStorage()
3.1: returnItem()
:StorageManager
4.2: damagesFound()
4: checkForDamages(item)
:TeamMember
5: flagDamaged(item)
:Warehouse

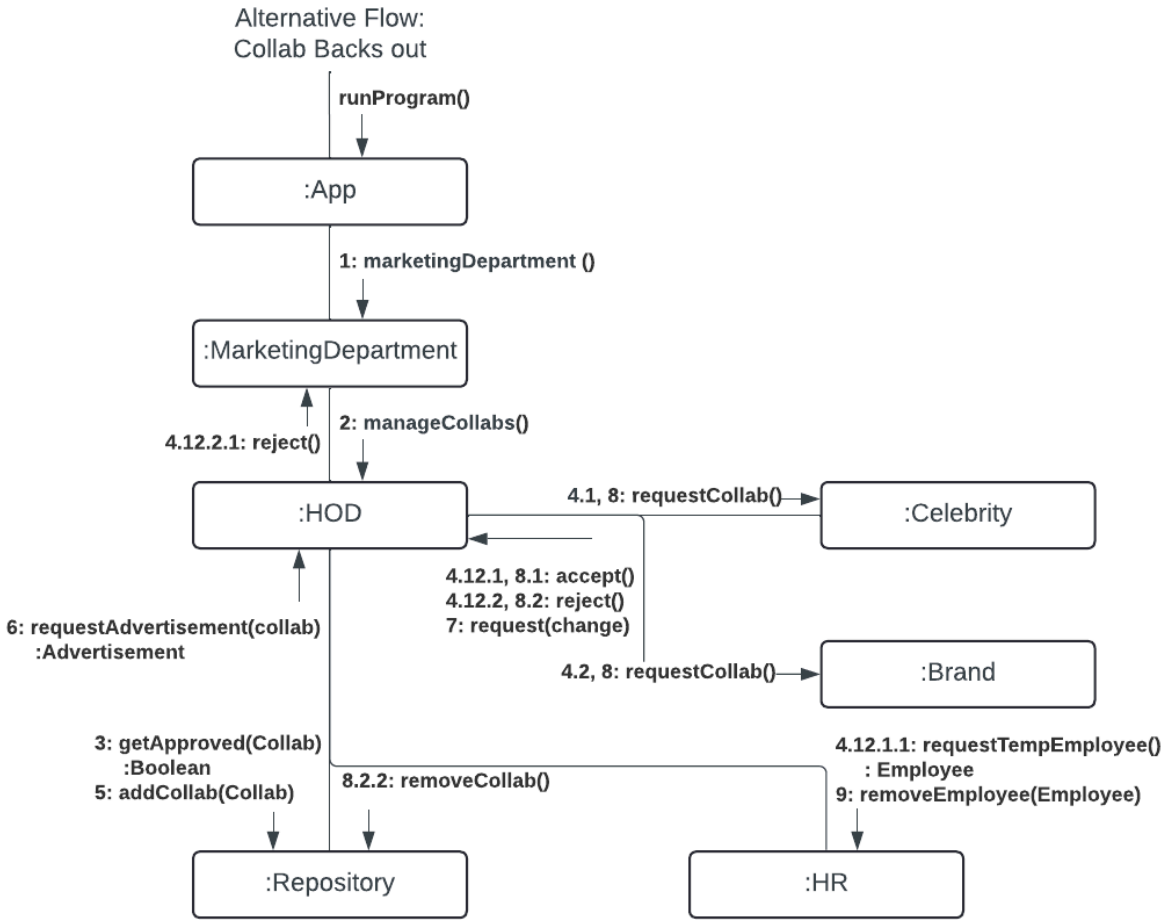
Marketing Department:



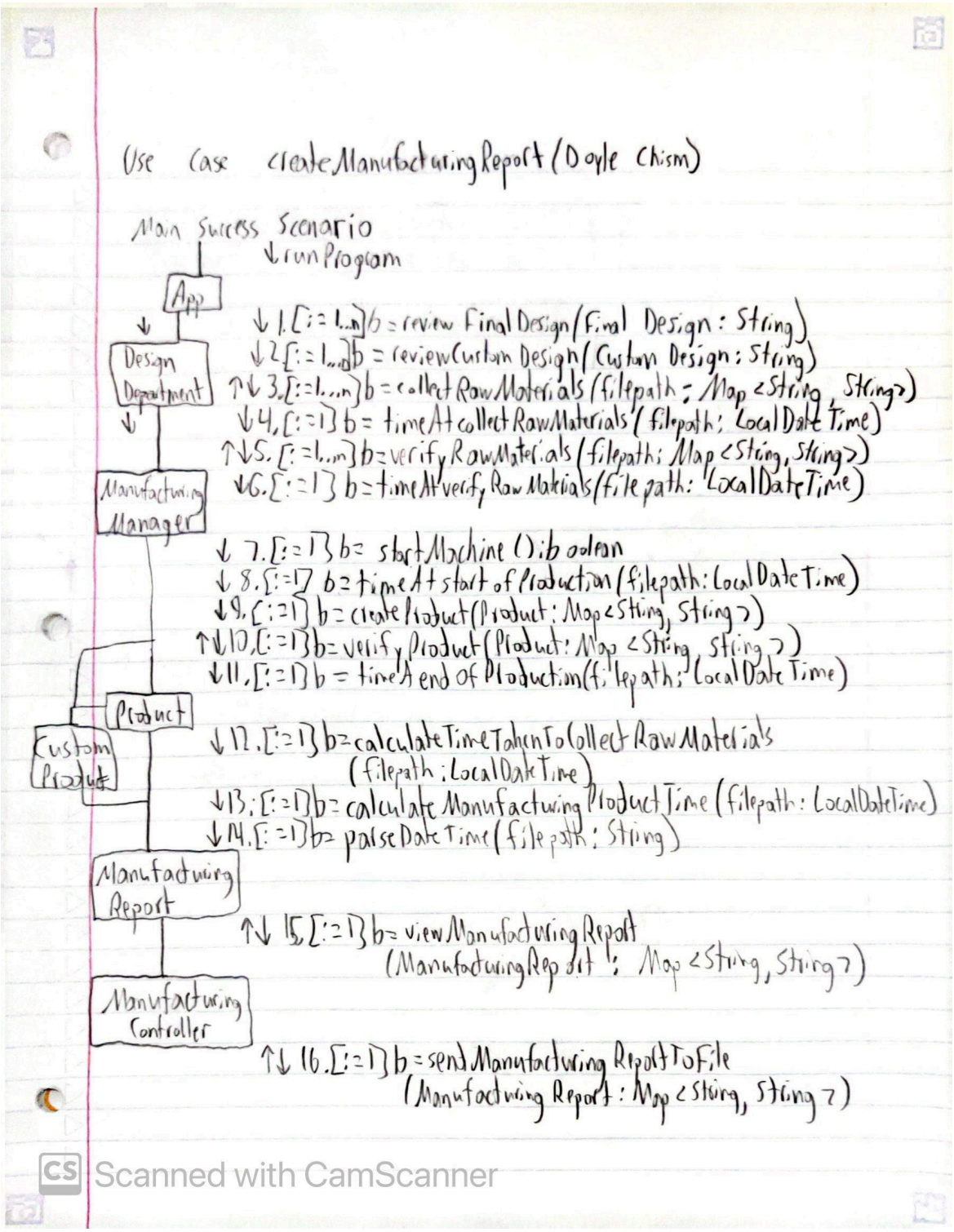



Alternative Flow:
Collab Reaches Out
runProgram()
:App
1: collabRequest()
:MarketingDepartment
4: reject()
2: verifyAcceptance()
:HOD
4.1, 5: createCollab
:Repository
3: verifyAcceptance()
:Boolean
4.2.1: addition()
:Boolean
4.2: requestAddition()
:AcceptedList


Alternative Flow:
Remove from List
runProgram()
:App
1: requestCollabRemoval(collab)
:MarketingDepartment
4.1: reject()
2: removeCollab()
4: removeCollab(collab)
Boolean
:HOD
:Repository
3: removeListItem()
:Boolean
:AcceptedList

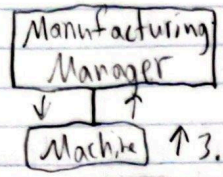

Alternative Flow:
Collab Backs out
runProgram()
:App
1: marketingDepartment ()
:MarketingDepartment
4.12.2.1: reject()
2: manageCollabs()
:HOD
4.1, 8: requestCollab()
:Celebrity
4.12.1, 8.1: accept()
4.12.2, 8.2: reject()
7: request(change)
6: requestAdvertisement(collab)
:Advertisement
4.2, 8: requestCollab()
:Brand
3: getApproved(Collab)
:Boolean
5: addCollab(Collab)
8.2.2: removeCollab()
4.12.1.1: requestTempEmployee()
: Employee
9: removeEmployee(Employee)
:Repository
:HR

Manufacturing Department[Doyle Chism]:



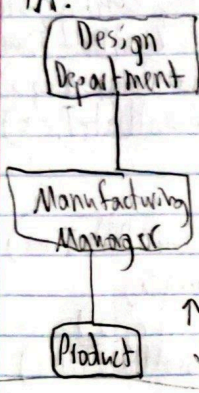
Alternate Flows

6A:



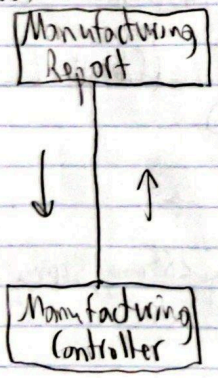
↓ 1. [i:=1] b = verifyMachineType(Machine; String)
↓ 2. [i:=1] b = startMachine(Machine: boolean)
↑ 3. [i:=1] b = machineStatus(Machine; boolean)

7A:



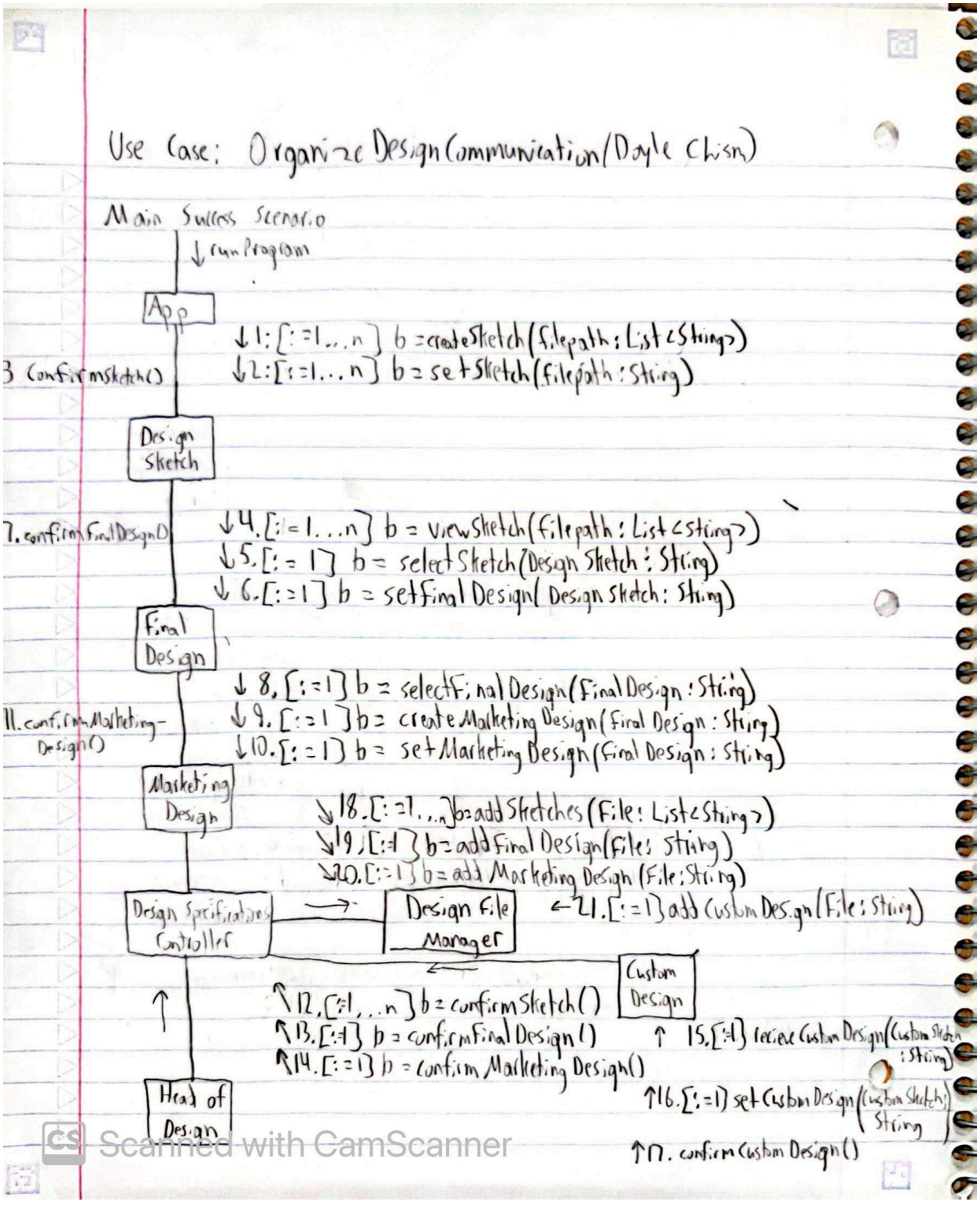
↓ 1. [i:=1...n] b = collectRawMaterials(filepath: Map<String, String>)
↓ 2. [i:=1] b = timeAtCollectRawMaterials(filepath: LocalDateTime)
↓ 3. [i:=1...n] b = verifyRawMaterials(filepath: Map<String, String>)
↓ 4. [i:=1] b = timeAtVerifyRawMaterials(filepath; LocalDateTime)
↓ 5. [i:=1] b = timeAtStartofProduction(filepath: LocalDateTime)
↓ 6. [i:=1...n] b = createProduct(Product: Map<String, String>)
↑↓ 7. [i:=1...n] b = verifyProduct(Product: Map<String, String>)
↓ 8. [i:=1] b = timeAtEndofProduction(filepath: LocalDateTime)

9A-9B:



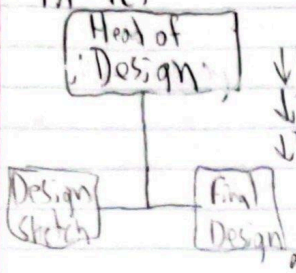
1. [i:=1] b = getStartCollectingTime(filepath: LocalDateTime)
2. [i:=1] b = getVerificationTime(filepath: LocalDateTime)
3. [i:=1] b = getStartTime(filepath: LocalDateTime)
4. [i:=1] b = getEndTime(filepath: LocalDateTime)
5. [i:=1] b = calculateTimeTakenToCollectMaterials() 1-2.
6. [i:=1] b = calculateManufacturingTime() 3-4

Design Department[Doyle Chism]:



Alternate Flows

1A - 1C:



↓1. [i:=1]b = selectSketch(DesignSketch : String)
↓2. [i:=1]b = createFinalDesign(DesignSketch : String)
↓3. [i:=1]b = removeDesignSketch(DesignSketch : String)

↑4. [i:=1]b = selectFinalDesign(FinalDesign : String)
↑5. [i:=1]b = removeFinalDesign(FinalDesign : String)

2 - 3A:

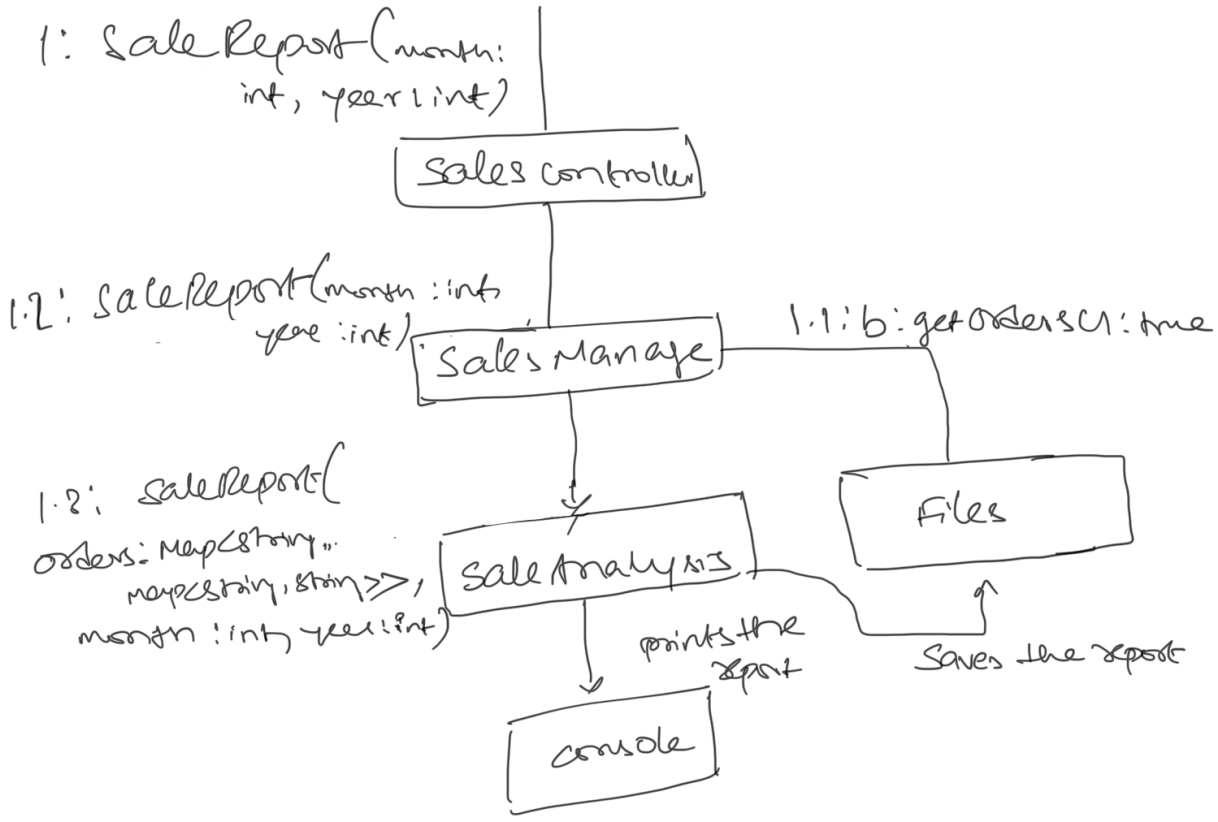
↑↓1. [i:=1]b = viewSketch(DesignSketch : String)
↑↓2. [i:=1]b = verifySketch(DesignSketch : String)
↑↓3. [i:=1]b = isVerified(DesignSketch : boolean)

7 - 7C:

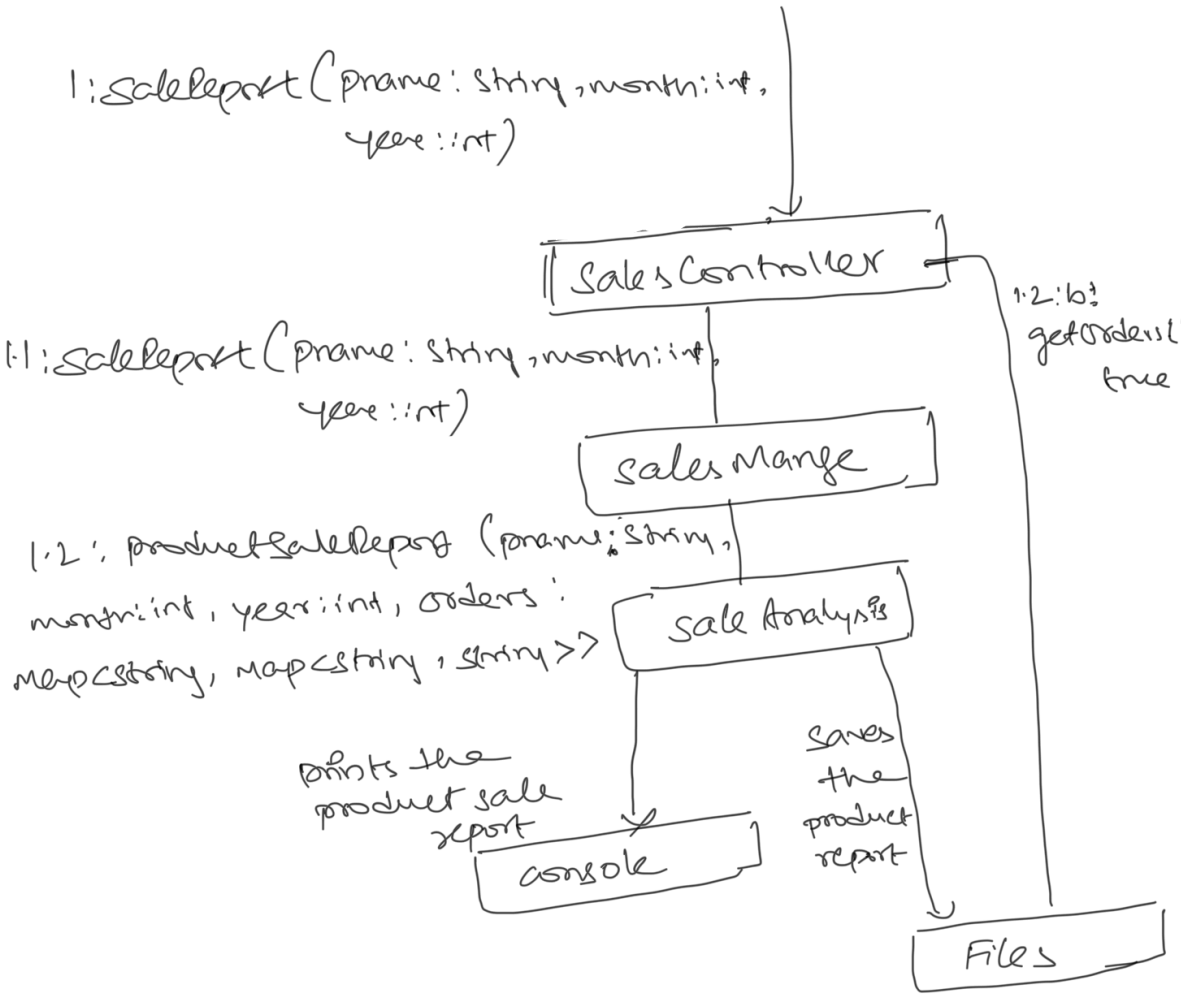
↓1. [i:=1]b = selectFinalDesign(FinalDesign : String)
↓2. [i:=1]b = createMarketingDesign(FinalDesign : String)
↓3. [i:=1]b = setMarketingDesign(MarketingDesign : String)

↑4. [i:=1]b = confirmMarketingDesign(filepath : boolean)

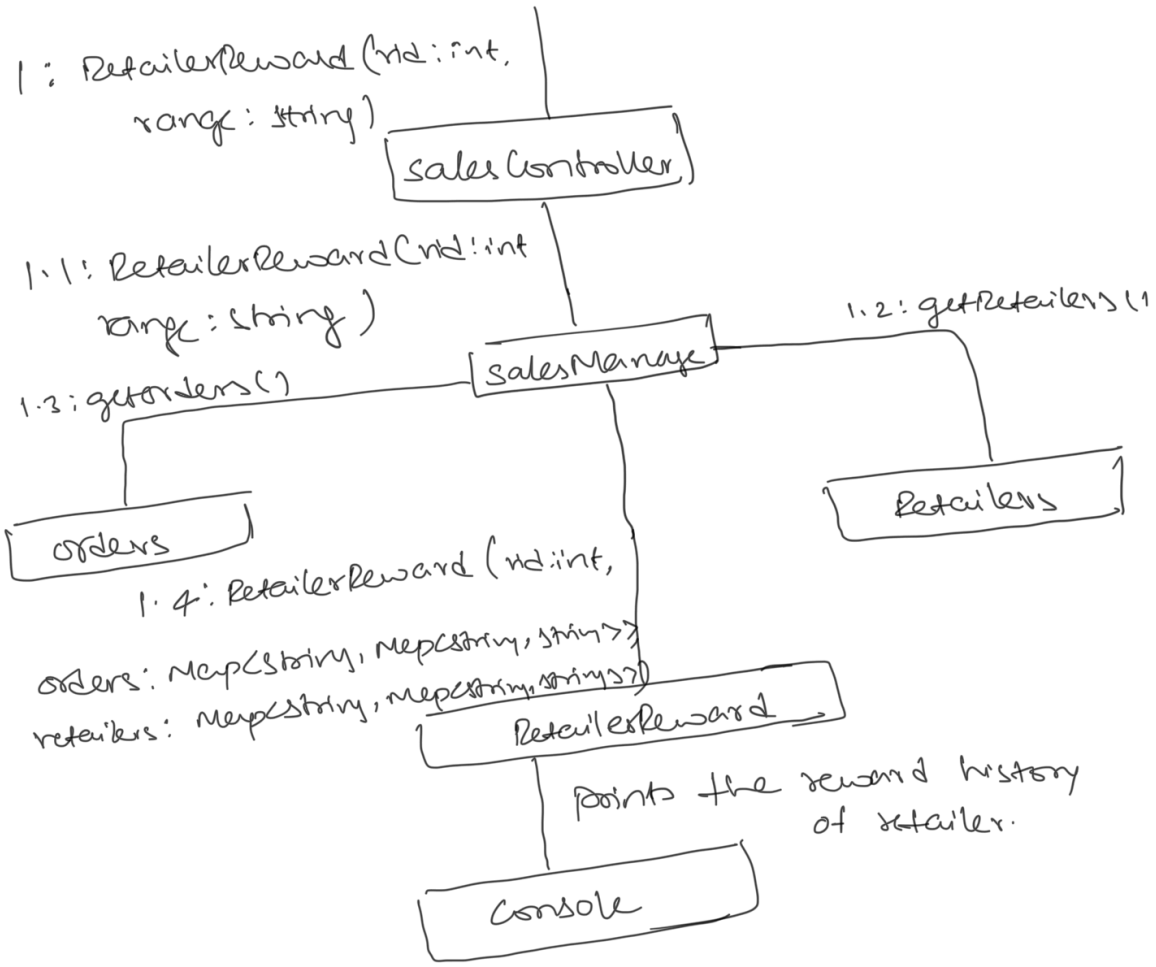
Sale Department:



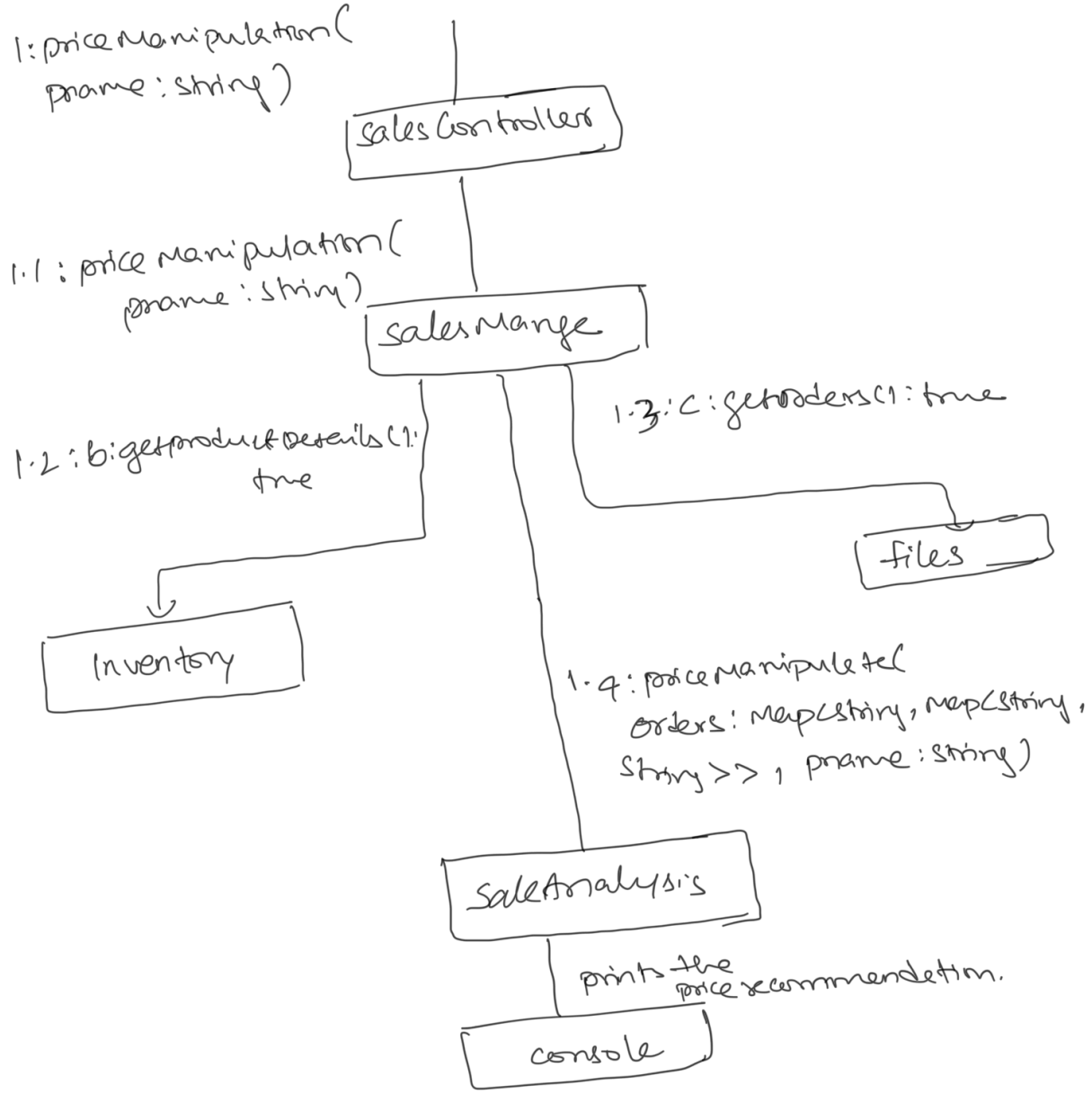
Extension 1


1: saleReport(pname: String, month: int, year: int)
SalesController
1.1: saleReport(pname: String, month: int, year: int)
salesManage
1.2: productSaleReport(pname: String, month: int, year: int, orders: Map<String, Map<String, String>>
Sale Analysis
1.2.1: getOrders(): true
prints the product sale report
console
saves the product report
Files

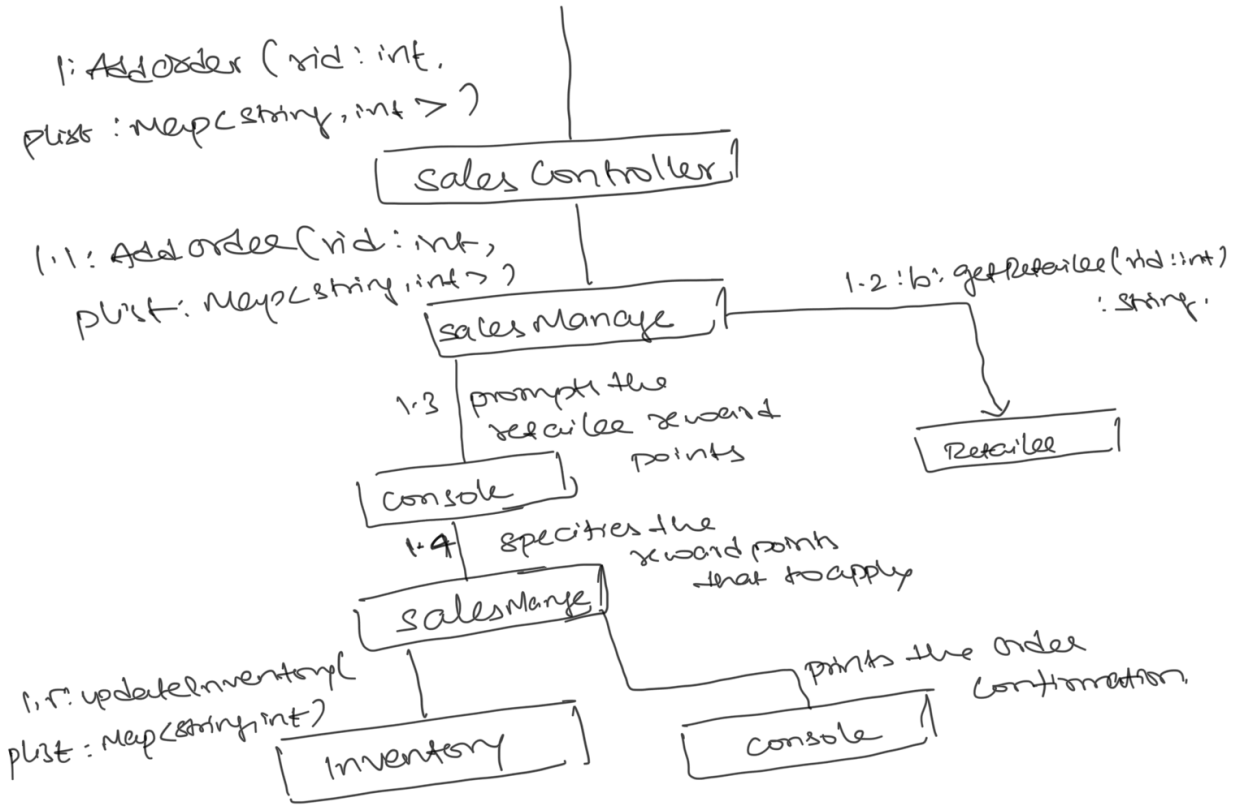
Use Case : TrackRetailerReward.



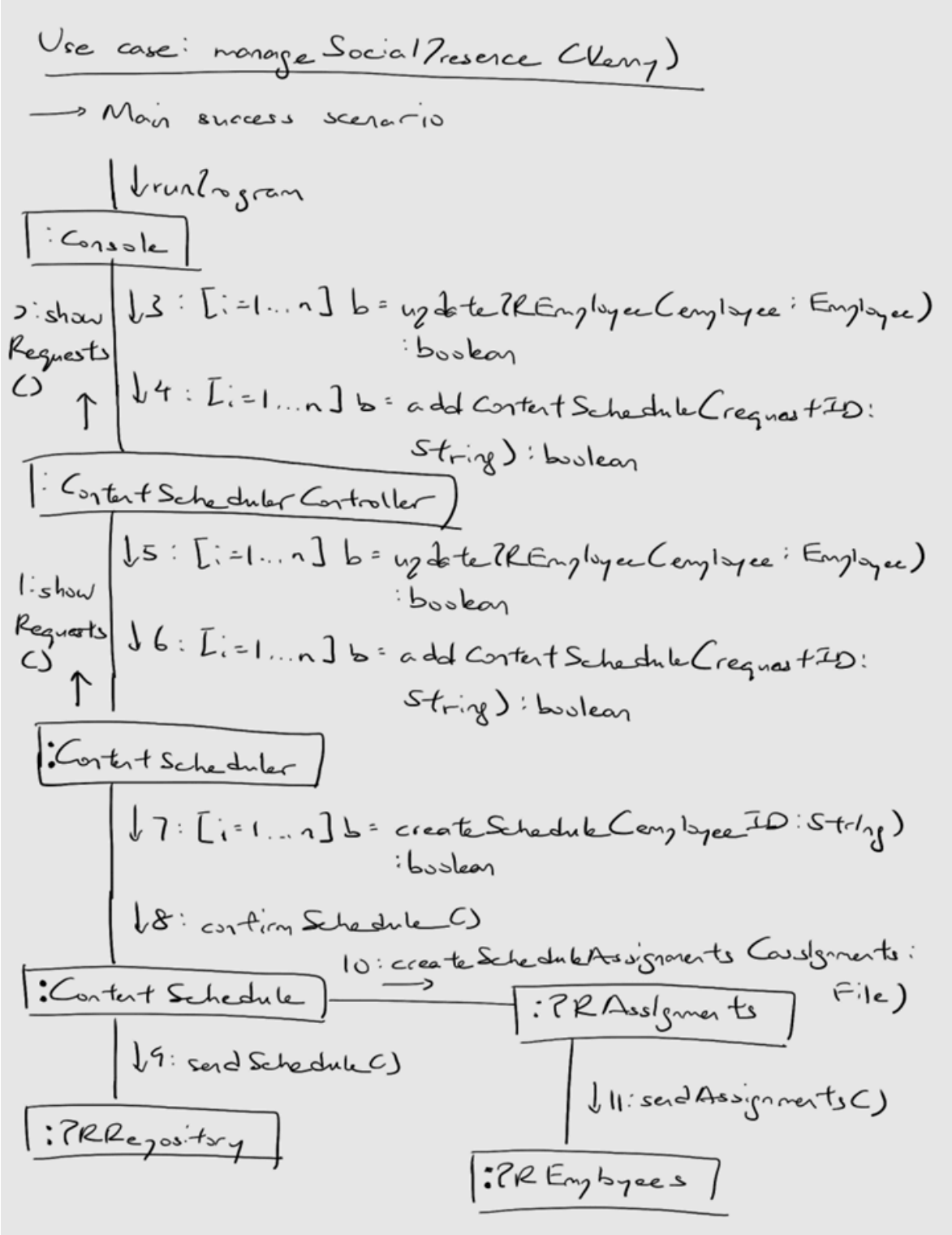
Use case : price Manipulation.



Use case : place new order

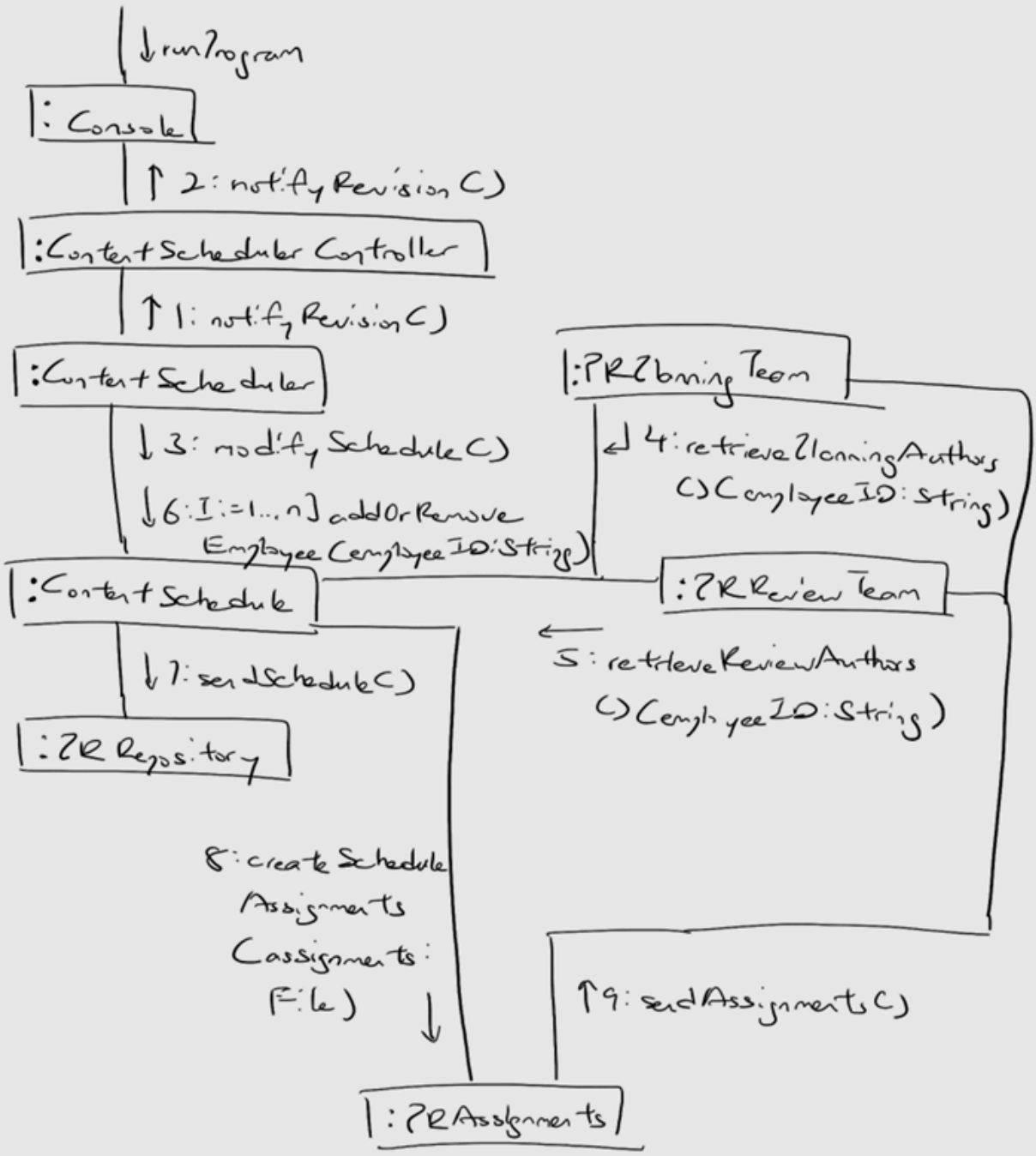


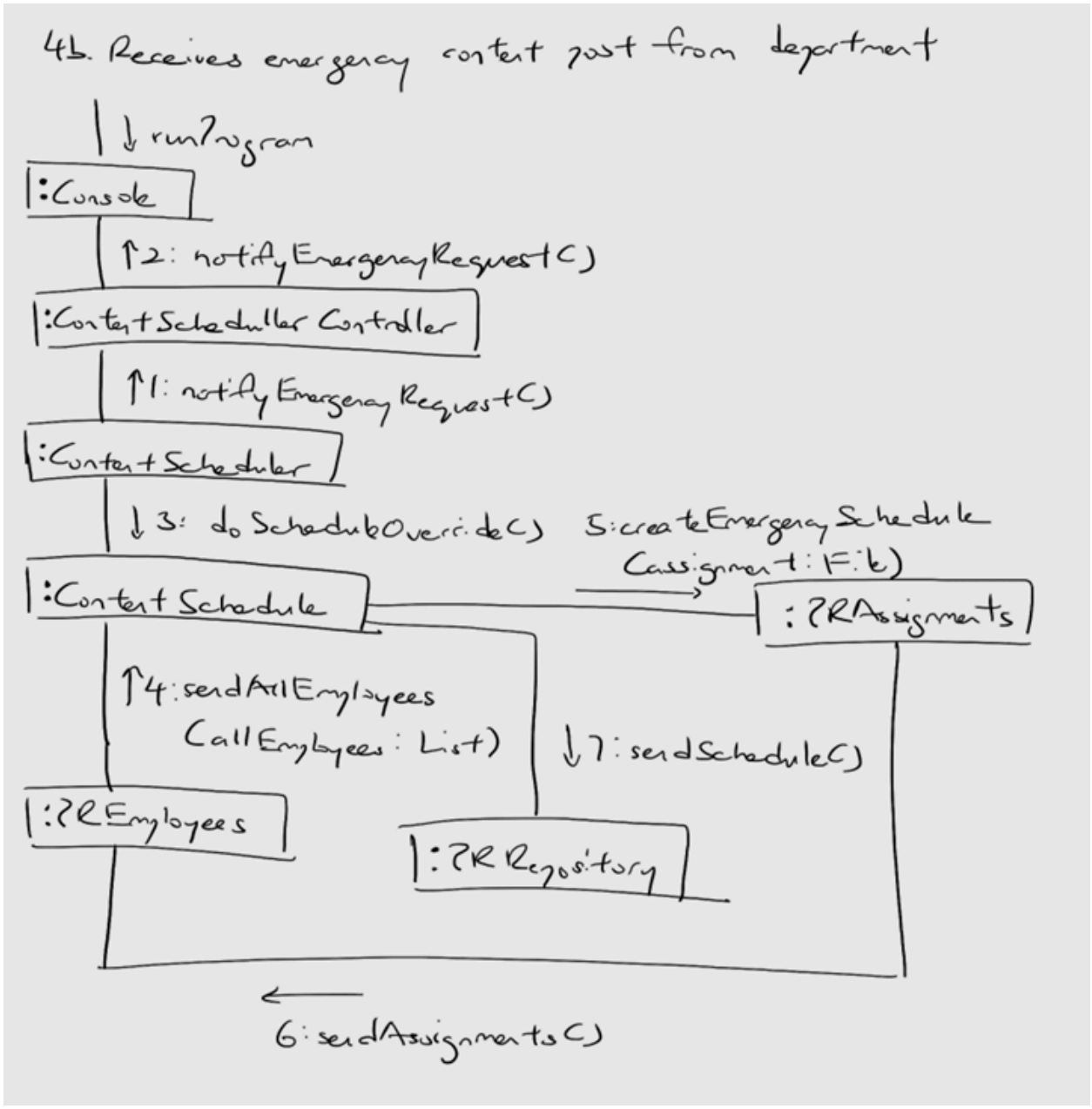
Public Relations:

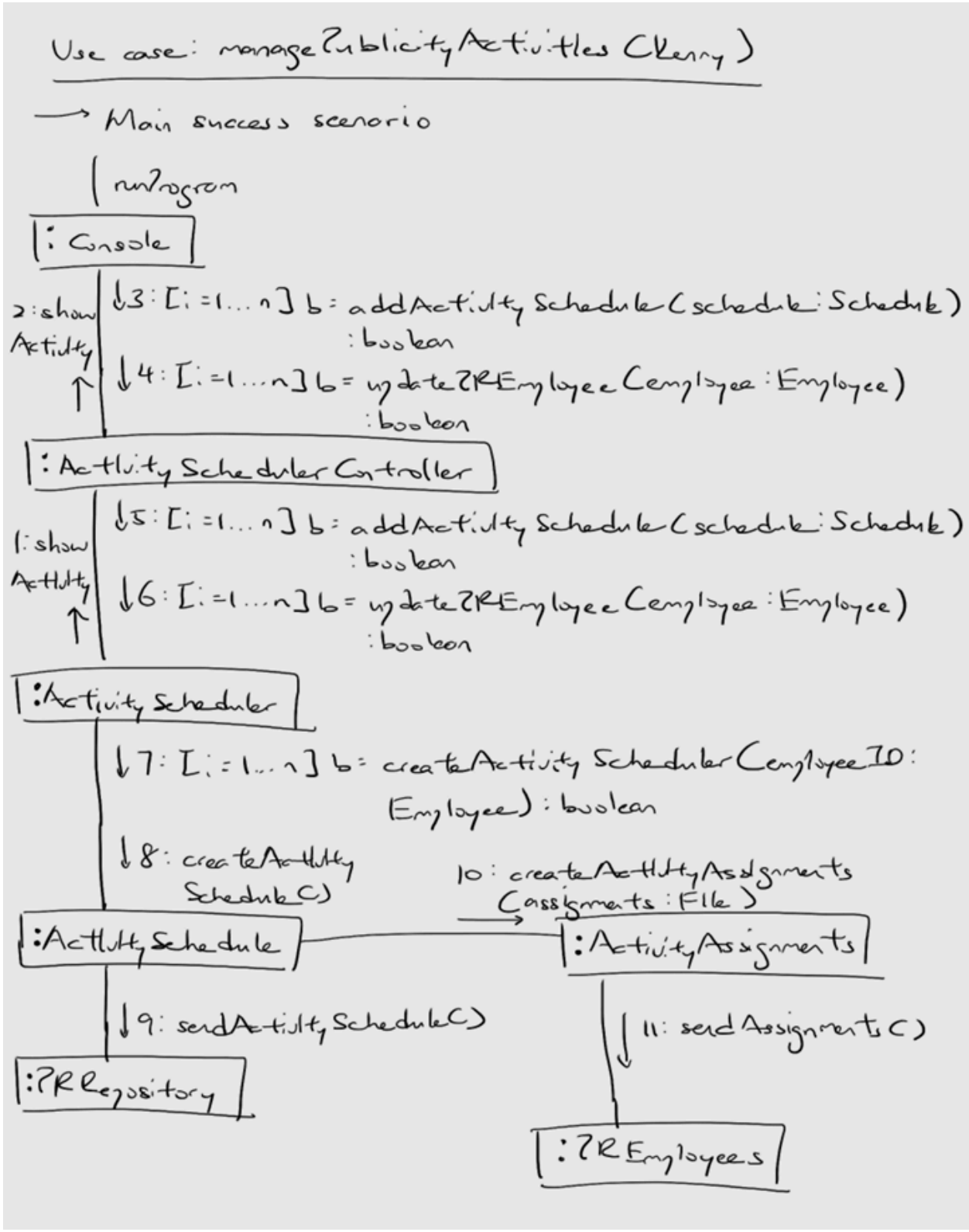


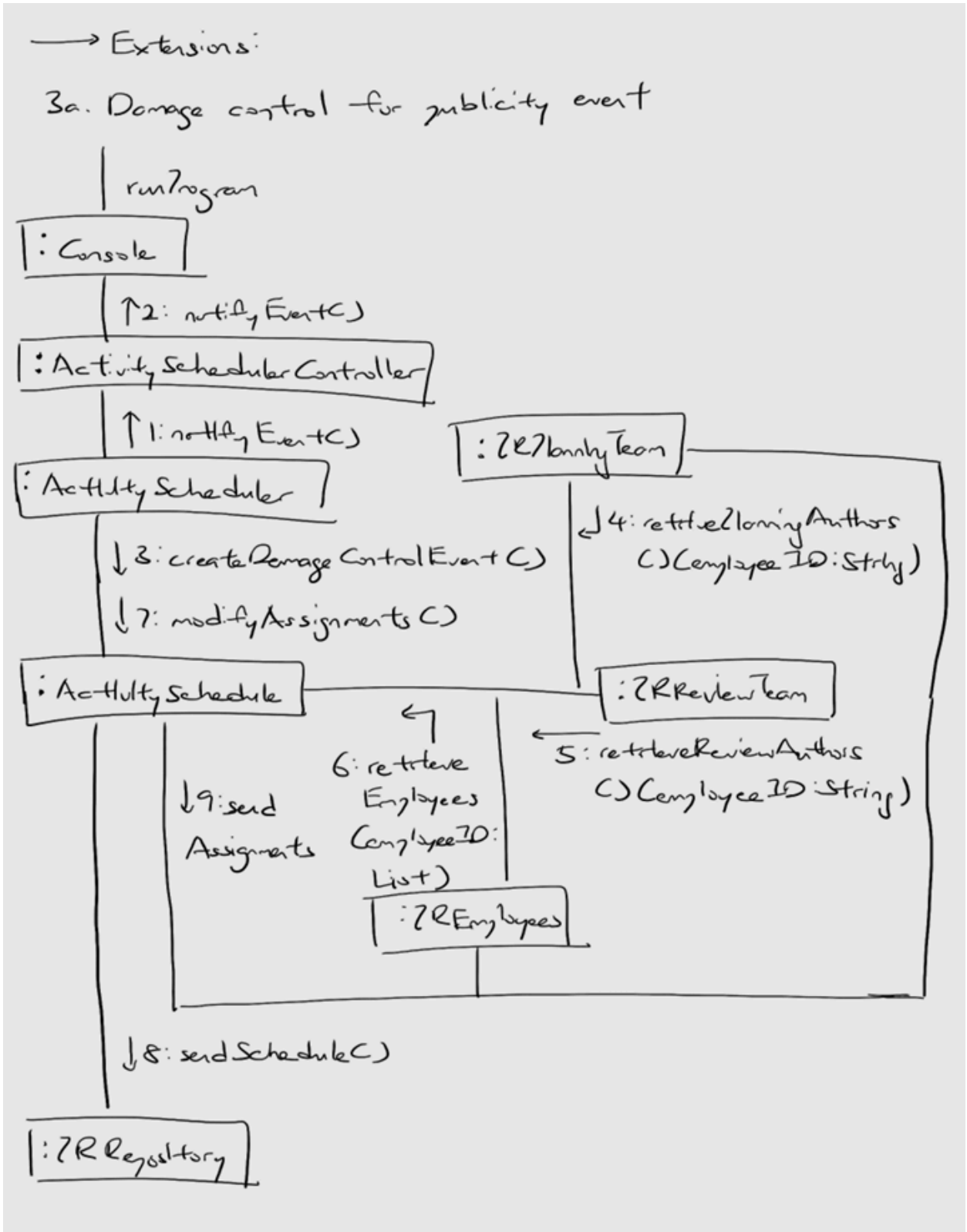
→ Extensions:

4a. Receives developed social content that requires revision.

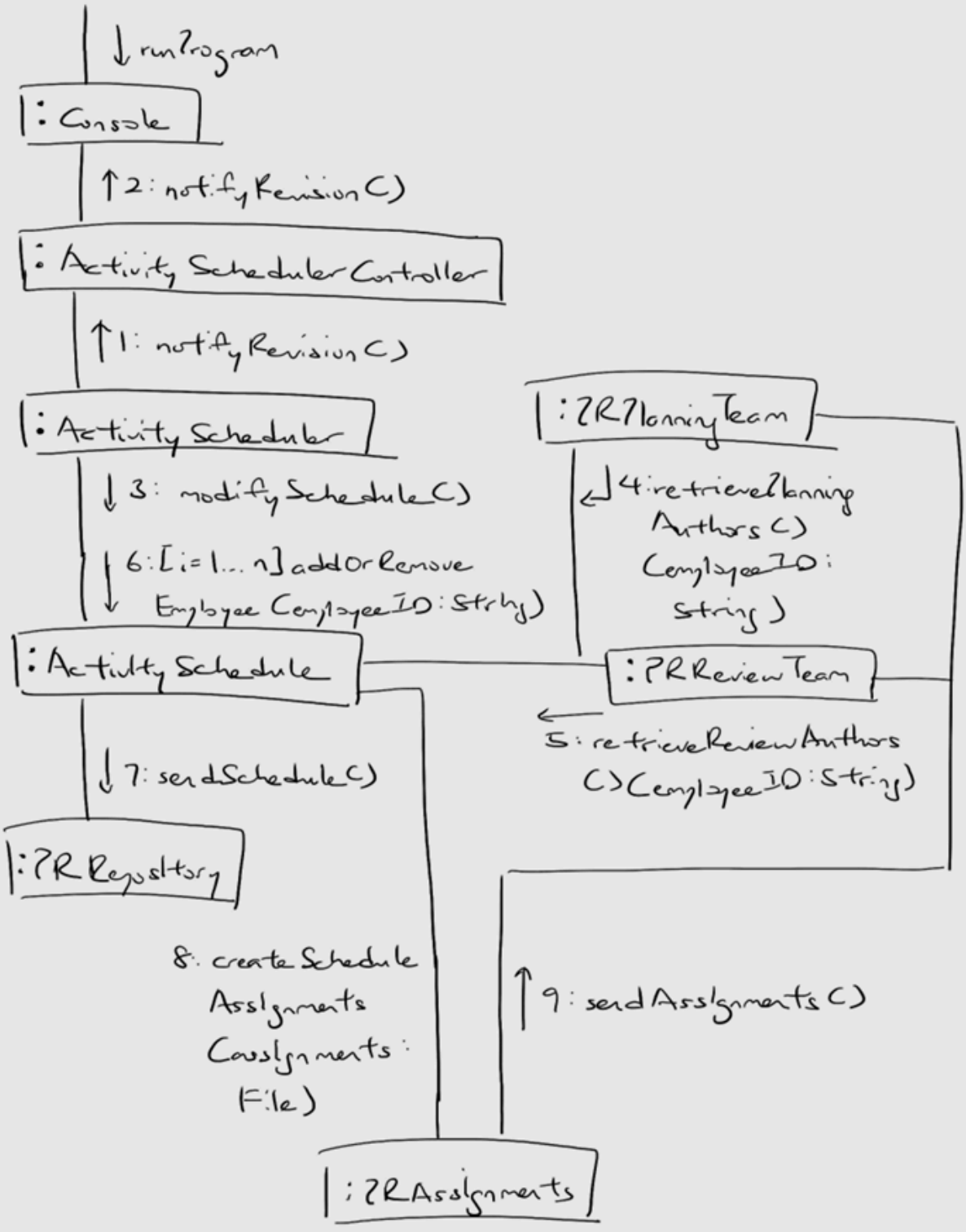



4b. Receives emergency content post from department
↓ runProgram
:Console
↑2: notifyEmergencyRequest()
:ContentSchedullerController
↑1: notifyEmergencyRequest()
:ContentScheduler
↓3: doScheduleOverride()
5: createEmergencySchedule
(assignment: File)
:ContentSchedule
:PRAssignments
↑4: sendAllEmployees
(allEmployees: List)
↓7: sendSchedule()
:PREmployees
:PRRepository
6: sendAssignments()

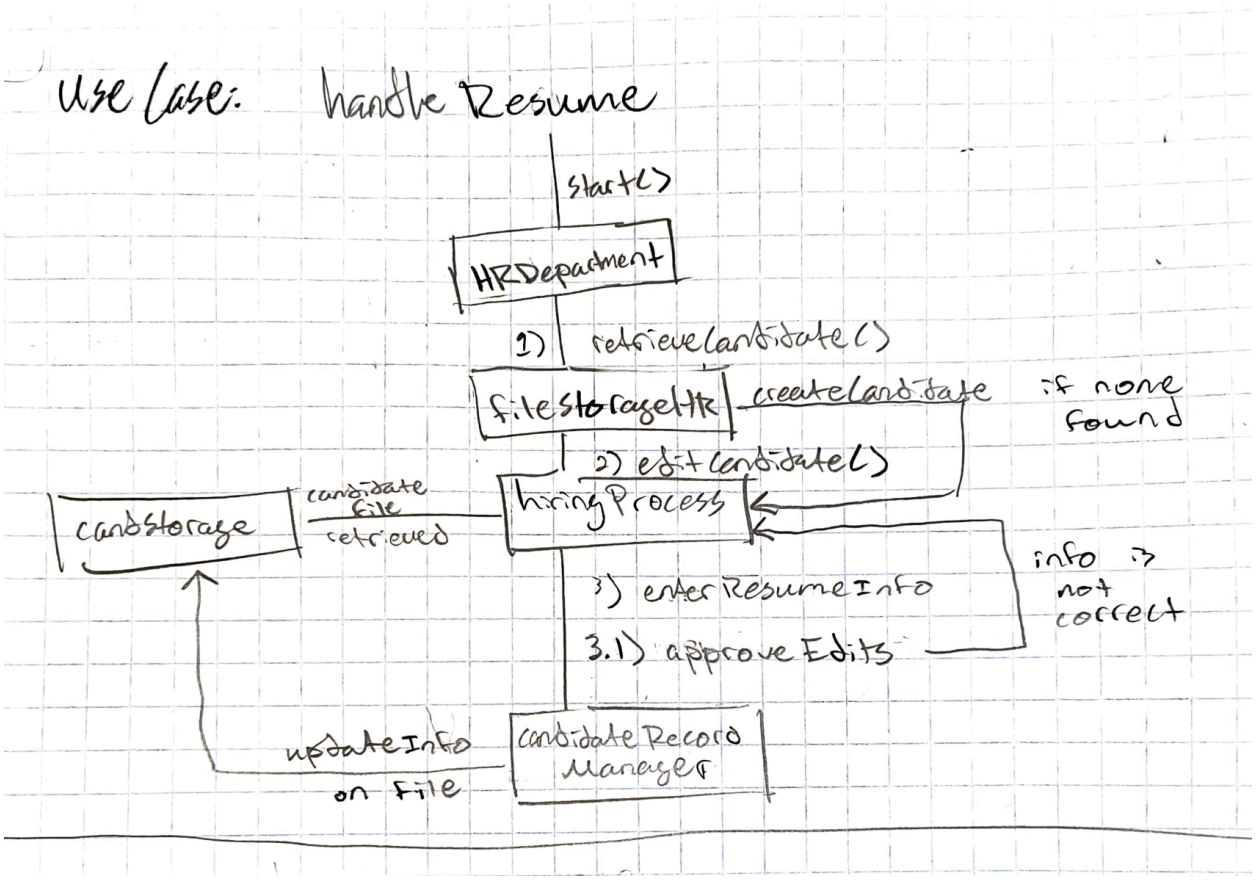

Use case: manageRublicityActivities (Kenny)
→ Main success scenario
| runProgram
|: Console
2: show Activity
↓3: [i=1...n] b = addActivity Schedule (schedule: Schedule) : boolean
↓4: [i=1...n] b = updateTREmployee (employee: Employee) : boolean
|: Activity Scheduler Controller
1: show Activity
↓5: [i=1...n] b = addActivity Schedule (schedule: Schedule) : boolean
↓6: [i=1...n] b = updateTREmployee (employee: Employee) : boolean
|: Activity Scheduler
↓7: [i=1...n] b = createActivity Scheduler (employeeID: Employee) : boolean
↓8: createActivity Schedule ()
10: createActivityAssignments (assignments: File)
|: Activity Schedule
|: Activity Assignments
↓9: sendActivity Schedule ()
↓11: sendAssignments ()
|: TR Repository
|: TR Employees

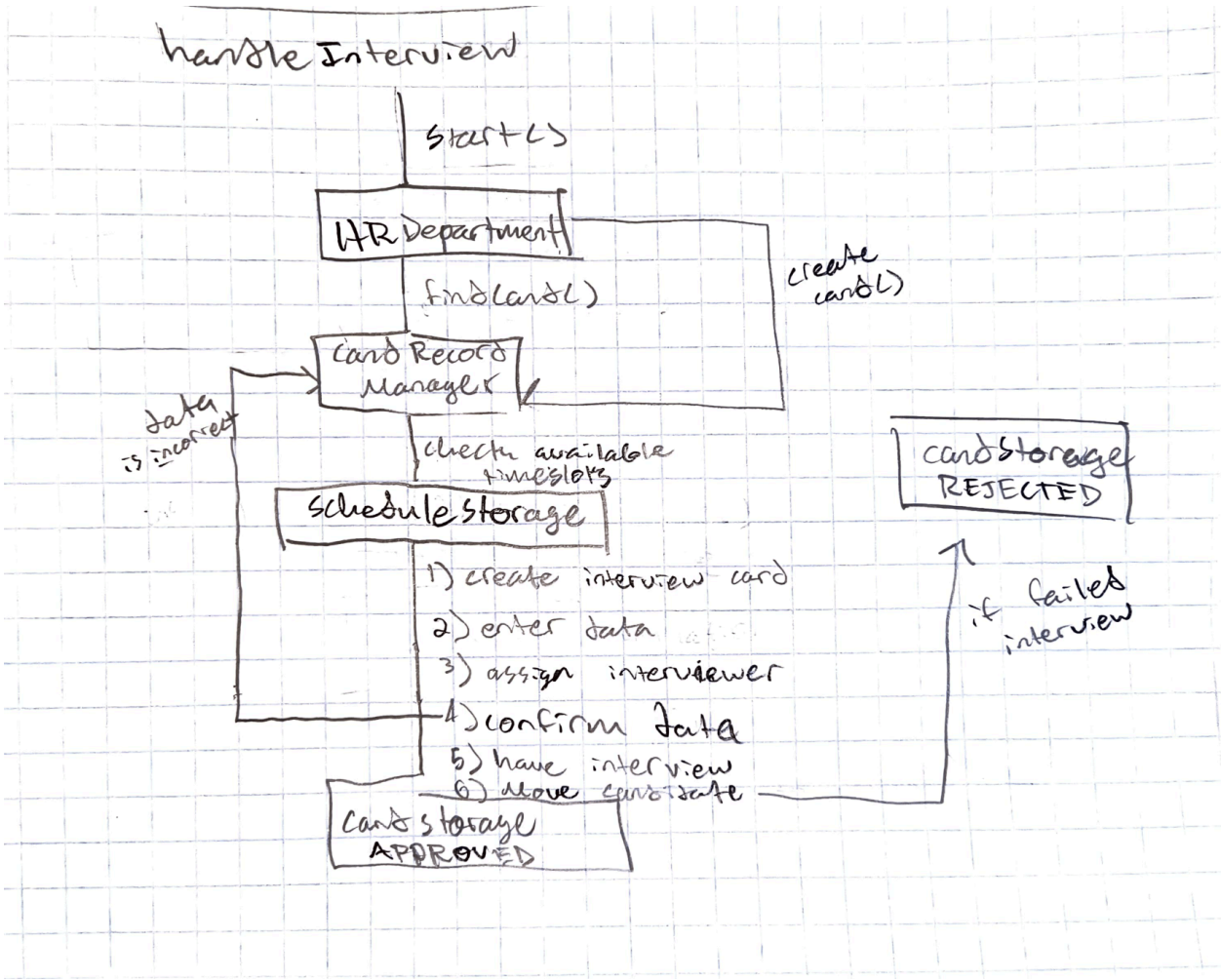

→ Extensions:
3a. Damage control for publicity event
runProgram
: Console
↑2: notifyEvent()
: ActivityScheduleController
↑1: notifyEvent()
: ActivityScheduler
: PRPlanningTeam
↓4: retrievePlanningAuthors
()(EmployeeID: String)
↓3: createDamageControlEvent()
↓7: modifyAssignments()
: ActivitySchedule
: PRReviewTeam
↖
6: retrieve
Employees
(EmployeeID:
List)
↖
5: retrieveReviewAuthors
()(EmployeeID: String)
↓9: send
Assignments
: PREmployees
↓8: sendSchedule()
: PRRepository

3b. Receives planned schedule event that needs revision



HR Department:




handleInterview
start()
HR Department
findCard()
createCard()
Card Record Manager
data is incorrect
checks available timeslots
ScheduleStorage
1) create interview card
2) enter data
3) assign interviewer
4) confirm data
5) have interview
6) move candidate
Card storage APPROVED
if failed interview
Card storage REJECTED

Implementation

Modeling Department:

```
public class Item implements IItem {
    static int nextid = 0;
    int id;

    Team associatedTeam;
    String itemType;
    String[] itemLocation;
    String itemName;
    RecurrenceType recurrenceType;

    boolean checkedOut;
    boolean damaged;


    Item(int i, Team associatedTeam, String itemType, String[]
itemLocation, String itemName, RecurrenceType recurrenceType,
boolean checkedOut, boolean damaged) {
        id = i;
        if (id >= nextid) nextid = id + 1;
        this.associatedTeam = associatedTeam;
        this.itemType = itemType;
        this.itemLocation = itemLocation;
        this.itemName = itemName;
        this.recurrenceType = recurrenceType;
        this.checkedOut = checkedOut;
        this.damaged = damaged;
    }

    public Item(Team associatedTeam, String itemType, String
itemName, RecurrenceType recurrenceType) {
        id = nextid;
        nextid++;
        this.associatedTeam = associatedTeam;
        this.itemType = itemType;
        this.itemLocation = new String[]{Integer.toString(id),
associatedTeam.toChar()};
        this.itemName = itemName;
        this.recurrenceType = recurrenceType;

        checkedOut = false;
        damaged = false;
    }

    @Override
    public int getId() {return id;}

    @Override
    public Team getAssociatedTeam() {return associatedTeam;}

    @Override
    public String getItemType() {return itemType;}

    @Override
    public String getItemName() {return itemName;}

    @Override
    public void setAssociatedTeam(Team team) {
        associatedTeam = team;
        this.itemLocation = new String[]{Integer.toString(id),
associatedTeam.toChar()};
    }

    @Override
    public void setType(String itemType) {this.itemType = itemType;}

    @Override
    public void setItemName(String name) {this.itemName = name;}
```

```
public class StorageManager extends Manager {
    Map<Team, Map<String, Item[]>> warehouse;

    public StorageManager(String name, Team team) {
        super(name, team);
        warehouse = fileManager.convertItemsToTeamCategoryMap();
    }

    public StorageManager(Team team) {
        super(team);
        warehouse = fileManager.convertItemsToTeamCategoryMap();
    }

    public StorageManager(int id, Employee employeeInfo, Team team) {
        super(id, employeeInfo, team);
        warehouse = fileManager.convertItemsToTeamCategoryMap();
    }

    public void addItem(Item item) throws IOException {
        fileManager.addItem(item);
    }

    public void updateItem(Item updatedItem, Team oldTeam) throws
IOException {
        fileManager.updateItem(updatedItem, oldTeam);
    }

    public void deleteItem(Team team, int itemID) {
        fileManager.deleteById(team, itemID);
    }

    public Boolean lookForDamages() {
        return null;
    }

    public Boolean requestOrder(Item item, LocalDate date) throws
IOException {
        LocalDate today = LocalDate.now();
        if (date.isAfter(today.plusDays(3))) {
            System.out.println("Requested item cannot be ordered sooner
than 3 days.");
            return false;
        }
        addItem(item);
        return true;
    }

    public void printWarehouse() {

    }

    public static StorageManager parse(Map<String, String> manager) {
        return new StorageManager(
                Integer.parseInt(manager.get("id")),
                Employee.parseEmployee(manager.get("employeeInfo")),
                Team.parseTeam(manager.get("team"))
        );
    }


}
```

```
package src.Modeling.src;

public enum RecurrenceType {
    DAILY,
```

```
    @Override
    public String getItemLocation() {return itemLocation[0] +
itemLocation[1];}

    @Override
    public void flagDamaged() {this.damaged = true;}

    @Override
    public void flagCheckedOut() {this.checkedOut = true;}

    @Override
    public void flagReturned() {this.checkedOut = false;}

    @Override
    public String toString() {
        return "\n" + this.associatedTeam.toString() + " Item: " + id +
                "\n Item Type: " + this.itemType +
                "\n Item Location: " + this.getItemLocation() +
                "\n Item Name: " + this.getItemName() +
                "\n Recurrence: " + this.recurrenceType.getRecurrence() +
                "\n Checked Out: " + this.checkedOut +
                "\n Damaged: " + this.damaged;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> itemDetails = new HashMap<>();
        itemDetails.put("id", Integer.toString(this.id));
        itemDetails.put("associatedTeam", this.associatedTeam.toString());
        itemDetails.put("itemType", this.itemType);
        itemDetails.put("itemLocation", Arrays.toString(this.itemLocation));
        itemDetails.put("itemName", this.itemName);
        itemDetails.put("recurrenceType", this.recurrenceType.toString());
        itemDetails.put("checkedOut", Boolean.toString(this.checkedOut));
        itemDetails.put("damaged", Boolean.toString(this.damaged));
        return itemDetails;
    }

    public static Item parse(Map<String, String> item) {
        return new Item(
                Integer.parseInt(item.get("id")),
                Team.parseTeam(item.get("associatedTeam")),
                item.get("itemType"),
                item.get("itemLocation").replaceAll("[\\[\\] ]", "").split(","),
                item.get("itemName"),
                RecurrenceType.parseType(item.get("recurrenceType")),
                Boolean.parseBoolean(item.get("checkedOut")),
                Boolean.parseBoolean(item.get("damaged"))
        );
    }
}
```

```
    private Map<String, Map<String, String>> getTargetMap(Team team)
{
        switch (team) {
            case CLOTHING:
                return clothingItems;
            case MAKEUP:
                return makeupItems;
            case MODELING:
                return modelingItems;
            default:
                throw new IllegalArgumentException("Unknown team: " +
team);
        }
    }

    public Map<Team, Map<String, Item[]>>
convertItemsToTeamCategoryMap() {
        Map<Team, Map<String, Item[]>> convertedMap = new
HashMap<>();

        for (Team team : Team.values()) {
            if(team.equals(Team.STORAGE)) continue;
            Map<String, Map<String, String>> itemTypeMap =
getItemMapForTeam(team);
```

```
    WEEKLY,
    MONTHLY,
    YEARLY,
    NONE;

    public String toString() {
        return switch (this) {
            case DAILY -> "DAILY";
            case WEEKLY -> "WEEKLY";
            case MONTHLY -> "MONTHLY";
            case YEARLY -> "YEARLY";
            case NONE -> "NONE";
        };
    }

    public String getRecurrence() {
        return switch (this) {
            case DAILY -> "Daily at 9am";
            case WEEKLY -> "Weekly on Mondays at 9am";
            case MONTHLY -> "First of the Month at 9am";
            case YEARLY -> "January 1st at 9am";
            case NONE -> "Never";
        };
    }

    public static RecurrenceType parseType(String type) {
        return switch (type) {
            case "DAILY", "daily", "Daily", "d", "D" -> DAILY;
            case "WEEKLY", "weekly", "Weekly", "w", "W" -> WEEKLY;
            case "MONTHLY", "monthly", "Monthly", "m", "M" -> MONTHLY;
            case "YEARLY", "yearly", "Yearly", "y", "Y" -> YEARLY;
            case "NONE", "none", "None", "n", "N" -> NONE;
            default -> throw new IllegalArgumentException();
        };
    }
}
```

```
private void initiateStorageManager() throws IOException {
        int userChoice = 0;
        while (userChoice != 7) {
            System.out.println("""

                    Please choose an action you want to take:
                     1: orderItem
                     2: updateItem
                     3: deleteItem
                     4: damageScan
                     5: returnItem
                     6: printItems
                     7: Back
                    """);
            userChoice = s.nextInt();
            s.nextLine();
            switch (userChoice) {
                case 1 -> {
                    Team team = null;
                    String type = "";
                    String name = "";
                    RecurrenceType recurrenceType = null;

                    LocalDate date = LocalDate.now();
                    DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM/dd/yyyy");

                    System.out.println("When do you need the item
by?(mm/dd/yyyy)");
                    date = LocalDate.parse(s.nextLine(), formatter);
                    if(date.isBefore(LocalDate.now().plusDays(3))) {
                        System.out.println("Cannot order an item sooner than 3
days, please move event or go without.");
                        continue;
                    }

                    int teamNumber = 10;
                    while (teamNumber > 3) {
                        System.out.println("""

                                What team are you requesting the item for
```

```
        Map<String, List<Item>> categoryItemMap = new HashMap<>();

        List<Item> itemList = new ArrayList<>();
        for (Map.Entry<String, Map<String, String>> itemTypeEntry :
itemTypeMap.entrySet()) {
            String category = itemTypeEntry.getKey();
            Map<String, String> itemDetailsMap =
itemTypeEntry.getValue();

            Item item = Item.parse(itemDetailsMap);
            itemList.add(item);

            categoryItemMap.put(category, itemList);
        }

        Map<String, Item[]> categoryItemArrayMap = new
HashMap<>();
        for (Map.Entry<String, List<Item>> entry :
categoryItemMap.entrySet()) {
            categoryItemArrayMap.put(entry.getKey(),
entry.getValue().toArray(new Item[0]));
        }

        convertedMap.put(team, categoryItemArrayMap);
    }

    return convertedMap;
  }

  private Map<String, Map<String, String>> getItemMapForTeam(Team
team) {
    switch (team) {
      case CLOTHING:
        return clothingItems;
      case MAKEUP:
        return makeupItems;
      case MODELING:
        return modelingItems;
      default:
        return new HashMap<>();
    }
  }

  public StorageManager getStorageManager() {
    return StorageManager.parse(teamMembers.get("Storage
Manager"));
  }

  public void setItems(Team team) {
    String category = team.name().toLowerCase();
    File folder = new File(repo + "warehouse/" + category);
    String[] files = folder.list();

    if (files != null) {
      for (String fileName : files) {
        editor.processTextFile(folder.getPath() + "/" + fileName);
        Map<String, Map<String, String>> itemMap =
editor.getRepositoryStringMap();

        getTargetMap(team).putAll(itemMap);
      }
    }
  }

  public void addItem(Item item) throws IOException {
    Team team = item.getAssociatedTeam();
    String category = team.name().toLowerCase();
    File file = new File(repo + "warehouse/" + category + "/" +
item.getItemType() + ".txt");

    if (!file.exists()) {
      Scanner scanner = new Scanner(System.in);
      System.out.println("Type " + item.getItemType() + " is not found
in the system. Would you like to create it? (y/n)");
      String input = scanner.nextLine();
      if (input.equalsIgnoreCase("y")) {
        if (file.createNewFile()) {
          System.out.println("File created successfully: " +
file.getAbsolutePath());
            1: Modeling
            2: Makeup
            3: Clothing
          """);
        teamNumber = s.nextInt();
        s.nextLine();
        switch (teamNumber) {
          case 1 -> {
            System.out.println("Is Modeling correct? (y/n)");
            String tmp = s.nextLine();
            if (tmp.equals("y")) {
              team = Team.MODELING;
            } else {
              teamNumber = 10;
            }
          }
          case 2 -> {
            System.out.println("Is Makeup correct? (y/n)");
            String tmp = s.nextLine();
            if (tmp.equals("y")) {
              team = Team.MAKEUP;
            } else {
              teamNumber = 10;
            }
          }
          case 3 -> {
            System.out.println("Is Clothing correct? (y/n)");
            String tmp = s.nextLine();
            if (tmp.equals("y")) {
              team = Team.CLOTHING;
            } else {
              teamNumber = 10;
            }
          }
        }
      }

      boolean verified = false;
      while (!verified) {
        System.out.println("What is the name of the item?");
        name = s.nextLine();
        System.out.println("Is " + name + " correct?(y/n)");
        String tmp = s.nextLine();
        if (tmp.equals("y")) {
          verified = true;
        }
      }

      verified = false;
      while (!verified) {
        assert team != null;
        System.out.println("What category is the item?\n"+
Arrays.toString(fileManager.getCategories(team)));
        type = s.nextLine();
        System.out.println("Is " + type + " correct?(y/n)");
        String tmp = s.nextLine();
        if (tmp.equals("y")) {
          verified = true;
        }
      }

      int recurrence = 10;
      while (recurrence > 5) {
        System.out.println("""

          Will you add a recurrence flag to this item?
            1: Daily
            2: Weekly
            3: Monthly
            4: Yearly
            5: None
          """);
        recurrence = s.nextInt();
        s.nextLine();
        switch (recurrence) {
          case 1 -> recurrenceType = RecurrenceType.DAILY;
          case 2 -> recurrenceType =
RecurrenceType.WEEKLY;
```

```
            } else {
                System.out.println("Failed to create the file.");
                return;
            }
        } else {
            return;
        }
    }

    Map<String, Map<String, String>> targetMap =
getTargetMap(team);

    targetMap.put("Item " + item.getId(), item.toMap());

    editor.setRepositoryStrings(targetMap);
    editor.writeToTextFile(file.getPath());
  }

  public List<Integer> getAllItemIdsForTeam(Team team) {
    List<Integer> itemIds = new ArrayList<>();

    // Retrieve the map for the team
    Map<String, Map<String, String>> itemMap = getTargetMap(team);

    // Iterate through all items and extract their IDs
    for (Map<String, String> itemDetails : itemMap.values()) {
      if (itemDetails.containsKey("id")) {
        int itemId = Integer.parseInt(itemDetails.get("id"));
        itemIds.add(itemId);
      }
    }

    return itemIds; // Return the list of item IDs
  }

  public Item getItemById(Team team, int itemId) {
    Map<String, Map<String, String>> itemMap = getTargetMap(team);

    for (Map<String, String> itemDetails : itemMap.values()) {
      if (itemDetails.containsKey("id") &&
Integer.parseInt(itemDetails.get("id")) == itemId) {
        return Item.parse(itemDetails);
      }
    }

    return null; // Return null if not found
  }

  public List<Item> findDamagedItems() {
    List<Item> damagedItems = new ArrayList<>();

    for (Team team : Team.values()) {
      if(team.equals(Team.STORAGE)) continue;
      Map<String, Map<String, String>> itemMap =
getTargetMap(team);

      for (Map<String, String> itemDetails : itemMap.values()) {
        if ("true".equalsIgnoreCase(itemDetails.get("damaged"))) {
          damagedItems.add(Item.parse(itemDetails)); // Add the
damaged item
        }
      }
    }

    return damagedItems;
  }

  public List<Integer> findCheckedOutIdsItems(Team team) {
    List<Integer> checkedOutItems = new ArrayList<>();
    Map<String, Map<String, String>> itemMap = getTargetMap(team);

    for (Map<String, String> itemDetails : itemMap.values()) {
      if ("true".equalsIgnoreCase(itemDetails.get("checkedOut"))) {
        checkedOutItems.add(Item.parse(itemDetails).getId()); // Add
the damaged item
      }
    }

    return checkedOutItems;
                case 3 -> recurrenceType =
RecurrenceType.MONTHLY;
                case 4 -> recurrenceType =
RecurrenceType.YEARLY;
                default -> recurrenceType = RecurrenceType.NONE;
              }
              System.out.println("Recurring " +
recurrenceType.getRecurrence() + "? (y/n)");
              String tmp = s.nextLine();
              if (!tmp.equals("y")) {
                recurrence = 10;
              }
            }

            Item newItem = new Item(team, type, name,
recurrenceType);

            storageManager.addItem(newItem);
            System.out.println(newItem);
          }
          case 2 -> {
            Team team = null;

            int teamNumber = 10;
            while (teamNumber > 3) {
              System.out.println("""

                  What team are you updating the item for
                   1: Modeling
                   2: Makeup
                   3: Clothing
                  """);
              teamNumber = s.nextInt();
              s.nextLine();
              switch (teamNumber) {
                case 1 -> {
                  System.out.println("Is Modeling correct? (y/n)");
                  String tmp = s.nextLine();
                  if (tmp.equals("y")) {
                    team = Team.MODELING;
                  } else {
                    teamNumber = 10;
                  }
                }
                case 2 -> {
                  System.out.println("Is Makeup correct? (y/n)");
                  String tmp = s.nextLine();
                  if (tmp.equals("y")) {
                    team = Team.MAKEUP;
                  } else {
                    teamNumber = 10;
                  }
                }
                case 3 -> {
                  System.out.println("Is Clothing correct? (y/n)");
                  String tmp = s.nextLine();
                  if (tmp.equals("y")) {
                    team = Team.CLOTHING;
                  } else {
                    teamNumber = 10;
                  }
                }
              }
            }

            List<Integer> ids =
fileManager.getAllItemIdsForTeam(team);
            int id = 0;

            System.out.println("What item are you updating?\n "+ ids);
            id = s.nextInt();
            s.nextLine();

            Item item = fileManager.getItemById(team, id);
            Team oldTeam = item.getAssociatedTeam();
            System.out.println(item.toString());
            System.out.println("""

                What part of the item are you updating?
```

```
    }

    public void updateItem(Item item, Team oldTeam) throws IOException
{
        Team newTeam = item.getAssociatedTeam();

        // Delete the item from the old team
        boolean deleted = deleteById(oldTeam, item.getId());
        if (!deleted) {
            System.out.println("Failed to delete item from old team.");
            return;
        }

        // Define the file paths
        String newFilePath = repo + "warehouse/" +
newTeam.name().toLowerCase() + "/" + item.getItemType() + ".txt";

        // Create new file if it doesn't exist
        File newFile = new File(newFilePath);
        if (!newFile.exists()) {
            new File(newFile.getParent()).mkdirs(); // Create directories if
necessary
            newFile.createNewFile();
        }

        // Add the updated item to the new team
        Map<String, Map<String, String>> targetMap =
getTargetMap(newTeam);
        targetMap.put("Item " + item.getId(), item.toMap());

        // Update the file for the new team
        editor.setRepositoryStrings(targetMap);
        editor.writeToTextFile(newFile.getPath());

        System.out.println("Item updated successfully.");
    }

    public String[] getCategories(Team team) {
        File f = new File(repo + "warehouse/" + team.toString());
        String[] files = f.list();
        if (files != null) {
            for (int i = 0; i < files.length; i++) {
                int dotIndex = files[i].lastIndexOf('.');
                if (dotIndex > 0) {
                    files[i] = files[i].substring(0, dotIndex);
                }
            }
        }
        return files;
    }

    public boolean deleteById(Team team, int itemId) {
        Map<String, Map<String, String>> targetMap =
getTargetMap(team);

        Iterator<Map.Entry<String, Map<String, String>>> iterator =
targetMap.entrySet().iterator();
        while (iterator.hasNext()) {
            Map.Entry<String, Map<String, String>> entry = iterator.next();
            Map<String, String> itemDetails = entry.getValue();

            if (itemDetails.containsKey("id") &&
Integer.parseInt(itemDetails.get("id")) == itemId) {
                iterator.remove();

                // Update the corresponding text file
                String category = team.name().toLowerCase();
                File file = new File(repo + "warehouse/" + category + "/" +
itemDetails.get("itemType") + ".txt");

                if (targetMap.isEmpty() && file.exists()) {
                    if (file.delete()) {
                        System.out.println("File deleted successfully: " +
file.getAbsolutePath());
                    } else {
                        System.out.println("Failed to delete the file.");
                    }
                } else {
                    editor.setRepositoryStrings(targetMap);
                                1: Name
                                2: Type
                                3: Team
                                4: Recurrence
                                5: set Damaged
                                6: Check out
                                """);
                    int choice = s.nextInt();
                    s.nextLine();

                    switch (choice) {
                        case 1 -> {
                            System.out.println("What is the new value of Name?");
                            item.setItemName(s.nextLine());
                        }
                        case 2 -> {
                            System.out.println("What is the new value of Type?");
                            item.setType(s.nextLine());
                        }
                        case 3 -> {
                            System.out.println("What is the new value of Team?
[Modeling, Makeup, Clothing]");

item.setAssociatedTeam(Team.parseTeam(s.nextLine()));
                        }
                        case 4 -> {
                            System.out.println("What is the new value of
Recurrence? [DAILY, WEEKLY, MONTHLY, YEARLY, NONE]");

item.setAssociatedTeam(Team.parseTeam(s.nextLine()));
                        }
                        case 5 -> {
                            item.flagDamaged();
                        }
                        case 6 -> {
                            item.flagCheckedOut();
                        }
                    }

                    storageManager.updateItem(item, oldTeam);
                    System.out.println(item);

                }
                case 3 -> {
                    Team team = null;

                    int teamNumber = 10;
                    while (teamNumber > 3) {
                        System.out.println("""

                                What team are you deleting the item for
                                 1: Modeling
                                 2: Makeup
                                 3: Clothing
                                """);
                        teamNumber = s.nextInt();
                        s.nextLine();
                        switch (teamNumber) {
                            case 1 -> {
                                System.out.println("Is Modeling correct? (y/n)");
                                String tmp = s.nextLine();
                                if (tmp.equals("y")) {
                                    team = Team.MODELING;
                                } else {
                                    teamNumber = 10;
                                }
                            }
                            case 2 -> {
                                System.out.println("Is Makeup correct? (y/n)");
                                String tmp = s.nextLine();
                                if (tmp.equals("y")) {
                                    team = Team.MAKEUP;
                                } else {
                                    teamNumber = 10;
                                }
                            }
                            case 3 -> {
                                System.out.println("Is Clothing correct? (y/n)");
                                String tmp = s.nextLine();
```

```
                editor.writeToTextFile(file.getPath());
            }
            return true;
        }
    }

    System.out.println("Item with ID " + itemId + " not found in " +
team.name() + ".");
    return false;
  }

  public void printItems() {
    System.out.println("Clothing Items:");
    printCategoryItems(clothingItems);

    System.out.println("Makeup Items:");
    printCategoryItems(makeupItems);

    System.out.println("Modeling Items:");
    printCategoryItems(modelingItems);
  }

  private void printCategoryItems(Map<String, Map<String, String>>
categoryItems) {
    for (Map.Entry<String, Map<String, String>> entry :
categoryItems.entrySet()) {
      System.out.println("  Item: " + entry.getKey());
      for (Map.Entry<String, String> detail :
entry.getValue().entrySet()) {
        System.out.println("    " + detail.getKey() + ": " +
detail.getValue());
      }
    }
  }
```

```
package src.Modeling.src;

public enum Team {
  MODELING,
  MAKEUP,
  CLOTHING,
  STORAGE;

  public String toString() {
    return switch (this) {
      case MODELING -> "Modeling";
      case MAKEUP -> "Makeup";
      case CLOTHING -> "Clothing";
      case STORAGE -> "Storage";
    };
  }

  public String toChar() {
    return switch (this) {
      case MODELING -> "Mo";
      case MAKEUP -> "Ma";
      case CLOTHING -> "C";
      case STORAGE -> "S";
    };
  }

  public static Team parseTeam(String team) {
    return switch (team) {
      case "Modeling" -> MODELING;
      case "Makeup" -> MAKEUP;
      case "Clothing" -> CLOTHING;
      case "Storage" -> STORAGE;
      default -> throw new IllegalArgumentException();
    };
  }
}
```

```
                if (tmp.equals("y")) {
                    team = Team.CLOTHING;
                } else {
                    teamNumber = 10;
                }
            }
        }

        List<Integer> ids =
fileManager.getAllItemIdsForTeam(team);
        int id = 0;

        System.out.println("What item are you deleting?\n "+
ids);
        id = s.nextInt();
        s.nextLine();

        Item item = fileManager.getItemById(team, id);
        System.out.println(item.toString());

        System.out.println("Are you sure you want to delete this
item?(y/n)");
        if(s.nextLine().equals("y")){
            storageManager.deleteItem(team, id);
        }
      }
    }
    case 4 -> {
      List<Item> damagedItems =
fileManager.findDamagedItems();
      if(damagedItems.isEmpty()) {
        System.out.println("No damaged items found!");
      } else {
        System.out.println(damagedItems.size() + " damaged
item(s) found, would you like to remove them?(y/n)");
        if(s.nextLine().equals("y")) {
          for(Item item: damagedItems) {
            System.out.println("Deleting Item " + item.getId());
            fileManager.deleteById(item.getAssociatedTeam(),
item.getId());
          }
        }
      }
    }
    case 5 -> {
      Team team = null;
      int teamNumber = 10;
      while (teamNumber > 3) {
        System.out.println("""

            What team are you requesting the item for
             1: Modeling
             2: Makeup
             3: Clothing
            """);
        teamNumber = s.nextInt();
        s.nextLine();
        switch (teamNumber) {
          case 1 -> {
            System.out.println("Is Modeling correct? (y/n)");
            String tmp = s.nextLine();
            if (tmp.equals("y")) {
              team = Team.MODELING;
            } else {
              teamNumber = 10;
            }
          }
          case 2 -> {
            System.out.println("Is Makeup correct? (y/n)");
            String tmp = s.nextLine();
            if (tmp.equals("y")) {
              team = Team.MAKEUP;
            } else {
              teamNumber = 10;
            }
          }
          case 3 -> {
            System.out.println("Is Clothing correct? (y/n)");
            String tmp = s.nextLine();
```

```
                        if (tmp.equals("y")) {
                            team = Team.CLOTHING;
                        } else {
                            teamNumber = 10;
                        }
                    }
                }
            }

            List<Integer> checkedOutItems =
fileManager.findCheckedOutIdsItems(team);
            if(checkedOutItems.isEmpty()) {
                System.out.println("No checked out items for "+ team);
                continue;
            }
            Item item = null;
            while (item == null) {
                System.out.println("Which item would you like to
return?\n" + checkedOutItems);
                item = fileManager.getItemById(team, s.nextInt());
                s.nextLine();

                System.out.println("Is the item damaged?(y/n)");
                if(s.nextLine().equals("y")) item.flagDamaged();
                item.flagReturned();

                System.out.println(item);
                storageManager.updateItem(item,
item.getAssociatedTeam());
            }
        }
        case 6 -> {
            fileManager.printItems();
        }
    }
  }
```

Marketing Department:

```
public class Celebrity implements ICollabMember {
    static int nextid = 0;
    int id;

    String name;
    ArrayList<Integer> advertisements;

    Employee tempEmployeeInfo;

    public Celebrity(int i, String name, ArrayList<Integer> advertisements,
Employee employee) {
        this.id = i;
        if (id >= nextid) nextid = id + 1;
        this.name = name;
        this.advertisements = advertisements;
        this.tempEmployeeInfo = employee;
    }

    public Celebrity(String name, ArrayList<Integer> advertisements) {
        id = nextid;
        nextid++;
        this.name = name;
        this.advertisements = advertisements;
        this.tempEmployeeInfo = new Employee("Celebrity", name,
Department.MODELING, "Temp Worker", "Contract", 1000000);

        MarketingDepartment.fileManager.addCollabMember(this);
    }

    public Celebrity(String name) {
```

```
public class Brand implements ICollabMember {
    static int nextid = 0;
    int id;

    String name;
    ArrayList<Integer> advertisements;

    public Brand(int i, String name, ArrayList<Integer> advertisements) {
        this.id = i;
        if (id >= nextid) nextid = id + 1;
        this.name = name;
        this.advertisements = advertisements;
    }

    public Brand(String name, ArrayList<Integer> advertisements) {
        id = nextid;
        nextid++;
        this.name = name;
        this.advertisements = advertisements;

        MarketingDepartment.fileManager.addCollabMember(this);
    }

    public Brand(String name) {
        id = nextid;
        nextid++;
        this.name = name;
        this.advertisements = new ArrayList<>();

        MarketingDepartment.fileManager.addCollabMember(this);
```

```
        id = nextid;
        nextid++;
        this.name = name;
        this.advertisements = new ArrayList<>();
        this.tempEmployeeInfo = new Employee("Celebrity", name,
Department.MODELING, "Temp Worker", "Contract", 1000000);

        MarketingDepartment.fileManager.addCollabMember(this);
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public String getName() {
        return name;
    }

    public String getEmployeeInfo() {
        return this.tempEmployeeInfo.toString();
    }

    @Override
    public void addAdvertisement(int id) {
        this.advertisements.add(id);
    }

    @Override
    public void removeAdvertisement(Integer id) {
        this.advertisements.remove(id);
    }

    @Override
    public void changeName(String name) {
        this.name = name;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> brandDetails = new HashMap<>();
        brandDetails.put("id", Integer.toString(this.id));
        brandDetails.put("name", this.name);
        brandDetails.put("advertisements", this.advertisements.toString());
        brandDetails.put("contractDetail", this.getEmployeeInfo());
        return brandDetails;
    }

    public static Celebrity parse(Map<String, String> celeb) {
        String[] elements = celeb.get("advertisements").replaceAll("[\\[\\] ]",
"").split(",");
        ArrayList<Integer> list = new ArrayList<>();
        for (String element : elements) {
            list.add(Integer.parseInt(element));
        }

        return new Celebrity(
                Integer.parseInt(celeb.get("id")),
                celeb.get("name"),
                list,
                Employee.parseEmployee(celeb.get("contractDetail"))
        );
    }

}
```

```
public class FileManager {
    private Map<String, Map<String, String>> teamMembers = new
HashMap<>();
    private Map<String, Map<String, String>> eventAdverts = new
HashMap<>();
    private Map<String, Map<String, String>> designAdverts = new
HashMap<>();
    private Map<String, Map<String, String>> approvedCollabs = new
HashMap<>();
    private Map<String, Map<String, String>> collaborations = new
HashMap<>();
```

```
    }

    @Override
    public int getId() {
        return id;
    }

    @Override
    public String getName() {
        return name;
    }

    @Override
    public void addAdvertisement(int id) {
        this.advertisements.add(id);
    }

    @Override
    public void removeAdvertisement(Integer id) {
        this.advertisements.remove(id);
    }

    @Override
    public void changeName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        String str = "\nCollab Member " + id + ":" + "\n Name: " +
getName() +
                "\n Advertisements: " + advertisements;
        return str;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> brandDetails = new HashMap<>();
        brandDetails.put("id", Integer.toString(this.id));
        brandDetails.put("name", this.name);
        brandDetails.put("advertisements", this.advertisements.toString());
        return brandDetails;
    }

    public static Brand parse(Map<String, String> brand) {
        String[] elements = brand.get("advertisements").replaceAll("[\\[\\] ]",
"").split(",");
        ArrayList<Integer> list = new ArrayList<>();
        for (String element : elements) {
            list.add(Integer.parseInt(element));
        }

        return new Brand(
                Integer.parseInt(brand.get("id")),
                brand.get("name"),
                list
        );
    }
}
```

```
public class HOD implements IHOD {

    private final Employee employeeInfo;
    private final ArrayList<Manager> managers;

    public ArrayList<EventAdvertisement> eventAdverts;
    public ArrayList<DesignAdvertisement> designAdverts;

    public ArrayList<ICollabMember> approvedCollabMembers;
    public ArrayList<ICollab> collabs;


    public HOD(Employee employeeInfo, ArrayList<Manager>
managers) {
        this.employeeInfo = employeeInfo;
        this.managers = managers;

        eventAdverts =
```

```
    private final PoorTextEditor editor = new PoorTextEditor();

    private final String repo = "src/Marketing/repository/";

    public FileManager() {
        fillRepos();
    }

    public void fillRepos() {
        File f = new File(repo + "department.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "department.txt");
            teamMembers = editor.getRepositoryStringMap();
        }

        f = new File(repo + "eventAdvert.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "eventAdvert.txt");
            eventAdverts = editor.getRepositoryStringMap();
        }

        f = new File(repo + "designAdvert.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "designAdvert.txt");
            designAdverts = editor.getRepositoryStringMap();
        }

        f = new File(repo + "collaborations/approved.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "collaborations/approved.txt");
            approvedCollabs = editor.getRepositoryStringMap();
        }

        f = new File(repo + "collaborations/collaborations.txt");
        if(f.exists()) {
            editor.processTextFile(repo + "collaborations/collaborations.txt");
            collaborations = editor.getRepositoryStringMap();
        }
    }

//    BrandCollaborations
    public ArrayList<ICollabMember> getApprovedCollabMembers() {
        ArrayList<ICollabMember> tmp = new ArrayList<>();
        for(Map<String, String> advert: eventAdverts.values()) {
            tmp.add(ICollabMember.parse(advert));
        }
        return tmp;
    }

    public void addCollabMember(ICollabMember collabMember) {
        approvedCollabs.put("Member " + collabMember.getId(),
collabMember.toMap());

        editor.setRepositoryStrings(approvedCollabs);
        editor.writeToTextFile(repo + "collaborations/approved.txt");
    }

    //    EventsAdverts:
    public void addEventAdvert(EventAdvertisement advert) {
        eventAdverts.put("Advert " + advert.getId(), advert.toMap());

        editor.setRepositoryStrings(eventAdverts);
        editor.writeToTextFile(repo + "eventAdvert.txt");
    }

    public ArrayList<EventAdvertisement> getEventAdverts() {
        ArrayList<EventAdvertisement> tmp = new ArrayList<>();
        for(Map<String, String> advert: eventAdverts.values()) {
            tmp.add(EventAdvertisement.parse(advert));
        }
        return tmp;
    }

    public EventAdvertisement getEventAdvertById(int id) {
        String key = "Advert " + id;
        Map<String, String> advertDetails = eventAdverts.get(key);
        return (advertDetails != null) ?
EventAdvertisement.parse(advertDetails) : null;
MarketingDepartment.fileManager.getEventAdverts();
        designAdverts =
MarketingDepartment.fileManager.getDesignAdverts();

        approvedCollabMembers =
MarketingDepartment.fileManager.getApprovedCollabMembers();
    }

    public HOD() {
        employeeInfo = new Employee("hod", "Head of Marketing",
Department.MARKETING, "HOD", "Employeed", 100000);
        managers = new ArrayList<>();
        managers.add(new Manager(Team.EDITING));
        managers.add(new Manager(Team.SOCIALMEDIA));
        managers.add(new Manager(Team.VIDEO));
        managers.add(new Manager(Team.DISTRIBUTION));

        MarketingDepartment.fileManager.addHOD(this);
    }

//    Get methods
    @Override
    public int getId() {
        return 0;
    }

    @Override
    public Employee getEmployeeInfo() {
        return employeeInfo;
    }

    @Override
    public ArrayList<EventAdvertisement> getEventAdverts() {return
eventAdverts;}

    @Override
    public ArrayList<DesignAdvertisement> getDesignAdverts() {return
designAdverts;}

    @Override
    public ArrayList<ICollabMember> getApprovedCollabMembers()
{return approvedCollabMembers;}

//    Creation
    @Override
    public EventAdvertisement createEventAdvert(Event event,
AdvertType type) {
        EventAdvertisement ad = new EventAdvertisement(event, type);
        eventAdverts.add(ad);
        return ad;
    }

    @Override
    public DesignAdvertisement createDesignAdvert(AdvertType type,
String notes) {
        DesignAdvertisement ad = new DesignAdvertisement(type, notes);
        designAdverts.add(ad);
        return ad;
    }

    @Override
    public ICollabMember addApprovedCollab(ICollabMember member) {
        approvedCollabMembers.add(member);
        return member;
    }

    @Override
    public ICollab addCollab(ICollab collab) {
        collabs.add(collab);
        return collab;
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> memberDetails = new HashMap<>();
        memberDetails.put("employeeInfo", this.employeeInfo.toString());
        Integer[] tmp = new Integer[managers.size()];
        for(int i = 0; i < managers.size(); i++) {
            tmp[i] = managers.get(i).getId();
```

```
    }

    //    DesignAdverts:
    public void addDesignAdvert(DesignAdvertisement advert) {
        designAdverts.put("Advert " + advert.getId(), advert.toMap());

        editor.setRepositoryStrings(designAdverts);
        editor.writeToTextFile(repo + "designAdvert.txt");
    }

    public ArrayList<DesignAdvertisement> getDesignAdverts() {
        ArrayList<DesignAdvertisement> tmp = new ArrayList<>();
        for(Map<String, String> advert: designAdverts.values()) {
            tmp.add(DesignAdvertisement.parse(advert));
        }
        return tmp;
    }

    public DesignAdvertisement getDesignAdvertById(int id) {
        String key = "Advert " + id;
        Map<String, String> advertDetails = designAdverts.get(key);
        return (advertDetails != null) ?
DesignAdvertisement.parse(advertDetails) : null;
    }


    //    TeamMembers:
    public void addTeamMember(TeamMember teamMember) {
        teamMembers.put("Team Member "+ teamMember.getId(),
teamMember.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addManager(Manager manager) {
        teamMembers.put(manager.getTeam().toString() + " Manager",
manager.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public void addHOD(HOD hod) {
        teamMembers.put("HOD", hod.toMap());

        editor.setRepositoryStrings(teamMembers);
        editor.writeToTextFile(repo + "department.txt");
    }

    public ArrayList<Manager> getManagers() {
        ArrayList<Manager> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if(member.containsKey("teamMembers")) {
                tmp.add(Manager.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers(Team team) {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (member.containsValue(team.toString()) &&
!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        return tmp;
    }

    public ArrayList<TeamMember> getTeamMembers() {
        ArrayList<TeamMember> tmp = new ArrayList<>();
        for(Map<String, String> member: teamMembers.values()) {
            if (!member.containsKey("teamMembers") &&
!member.containsKey("managers")) {
                tmp.add(TeamMember.parse(member));
            }
        }
        memberDetails.put("managers", Arrays.toString(tmp));
        return memberDetails;
    }

    @Override
    public void addManager(Manager manager) {
        managers.add(manager);
        MarketingDepartment.fileManager.addManager(manager);
    }

    @Override
    public Manager getManager(Team team) {
        for (Manager manager: managers) {
            if (manager.getTeam() == team) {
                return manager;
            }
        }
        return null;
    }

    @Override
    public String toString() {
        StringBuilder str = new StringBuilder("\nHOD: ");
        str.append("\nEmployeeInfo:
").append(this.employeeInfo.toString());
        str.append("\nManagers: ");
        for (Manager manager : managers) {
            str.append("\n  ").append(manager.toString());
        }
        return str.toString();
    }

    @Override
    public Event requestPhotoshoot(int modelId) {
        return
App.modelingDepartment.requestPhotoshoot(Integer.toString(modelId),
"");
    }

    public static HOD parse(Map<String, String> hod) {
        return new HOD(
                Employee.parseEmployee(hod.get("employeeInfo")),
                MarketingDepartment.fileManager.getManagers()
        );
    }
}
```

```
package src.Marketing;

import src.App;

import src.Marketing.src.*;
import src.Modeling.src.Event;
import src.Modeling.src.Item;
import src.Modeling.src.RecurrenceType;
import src.Security.src.SecurityRequestScheduler;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class MarketingDepartment {
    public static HOD hod;
    public static FileManager fileManager;
    Scanner s = new Scanner(System.in);

    public MarketingDepartment() {
        fileManager = new FileManager();
        hod = fileManager.getHOD();
    }

    public void start() throws Exception {
        int login = 0;
        while (login != 3) {
```

```
        }
        return tmp;
    }

    public HOD getHOD() {
        return HOD.parse(teamMembers.get("HOD"));
    }
}
```

```
package src.Marketing.src;

import src.Marketing.MarketingDepartment;
import src.Marketing.src.interfaces.IAdvertisement;
import src.Marketing.src.interfaces.ICollab;
import src.Marketing.src.interfaces.ICollabMember;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class Collab implements ICollab {
    static int nextid = 0;
    int id;

    ICollabMember member;
    ArrayList<IAdvertisement> advertisements;



    public Collab(int i, ICollabMember member, ArrayList<Integer>
advertisements) {
        this.id = i;
        if (id >= nextid) nextid = id + 1;
        this.member = member;
        this.advertisements = new ArrayList<>();
        for(Integer id: advertisements) {
//            TODO: parse method
        } = advertisements;
    }

    public Collab(ICollabMember member, ArrayList<IAdvertisement>
advertisements) {
        id = nextid;
        nextid++;
        this.member = member;
        this.advertisements = advertisements;

        MarketingDepartment.fileManager.addCollab(this);
    }

    public Collab(ICollabMember member) {
        id = nextid;
        nextid++;
        this.member = member;
        this.advertisements = new ArrayList<>();

        MarketingDepartment.fileManager.addCollab(this);
    }

    @Override
    public void addAdvertisement(IAdvertisement advertisement) {
        this.advertisements.add(advertisement);
    }

    @Override
    public void removeAdvertisement(IAdvertisement advertisement) {
        this.advertisements.remove(advertisement);
    }

    @Override
    public ArrayList<Integer> getAdvertisementIds() {
        ArrayList<Integer> tmp = new ArrayList<>();
        for(IAdvertisement advert: advertisements) {
            tmp.add(advert.getId());
        }
        return tmp;
    }
```

```
        System.out.println("""

                Which User are you Logging in as?
                 1: Head of Department
                 2: Admin
                 3: Exit Modeling Department
                """);
        login = s.nextInt();
        s.nextLine();
        switch (login) {
            case 1 -> initiateHOD();
            case 2 -> initiateAdmin();
        }
    }
    App.prompt();
}

private void initiateHOD() {
    int c = 0;

    while (c != 6) {
        System.out.println("""

                Please choose an action you want to take:
                 1: advertiseEven
                 2: advertiseDesign
                 3: getAllEventAdvertisements
                 4: getAllDesignAdvertisements
                 5: manageCollaborations
                 6: Back
                 111: Create Security Schedule
                 112: Show All Security Requests
                 113: Delete Security Request By ID
                """);
        c = s.nextInt();
        switch (c) {
            case 1 -> {
                ArrayList<Integer> eventIds = new ArrayList<>();
                for(Event event: App.modelingDepartment.getEvents()) {
                    eventIds.add(event.getId());
                }
                System.out.println("Which Event: " + eventIds);
                int event = s.nextInt();

                AdvertType type = AdvertType.COMMERCIAL;
                System.out.println("""
                        What type of Advertisement?
                         1: Billboard
                         2: Magazine
                         3: Social Media
                         4: Commercial""");
                int x = s.nextInt();
                switch (x) {
                    case 1 -> type = AdvertType.BILLBOARD;
                    case 2 -> type = AdvertType.MAGAZINE;
                    case 3 -> type = AdvertType.SOCIALMEDIA;
                }
                EventAdvertisement e =
hod.createEventAdvert(App.modelingDepartment.getEvent(event),
type);
                System.out.println(e.toString());
                fileManager.addEventAdvert(e);
            }
            case 2 -> {
                AdvertType type = AdvertType.COMMERCIAL;
                System.out.println("""
                        What type of Advertisement?
                         1: Billboard
                         2: Magazine
                         3: Social Media
                         4: Commercial""");
                int x = s.nextInt();
                switch (x) {
                    case 1 -> type = AdvertType.BILLBOARD;
                    case 2 -> type = AdvertType.MAGAZINE;
                    case 3 -> type = AdvertType.SOCIALMEDIA;
                }
                s.nextLine();
                System.out.println("Any Special Notes? You will be
```

```
    @Override
    public int getId() {
        return id;
    }

    @Override
    public ICollabMember getMember() {
        return member;
    }

    @Override
    public String toString() {
        return "\nCollab " + id + ":" + "\n Member: " + member.toString() +
                "\n Advertisements: " + getAdvertisementIds();
    }

    @Override
    public Map<String, String> toMap() {
        Map<String, String> brandDetails = new HashMap<>();
        brandDetails.put("id", Integer.toString(this.id));
        brandDetails.put("member", Integer.toString(this.member.getId()));
        brandDetails.put("advertisements",
this.getAdvertisementIds().toString());
        return brandDetails;
    }

    public static Collab parse(Map<String, String> brand) {
        String[] elements = brand.get("advertisements").replaceAll("[\\[\\] ]",
"").split(",");
        ArrayList<Integer> list = new ArrayList<>();
        for (String element : elements) {
            list.add(Integer.parseInt(element));
        }

        return new Collab(
                Integer.parseInt(brand.get("id")),
                brand.get("name"),
                list
        );
    }

}
```

```
advertising a ball gown");
                String str = s.nextLine();
                DesignAdvertisement e = hod.createDesignAdvert(type,
str);
                System.out.println(e.toString());
                fileManager.addDesignAdvert(e);
            }
            case 3 -> {
                for(EventAdvertisement ad: hod.getEventAdverts()) {
                    System.out.println(ad.toString());
                }
                System.out.println();
            }
            case 4 -> {
                for(DesignAdvertisement ad: hod.getDesignAdverts()) {
                    System.out.println(ad.toString());
                }
                System.out.println();
            }
            case 5 -> initiateManagement();

            case 111 -> {
                SecurityRequestScheduler scheduler = new
SecurityRequestScheduler();
                scheduler.addSecurityRequest();
            }
            case 112 -> {
                SecurityRequestScheduler scheduler1 = new
SecurityRequestScheduler();
                scheduler1.showAllSecurityRequests();
            }
            case 113 -> {
                SecurityRequestScheduler scheduler2 = new
SecurityRequestScheduler();
                scheduler2.deleteScheduleByID();
            }
        }
    }
}

    private void initiateAdmin() {
        Scanner s = new Scanner(System.in);
        int c = 0;

        while (c != 2) {
            System.out.println("""
                    Please choose an action you want to take:\s
                     1: manage HOD
                     2: Back""");
            c = s.nextInt();
            if (c == 1) {
                int response = 0;
                while (response != 2) {
                    System.out.println("""
                            Managing HOD:\s
                             1: add
                             2: Back""");
                    response = s.nextInt();
                    if (response == 1) {
                        src.Modeling.src.HOD hod = new
src.Modeling.src.HOD();
                        System.out.println(hod);
                    }
                }
            }
        }
    }

    private void initiateManagement() {
        int userChoice = 0;
        while (userChoice != 7) {
            System.out.println("""

                    Please choose an action you want to take:
                     1: requestCollab
                     2: updateCollab
                     3: deleteCollab
                     4: printApprovedMembers
                     5: printCollabs
```

```
                7: Back
                """);
        userChoice = s.nextInt();
        s.nextLine();
        switch (userChoice) {
            case 1 -> {
            }
            case 2 -> {}
            case 3 -> {}
            case 4 -> {}
            case 5 -> {
            }
            case 6 -> {
            }
        }
    }
  }
}
```

HR Department:

```
package src.HR.src;

import src.HR.src.interfaces.ICandidate;

/**
 * @author Sam Gumm
 */

public class Candidate implements ICandidate {
        String candidateId;
        String name;
        String positionApplied;
        candidateStatus status;

        /**
        * @param candidateId the Candidate's identifying
number
        * @param name the Candidates name
        * @param positionApplied the position the
Candidate has applied to
        * @param status  the Status of the Candidate in
the Hiring process
        */
        public Candidate(String candidateId, String name,
String positionApplied, candidateStatus status) {
        this.candidateId = candidateId;
        this.name = name;
        this.positionApplied = positionApplied;
        this.status = status;
        }

        /**
        *
        * @return name
        */
        @Override
        public String getName() {
        return name;
        }

        /**
        *
        * @return candidateId
        */
        @Override
        public String getCandidateId() {
        return candidateId;
        }

        /**
        *
        * @return positionApplied
        */
        @Override
        public String getPositionApplied() {
        return positionApplied;
        }

        /**
        * @return status
        */
        @Override
        public candidateStatus getStatus() {
        return status;
        }

        /**
        *
        * @param status the Status can be 5 things:
        *       Applied --> they have submitted an
application, but not had an interview
        *       Pending --> they are waiting on an
interview
        *       Approved --> they have had an interview
and are approved for next stage
        *       Rejected --> they have been ultimately
rejected at any stage
        *       Hired --> they are set to be turned into
```

```
employees
		*/
		@Override
		public void setStatus(String status) {

		if(status == null) {
		throw new NullPointerException("Status cannot be
null");
		}

		//TODO: use enumeration here (maybe?)
		else if (status.equalsIgnoreCase("APPLIED") ||
status.equalsIgnoreCase("Pending") ||
status.equalsIgnoreCase("Approved") ||
status.equalsIgnoreCase("Rejected") ||
status.equalsIgnoreCase("Hired")) {
		this.status = candidateStatus.valueOf(status);
		}
		else {
		throw new IllegalArgumentException("Status must
be: Applied, Pending, Approved, Rejected, Hired");
		}
		}

		/**
		* @return String
		*/
		@Override
		public String toString() {
		return String.format("ID: %s, Name: %s, Position
Applied: %s, Status: %s",
		candidateId, name, positionApplied, status);
		}
}package src.HR.src;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.*;

/**
 * Manages Candidate records, allowing for adding,
removing, updating, moving and displaying Candidate.
 * @author Sam Gumm
 */
public class candidateRecordManager {
		Map<String, Map<String, String>> data = new
LinkedHashMap<>();
		private final fileStorageHR storageHR;

		public candidateRecordManager(fileStorageHR
storageHR) {
		this.storageHR = storageHR;
		}



		/**
		* Add a new candidate into a txt file in format
"candID".txt and stores it to candidateStorage
		* @param candidate the Candidate object to be
stored to file
		*/
		public void addCandidate(Candidate candidate)
throws IOException {
		Map<String, String> candidateObject = new
LinkedHashMap<>();
		candidateObject.put("candidateID",
candidate.getCandidateId());
		candidateObject.put("name",
candidate.getName());
		candidateObject.put("positionApplied",
candidate.getPositionApplied());
		candidateObject.put("status",
candidate.getStatus().toString());
		data.put(candidate.getCandidateId(),
candidateObject);
		storageHR.poorJarser.setRepositoryStrings(data);
		//TODO: Print out all potential statuses here
		String filepathToCandidateStorage =
String.valueOf(storageHR.getCandidateStoragePath(String.v
alueOf(candidate.getStatus())));

storageHR.poorJarser.writeToTextFile(filepathToCandidateSt
orage + "/" + candidate.getCandidateId() + ".txt");
		data.clear();
		}

		/**
		* Remove a candidate by ID
		* @param candidateID ID of the Candidate file to
remove
		*
		*/
		public void removeCandidate(String candidateID)
throws Exception {
		if(data.containsKey(candidateID)) {
		try {
		data.remove(candidateID);
		storageHR.poorJarser.setRepositoryStrings(data);
		} catch (Exception e) {
		throw new Exception("Error in removeCandidate
when removing found ID from LinkedHashMap<>(): \n" +
e.getMessage());
		}
		}

		String filePath =
String.valueOf(findCandidateFile(candidateID));
		if(filePath == null) {
		throw new NullPointerException("filepath is null");
		}

		try {
		storageHR.deleteFile(filePath);
		} catch (Exception e) {
		throw new Exception("Error in removeCandidate
when deleting file from repository with file path: "
		+ filePath + "\n" + e.getMessage());
		}
```

```
	}

	public Path findCandidateFile(String candidateID)
throws IOException {
	Path base =
storageHR.getDefault_filepath_candidateStorage();
	String candidateFileName = candidateID + ".txt"; //
The candidate file name format

	try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(base)) {
	for (Path statusDir : directoryStream) {
	if (Files.isDirectory(statusDir)) { // Ensure it's a
directory
	Path candidateFile =
statusDir.resolve(candidateFileName);
	if (Files.exists(candidateFile)) {
		return candidateFile;
	}
	}
	}
	}
	return null; // Return null if the candidate file is not
found
	}



	private void moveCandidate(String candidateID,
candidateStatus status) throws IOException {
	// Find the current candidate file path
	Path currentCandidateFile =
findCandidateFile(candidateID);
	if (currentCandidateFile == null ||
!Files.exists(currentCandidateFile)) {
	throw new FileNotFoundException("Candidate file
not found: " + candidateID + ".txt");
	}

	// Get the path to the new status folder
	Path newStatusFolder =
storageHR.getCandidateStoragePath(status.toString());
	if (!Files.exists(newStatusFolder)) {
	Files.createDirectories(newStatusFolder);
	}

	// Construct the destination path for the candidate
file
	Path destinationFile =
newStatusFolder.resolve(candidateID + ".txt");

	// Move the candidate file to the new status folder
	Files.move(currentCandidateFile, destinationFile,
StandardCopyOption.REPLACE_EXISTING);

	System.out.println("Candidate was successfully
moved to " + destinationFile);
	}

	/**
	* Takes the CandidateID of the Candidate object to
be changed, then checks candidateStorage
	* for the ID. If found, the filepath is returned, and
the candidateObject is parsed, altered, and
	* moved to the new Status folder if necessary.
	* @param candidateId ID of the candidate to be
updated
	*/
	public void updateCandidate(String candidateId)
throws Exception {
	String filepath;

	//As findCandidateFolder may return null, we need
to make sure that doesn't happen
	filepath =
Objects.requireNonNull(findCandidateFile(candidateId)).toStr
ing();
	//System.out.println("updateCandidate filepath
from finding: " + filepath);

	//read from Candidate file
	storageHR.poorJarser.processTextFile(filepath);

	//initialize data from current repo
	Map<String, Map<String, String>> data =
storageHR.poorJarser.getRepositoryStringMap();

	//make sure data is not null
	//TODO: is this necessary?
	if(data == null) {
	throw new Exception("data was not initialized
properly: \n");
	}

	//initialize candidateObject to modify
	Map<String, String> candidateObject =
data.get(candidateId);
	if(candidateObject == null) {
	candidateObject = new LinkedHashMap<>();
	}
	Scanner scanner = new Scanner(System.in);

	System.out.println("Enter candidate name: ");
	String name = scanner.next();
	System.out.println("Enter position applied: ");
	String positionApplied = scanner.next();
	System.out.println("Enter in new candidate hiring
status: \nApplied\nPending\nApproved\nRejected\nHiring\n
Enter Here: ");
	candidateStatus candidateStatus =
src.HR.src.candidateStatus.valueOf(scanner.next().toUpper
Case());
	candidateObject.put("candidateName", name);
	candidateObject.put("positionApplied",
positionApplied);
	candidateObject.put("candidateStatus",
String.valueOf(candidateStatus));
	data.put(candidateId, candidateObject);
	storageHR.poorJarser.setRepositoryStrings(data);
	moveCandidate(candidateId, candidateStatus);
	storageHR.poorJarser.writeToTextFile(filepath);
	storageHR.deleteFile(filepath);
	}
```

```
	public void displayCandidatesByStatus(String
canStatus) throws Exception {
	try {
	String statusDirName = canStatus.toUpperCase();

	Path statusDir =
storageHR.getCandidateStoragePath(statusDirName);

	if(statusDir == null) {
	throw new NullPointerException("statusDir is
null");
	}

	if(!Files.exists(statusDir) ||
!Files.isDirectory(statusDir)) {
	System.out.println("No candidates found with
status: " + canStatus);
	return; // Add return to exit the method if directory
doesn't exist
	}

	try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(statusDir, "*.txt")) {
	boolean candidatesFound = false;
	for (Path path : directoryStream) {
	candidatesFound = true;
	storageHR.loadFileAndPrint(path.toString()); //
Pass full path as string
	}
	if(!candidatesFound) {
	System.out.println("No candidates found with
status: " + canStatus);
	}
	}
	} catch (Exception e) {
	throw new Exception("displayCandidatesByStatus
failed: \n" + e.getMessage(), e);
	}
	}



}package src.HR.src;

public enum candidateStatus {
	APPLIED,
	APPROVED,
	HIRING,
	PENDING,
	REJECTED;

	@Override
	public String toString() {
	return switch (this) {
	case APPLIED -> "Applied";
	case APPROVED -> "Approved";
	case HIRING -> "Hiring";
	case PENDING -> "Pending";
	case REJECTED -> "Rejected";
	};
	}
}

package src.HR.src;

/**
 * @author Sam Gumm
 */
public enum Department {
	ENGINEERING,
	MARKETING,
	HUMAN_RESOURCES,
	FINANCE,
	DESIGN,
	MODELING,
	MANUFACTURING;

	@Override
	public String toString() {
	return switch (this) {
	case ENGINEERING -> "Engineering";
	case MARKETING -> "Marketing";
	case HUMAN_RESOURCES ->
"Human_Resources";
	case FINANCE -> "Finance";
	case DESIGN -> "Design";
	case MODELING -> "Modeling";
	case MANUFACTURING -> "Manufacturing";
	};
	}
}
package src.HR.src;

import src.HR.src.interfaces.IEmployee;

/**
 * @author Sam Gumm
 */
public class Employee implements IEmployee {
	String employeeId;
	String name;
	Department department;
	String position;
	String employmentStatus;
	int salary;

	/**
	*
	* @param employeeId String, the Employee's
identifying number
	* @param name String the Employee's name
	* @param department Department, the
Employee's assigned department
	* @param position String, the Employee's
assigned position
	* @param employmentStatus String, the
Employee's current employment status:
	*                                  Onboarding --> Just
hired from Candidate
	*                                  Active --> Finished
```

```
Onboarding tasks, full employee
	*                                   Offboarding --> Pending being deleted from system (Employee has been fired)
	* @param salary int, the Employee's recorded salary
	*/
	public Employee(String employeeId, String name, Department department, String position, String employmentStatus, int salary) {
	this.employeeId = employeeId;
	this.name = name;
	this.department = department;
	this.position = position;
	this.employmentStatus = employmentStatus;
	this.salary = salary;
	}

	/**
	*
	* @return name
	*/
	@Override
	public String getName() {
	return name;
	}

	/**
	*
	* @return employeeId
	*/
	@Override
	public String getEmployeeID() {
	return employeeId;
	}

	/**
	* @return department
	*/
	@Override
	public String getDepartment() {
	return String.valueOf(department);
	}

	/**
	*
	* @return position
	*/
	@Override
	public String getPosition() {
	return position;
	}

	/**
	*
	* @return employmentStatus
	*/
	@Override
	public String getEmployementStatus() {
	return employmentStatus;
	}

	/**
	* @return salary
	*/
	@Override
	public String getSalary() {
	return String.valueOf(salary);
	}

	/**
	* @return String
	*/
	@Override
	public String toString() {
	return String.format("ID: %s, Name: %s, Department: %s, Position: %s, Status: %s, Salary: %d",
	employeeId, name, department, position, employmentStatus, salary);
	}

	public static Employee parseEmployee(String employeeString) {
	// Remove unnecessary spaces and split the string by commas
	String[] parts = employeeString.split(", ");

	// Extract the individual fields using the known format
	String employeeId = parts[0].split(": ")[1];
	String name = parts[1].split(": ")[1];
	Department department = Department.valueOf(parts[2].split(": ")[1].toUpperCase()); // Assuming Department is an enum
	String position = parts[3].split(": ")[1];
	String employmentStatus = parts[4].split(": ")[1];
	int salary = Integer.parseInt(parts[5].split(": ")[1]);

	// Create and return the Employee object
	return new Employee(employeeId, name, department, position, employmentStatus, salary);
	}
}
package src.HR.src;


import java.io.*;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardCopyOption;
import java.util.LinkedHashMap;
import java.util.Map;
import java.util.Objects;
import java.util.Scanner;

/**
 * @author Sam Gumm
 */
public class employeeRecordManager {
	Map<String, Map<String, String>> data = new LinkedHashMap<>();
```

```
	private final fileStorageHR storageHR;

	/*
	TODO:
	- make methods to transfer employees to new
folders
	- implement this in updateEmployee
	*/

	public employeeRecordManager(fileStorageHR
storageHR) {
	this.storageHR = storageHR;
	}

	/**
	* Adds an Employee to record, then persistently
stores it in "empID".txt
	*
	* @param employee the Employee object to be
added
	*/
	// Add a new employee
	public void addEmployee(Employee employee)
throws IOException {
	//TODO: have counter that increments for
employeeID
	Map<String, String> employeeObject = new
LinkedHashMap<>();
	employeeObject.put("name",
employee.getName());
	employeeObject.put("employeeID",
employee.getEmployeeID());
	employeeObject.put("employeeDepartment",
employee.getDepartment());
	employeeObject.put("employeePosition",
employee.getPosition());
	employeeObject.put("employeeStatus",
employee.getEmployementStatus());
	employeeObject.put("employeeSalary",
employee.getSalary());
	data.put(employee.getEmployeeID(),
employeeObject);
	storageHR.poorJarser.setRepositoryStrings(data);
	String filepathToEmployeeStorage =
String.valueOf(storageHR.getEmployeeStoragePath(String.v
alueOf(employee.getDepartment())));

storageHR.poorJarser.writeToTextFile(filepathToEmployeeSt
orage + "/" + employee.getEmployeeID() + ".txt");
	}

	/**
	* Removes employee FILE from employeeStorage
repo, uses fileStorageHR filepath to find file.
	* Also attempts to remove Employee from
LinkedHashMap record if it exists there as well.
	*
	* @param employeeID the EmployeeID to be
removed
	*
	*/
	public void removeEmployee(String employeeID)
throws Exception {
	//remove employee from data LinkedHashMap if it
contains the id
	if(data.containsKey(employeeID)) {
	try {
	data.remove(employeeID);
	storageHR.poorJarser.setRepositoryStrings(data);
	} catch (Exception e) {
	throw new Exception("Error in removeEmployee
when removing found ID from LinkedHashMap<>(): \n" +
e.getMessage());
	}
	}

	String filepath =
String.valueOf(findEmployeeFile(employeeID));
	if(filepath == null) {
	throw new NullPointerException("filepath from
removeEmployee is null");
	}

	try {
	storageHR.deleteFile(filepath);

	} catch (Exception e) {
	throw new Exception("Error in removeEmployee
when deleting file from repository with file path: "
	+ filepath + "\n" + e.getMessage());
	}
	}


	private void moveEmployee(String employeeID,
Department department) throws IOException {
	// Find the current employee file path
	Path currentEmployeeFile =
findEmployeeFile(employeeID);
	if (currentEmployeeFile == null ||
!Files.exists(currentEmployeeFile)) {
	throw new FileNotFoundException("Employee file
not found: " + employeeID + ".txt");
	}

	// Get the path to the new Department folder
	Path newDepartmentFolder =
storageHR.getEmployeeStoragePath(department.toString().t
oUpperCase());
	if (!Files.exists(newDepartmentFolder)) {
	throw new FileNotFoundException("Department
folder not found: " + department.toString().toUpperCase());
	}

	// Construct the destination path for the candidate
file
	Path destinationFile =
newDepartmentFolder.resolve(employeeID + ".txt");

	// Move the candidate file to the new status folder
	Files.move(currentEmployeeFile, destinationFile,
StandardCopyOption.REPLACE_EXISTING);
```

```
        System.out.println("Employee was successfully
moved to " + destinationFile);
        }
        /**
        * Takes the EmployeeID of the Employee object to
be changed, then checks the LinkedHashMap
        * for the ID; if it finds the ID, the associated
Employee is altered and re-uploaded to file,
        * otherwise, an error is thrown.
        *
        * @param employeeID the associated ID of the
Employee
        */
        public void updateEmployee(String employeeID)
throws Exception {
        String filepath;

        filepath =
Objects.requireNonNull(findEmployeeFile(employeeID)).toSt
ring();

        //read from Employee file
        storageHR.poorJarser.processTextFile(filepath);

        //initialize data from current repo
        Map<String, Map<String, String>> data =
storageHR.poorJarser.getRepositoryStringMap();

        //make sure data is not null
        //TODO: is this necessary?
        if(data == null) {
        throw new Exception("data was not initialized
properly: \n");
        }

        //initialize candidateObject to modify
        Map<String, String> employeeObject =
data.get(employeeID);
        if(employeeObject == null) {
        employeeObject = new LinkedHashMap<>();
        }
        removeEmployee(employeeID);
        Scanner scanner = new Scanner(System.in);

        for(int i = 0; i < Department.values().length; i++) {
        System.out.println(Department.values()[i].name());
        }
        System.out.println("Enter new employee
Department: ");
        Department department =
src.HR.src.Department.valueOf(scanner.next().toUpperCase
());

        System.out.println("Enter in new employee
position: ");
        String position = scanner.next();

        System.out.println("Enter in new employee Status:
");
        String status = scanner.next();

        System.out.println("Enter in new employee salary:
");
        String salary = scanner.next();

        employeeObject.put("employeeID", employeeID);
        employeeObject.put("name",
employeeObject.get("name"));
        employeeObject.put("department",
department.toString());
        employeeObject.put("position", position);
        employeeObject.put("status", status);
        employeeObject.put("salary", salary);

        data.put(employeeID, employeeObject);


        storageHR.poorJarser.setRepositoryStrings(data);
        storageHR.poorJarser.writeToTextFile(filepath);
        moveEmployee(employeeID, department);
        }

        /**
        * Display all records currently in the
LinkedHashMap "data"
        */
        public void displayCurrentEmployeeRecords() {
        data.values().forEach(System.out::println);
        }

        //TODO: move this to fileStorageHR and replace
with method call
        public void displayFileRecords(String folderPath)
throws Exception {
        File folder = new File(folderPath);
        if(folder.isDirectory()) {
        File[] files = folder.listFiles();
        int i = 0;
        do {
        assert files != null;
        File file = files[i];
        if(file.isFile() && file.getName().endsWith(".txt")) {
        try (BufferedReader br = new BufferedReader(new
FileReader(file))) {
                String line;
                while ((line = br.readLine()) != null) {
                System.out.println(line);
                }
        } catch (IOException e) {
                throw new RuntimeException(e);
        }
        }
        i++;
        } while (i < files.length);
        }
        else {
        throw new Exception("File is not a directory");
        }
        }
```

```
	/**
	* @param departmentName the Department to
iterate through.
	*/
	public void
displayEmployeesByDepartment(Department
departmentName) throws Exception {
	try {
	String statusDirName =
String.valueOf(departmentName).toUpperCase();

	Path statusDir =
storageHR.getEmployeeStoragePath(statusDirName);

	if(statusDir == null) {
	throw new NullPointerException("statusDir is
null");
	}

	if(!Files.exists(statusDir) ||
!Files.isDirectory(statusDir)) {
	System.out.println("No candidates found with
status: " + departmentName);
	return;
	}

	try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(statusDir, "*.txt")) {
	boolean employeesFound = false;
	for (Path path : directoryStream) {
	employeesFound = true;
	storageHR.loadFileAndPrint(path.toString()); //
Pass full path as string
	}
	if(!employeesFound) {
	System.out.println("No candidates found with
status: " + departmentName);
	}
	}
	} catch (Exception e) {
	throw new
Exception("displayEmployeesByDepartment failed: \n" +
e.getMessage(), e);
	}
	}

	public Path findEmployeeFile(String employeeID)
throws IOException {
	Path base =
storageHR.getDefault_filepath_employeeStorage();
	String employeeFileName = employeeID + ".txt";

	try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(base)) {
	for (Path statusDir : directoryStream) {
	if (Files.isDirectory(statusDir)) { // Ensure it's a
directory
	Path employeeFile =
statusDir.resolve(employeeFileName);
	if (Files.exists(employeeFile)) {
		return employeeFile;
	}
	}
	}
	}
	return null; // Return null if the candidate file is not
found
	}
}
package src.HR.src;
import src.TextEditor.*;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Scanner;

public class fileStorageHR {

	private final Path
default_filepath_employeeStorage = Paths.get("src", "HR",
"repository", "employeeStorage");
	private final Path
default_filepath_candidateStorage = Paths.get("src", "HR",
"repository", "candidateStorage");
	String filepath = "";
	public PoorTextEditor poorJarser = new
PoorTextEditor(); //haha haha

	public fileStorageHR() {}

	public String getFilepath() {
	return filepath;
	}

	public Path
getDefault_filepath_employeeStorage() {
	return default_filepath_employeeStorage;
	}

	public Path
getDefault_filepath_candidateStorage() { return
default_filepath_candidateStorage; }

	public void setFilepath(String filepath) {
	this.filepath = filepath;
	}

	/**
	* Deletes a given file, expects to be given the
filepath however
	* @param filename name of File to be deleted,
must contain file extension
	*/
	public void deleteFile(String filename) throws
Exception {
	try {
	Path filePath = Paths.get(getFilepath(), filename);
```

```
            File file = filePath.toFile();
            System.out.println("Attempting to delete: " +
file.getAbsolutePath());

            if (file.exists()) {
            if (file.delete()) {
            System.out.println(filename + " deleted
successfully");
            } else {
            throw new Exception("Failed to delete " +
filename);
            }
            } else {
            throw new FileNotFoundException("File not found:
" + file.getAbsolutePath());
            }
            } catch (Exception e) {
            throw new Exception(e.getMessage());
            }
            }



            /**
            * @param folderName the name of the
Department folder to be accessed
            * @return returns a String representation of the
folderPath to the Department Folder
            * @throws IOException error checking
            */
            public Path getEmployeeStoragePath(String
folderName) throws IOException {
            Path folderPath = Paths.get("src", "HR",
"repository", "employeeStorage",
folderName.toUpperCase());
            if (!Files.exists(folderPath)) {
            throw new FileNotFoundException("Folder not
found: " + folderName.toUpperCase());
            }

            return folderPath.toAbsolutePath();
            }

            /**
            * Works by using the working directory saved at
runtime to create the folderPath
            *  it then tries to find the absolute path of the
folder, in order to work across
            *  different machines.
            * @param folderName the specific Status folder to
set the path to
            * @return returns a Path object that contains the
absolute path of the defined folder
            * @throws IOException error checking
            */
            public Path getCandidateStoragePath(String
folderName) throws IOException {
            Path folderPath = Paths.get("src", "HR",
"repository", "candidateStorage",
folderName.toUpperCase());
            if (!Files.exists(folderPath)) {
            throw new FileNotFoundException("Folder not
found: " + folderName.toUpperCase());
            }

            return folderPath.toAbsolutePath();
            }

            /**
            *
            * @param filePath the path to the file to be printed
out to console
            * @throws Exception error checking
            */
            public void loadFileAndPrint(String filePath) throws
Exception {
            try {
            System.out.println("Attempting to load: " +
filePath);
            File file = new File(filePath);
            System.out.println("Loading file: " +
file.getAbsolutePath());

            // Check if the file exists
            if (!file.exists()) {
            throw new FileNotFoundException("File not found:
" + file.getAbsolutePath());
            }

            try (Scanner scanner = new Scanner(file)) {
            while(scanner.hasNextLine()) {
            String line = scanner.nextLine();
            System.out.println(line);
            }
            }
            } catch (Exception e) {
            throw new Exception("Error encountered in
loadFileAndPrint with file: "
            + filePath + "\n" + e.getMessage(), e);
            }
            }

}
package src.HR.src;

import java.io.*;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class hiringProcess {
            /*
            TODO:
            - set up the hiring process
            - set up the folder access
            - set up the folder storage system
            - have txt files that will be stored into the
corresponding day of the week
```

```
        - the txt files will store what current candidates are going to be interviewed that day
        - ...
        Thought process:
        Need to have counter for interview ID
        -> have this stored in a txt file somewhere, everytime the counter increments, the txt counter is
        also incremented;
        Have interviewer folders, and ability to create and remove the folders
        each folder has days of the week as folders inside of them
        each day of the week has txt files that represent time slots, when a interview is scheduled,
        it gets saved as, for example a 12:15 slot, (maybe have it be appended to the InterviewID 000_1215

        1215_1.txt
        //TODO: INITIALIZE THE CARDS EXACTLY LIKE THIS
        //
        Time: 1215
        Candidate: Someone else
        Interviewer: Someone
        Notes: blank
                                //

        the operator can choose to see all timeslots currently created in a print out fashion, with days as headers
        they can also type in the timeslot (maybe have this changed to candidate name or something when sent to complete)
        there is a completed interviews folder where the user opens up a timeslot, and then chooses to send it to complete
        this opens up the option for them to add in any notes that they may have, and then saves the candidate in
        this format in the completed folder
        someone.txt
        //
        Candidate Name: Something
        Interviewer: Something else
        Notes: //something here idk
        Recommendation: Be sent to such and such department as this position idk

//
        the user should be able to get full or individual print outs of all folders available, including their contents

        */

        /*
        STEPS:
        - create interview from info
        - assign to interviewer
        - set timeslot
        - repeat until finished
        > interview is had
        > interviewer records notes about interview
        - insert notes of interviewer into the interview txt file
        - move the txt file to Completed
        */

        /*
        Necessities:
        - See all interviews arranged by interviewer
        - See only one interviewer's timeslots
        - See one timeslot's internals
        - Global counter for interviewID
        */

        private final Path folderPathSchedules = Paths.get("src", "HR", "repository", "scheduleStorage");
        private final Path folderPathIDs = Paths.get("src", "HR", "repository", "scheduleStorage", "idStorage");

        //TODO: Works
        public void createInterview() throws IOException {
        /*
        TODO:
        - alter this to create a folder for an interviewer
        and upon assigning a card to them, either the folder exists and it is deposited there
        or it creates a new folder after prompting the user to confirm that the chosen interviewer is correct
        */
        String data = "";
        Scanner input = new Scanner(System.in);
        System.out.println("Please enter Interview Time (HH:MM)");
        String interviewTime = input.nextLine();
        System.out.println("Please ensure this is the correct time y/n: " + interviewTime);
        String answer = input.nextLine();
        if (answer.equals("n") || answer.equals("N")) {
        //Currently needs to be this way otherwise pulling data will not work correctly (NO SEMICOLONS)
        System.out.println("Please enter Interview Time (HHMM)");
        interviewTime = input.nextLine();
        }
        data += "Interview Time: " + interviewTime + "\n";


        //candidate name
        System.out.println("Please enter Candidate Name (First Last):");
        String candidateName = input.nextLine().strip();
        System.out.println("Please ensure this is the correct Candidate y/n: " + candidateName);
        answer = input.nextLine();
        if (answer.equals("n") || answer.equals("N")) {
        System.out.println("Please enter Candidate Name (First Last):");
        candidateName = input.nextLine();
        }
        data += "Candidate Name: " + candidateName + "\n";
```

```
            //assign interviewer
            System.out.println("Please choose Interviewer to
assign:");
            //File folder = folderPathSchedules.toFile();
            String interviewer = input.nextLine();
            System.out.println("Please ensure this is the
correct Interviewer y/n: " + interviewer);
            answer = input.nextLine();
            if (answer.equals("n") || answer.equals("N")) {
            System.out.println("Please choose Interviewer to
assign:");
            interviewer = input.nextLine();
            }
            data += "Interviewer: " + interviewer + "\n";

            //add notes
            System.out.println("PLease enter notes: ");
            String notes = input.nextLine();
            System.out.println("Please ensure this is correct
(y/n): " + notes);
            answer = input.nextLine();
            if (answer.equals("n") || answer.equals("N")) {
            System.out.println("Please choose Interviewer to
assign:");
            notes = input.nextLine();
            }
            data += "Notes: " + notes + "\n";

            System.out.println("generating ID...");

            //checking to make sure folder is there
            int idCounter = 0;
//          System.out.println("idCounter: " + idCounter);
            Path filePath;
            try {
            filePath = Paths.get(folderPathIDs.toString(),
"idFile.txt");
            } catch (Exception e) {
            throw new FileNotFoundException("File not
found");
            }
//          System.out.println(filePath);

            //read integer then increment and write to file
            try(BufferedReader br = new BufferedReader(new
FileReader(filePath.toFile()))) {
            String line = br.readLine();
//          System.out.println(line);
            idCounter = Integer.parseInt(line);
//          System.out.println("idCounter" + idCounter);
            CharSequence chars = ++idCounter + "";
//          System.out.println(chars);
            Files.writeString(filePath, chars);
            }

            //Save file to scheduleStorage TODO: clean up
            File interview = new
File(folderPathSchedules.toFile(), interviewTime + "_" +
idCounter + ".txt");
            Files.writeString(interview.toPath(), data);
            if (interview.createNewFile()) {
            System.out.println("File Created");
            }
//          String content = new
String(Files.readAllBytes(interview.toPath()));
//          System.out.println("Total Space: " +
content.length() + " Bytes");
//          System.out.println("folderPathSchedules: " +
folderPathSchedules);
//          System.out.println("Interview Time: " +
interviewTime);
//          System.out.println(interview.exists());
//          System.out.println(data);
            }

            public void editInterview() throws IOException {
            /* TODO
            - trawl through scheduleStorage until file is found
            - take user input to replace:
            1. Interview Time
            2. Candidate Name
            3. Interviewer on File
            4. Interview Notes
            - STEPS:
            - read file and store to String
            - iterate through String and change marked section
            - replace file with new string
            */
            //Take user input
            Scanner scanner = new Scanner(System.in);
            System.out.println("Please enter interview ID to
edit:");
            String interviewID = scanner.nextLine();
            System.out.println("Please ensure this is the
correct time y/n: " + interviewID);
            String answer = scanner.nextLine();
            //TODO: replace with while loop
            if (answer.equals("n") || answer.equals("N")) {
            System.out.println("Please enter Candidate Name
(First Last):");
            interviewID = scanner.nextLine();
            }



            //search for file and read to string
            //TODO: extract into a method that returns data()[]
            File folder = folderPathIDs.toFile();
            if (!folder.exists()) {
            System.out.println("Folder does not exist");
            }
            Path filePath = null;
            List<String[]> data = new ArrayList<>();
            String[] payload = null;
            try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(folderPathSchedules)) {
            for (Path path : directoryStream) {
            //TODO prolly need to rework to iterate through the
file name until after the _ for ID
            if
(path.getFileName().toString().contains(interviewID)) {
            filePath = path;
```

```
            System.out.println("Found file: " +
path.getFileName().toString());
//                      System.out.println("interviewPath: " +
path);
            //System.out.println(Files.exists(path) ? "Yes" :
"No");
            Scanner fileScanner = new Scanner(path);
            while (fileScanner.hasNextLine()) {
                        //TODO replace with HashMap prolly
                        assert false;
                        payload = fileScanner.nextLine().split(":
");

System.out.println(Arrays.toString(payload));
                        data.add(payload);
            }
            break;
            }
            }
            }

            //TODO: this will eventually call the above method
and use the data()[] it returns
            while(true) {
            System.out.println("Please choose operation:
[E]dit, [Q]uit:");
            String operation = scanner.nextLine();

            if (operation.equals("E")) {
            System.out.println("Choose what information to
edit: \n[I]nterviewer Assigned\n[C]andidate Name\n[N]otes");
            String userInput = scanner.nextLine();
            switch (userInput) {
            case "I" -> {
                        System.out.println("Please enter new
Interviewer to assign: ");
                        String newInterviewer =
scanner.nextLine();
                        System.out.println("Is " + newInterviewer
+ " correct? (y/n): " );
                        answer = scanner.nextLine();
                        if(answer.equals("n") ||
newInterviewer.equals("N")) {
                        System.out.println("Please enter new
Interviewer to assign: ");
                        newInterviewer = scanner.nextLine();
                        }
                        for (String[] datum : data) {
                        for (int j = 0; j < datum.length; j++) {
                        if (datum[j].equals("Interviewer")) {
//                                  System.out.println("Found
Interviewer section at: " + j);
                        int temp = j;
                        datum[++temp] = newInterviewer;
                        System.out.println(datum[j + 1]);
                        break;
                        }
                        }
                        }
                        assert filePath != null;
                        System.out.println("Started the writing
process");
                        StringBuilder newPayload = new
StringBuilder();
                        for (String[] datum : data) {
                        for (String s : datum) {
                        newPayload.append(s).append(" ");
                        System.out.println(s);
                        }
                        newPayload.append("\n");
                        }
                        System.out.println("Finished writing
process: " + newPayload);
                        Files.writeString(filePath,
newPayload.toString());
            }

            case "C" -> {
                        System.out.println("Please enter new
Candidate name: ");
                        String newCandidate =
scanner.nextLine();
                        System.out.println("Is " + newCandidate
+ " correct? (y/n): " );
                        answer = scanner.nextLine();
                        if(answer.equals("n") ||
newCandidate.equals("N")) {
                        System.out.println("Please enter new
Candidate name: ");
                        newCandidate = scanner.nextLine();
                        }
                        for (String[] datum : data) {
                        for (int j = 0; j < datum.length; j++) {
                        if (datum[j].equals("Candidate")) {
//                                  System.out.println("Found
Candidate section at: " + j);
                        int temp = j;
                        datum[++temp] = newCandidate;
                        System.out.println(datum[j + 1]);
                        break;
                        }
                        else {
                        System.out.println("Was not able to find
Candidate section...");
                        }
                        }
                        }
                        assert filePath != null;
                        System.out.println("Started the writing
process");
                        String newPayload = "";
                        for (String[] datum : data) {
                        for (int j = 0; j < datum.length; j++) {
                        newPayload += datum[j] + " ";
                        System.out.println(datum[j].toString());
                        }
                        newPayload += "\n";
                        }
                        System.out.println("Finished writing
process: " + newPayload);
                        Files.writeString(filePath, newPayload);
            }
```

```
            case "N" -> {
                        System.out.println("Please enter new
notes: ");
                        String newNotes = scanner.nextLine();
                        System.out.println("Is '" + newNotes + "'
correct? (y/n): " );
                        answer = scanner.nextLine();
                        if(answer.equals("n") ||
newNotes.equals("N")) {
                        System.out.println("Please enter new
notes: ");
                        newNotes = scanner.nextLine();
                        }
                        for (String[] datum : data) {
                        for (int j = 0; j < datum.length; j++) {
                        if (datum[j].equals("Notes")) {
//                                  System.out.println("Found
notes section at: " + j);
                        int temp = j;
                        datum[++temp] = newNotes;
                        System.out.println(datum[j + 1]);
                        break;
                        }
                        }
                        }
                        assert filePath != null;
                        System.out.println("Started the writing
process");
                        String newPayload = "";
                        for (String[] datum : data) {
                        for (int j = 0; j < datum.length; j++) {
                        newPayload += datum[j] + " ";
                        System.out.println(datum[j].toString());
                        }
                        newPayload += "\n";
                        }
                        System.out.println("Finished writing
process: " + newPayload);
                        Files.writeString(filePath, newPayload);
            }
            default -> {
                        break;
            }
            }
            }

            else if (operation.equals("Q")) {
            break;
            }

            else {
            System.out.println("Incorrect input...");
            }

            }

            //take more user input
```

```
            }

            public void recordInterviewNotes() {
            /* TODO
            - Find given file name
            - Append to file contents:
            //
            <InterviewerName>'s Notes:
                        //insert something here
                        .
                        .
                        .
                                    //
            - Save the file
            - the file should be moved to complete after this
            */
            }

            private void moveInterviewToComplete(String
interviewID) {
            /* TODO
            - takes interviewID from user, finds the txt file, then
moves it to the completed folder after:
            - changing the name to be the candidate's name
appended by interviewID
            Everything else related to the handover process
will be accomplished by other methods
            */
            }

            public void printInterviewNotes() {

            }

            public void printInterview(String interviewID)
throws IOException {
            System.out.println("Interview ID: " + interviewID);
            try (DirectoryStream<Path> directoryStream =
Files.newDirectoryStream(folderPathSchedules)) {
            for (Path file : directoryStream) {
            String fileName = file.getFileName().toString();
            if (fileName.matches(".*_" + interviewID + ".txt")) {
            Files.lines(file).forEach(System.out::println);
            break;
            }
            }
//          for (Path statusDir : directoryStream) {
//          if (Files.isDirectory(statusDir)) { // Ensure it's a
directory
//                      System.out.println("Executing first if
statement from statusDir: " + statusDir);
//                      try (DirectoryStream<Path> filesStream
= Files.newDirectoryStream(statusDir)) {
//                      for (Path file : filesStream) {
//                      System.out.println(file.toString());
//                      if (Files.isRegularFile(file)) { // Ensure it's
a file
//                      String fileName =
file.getFileName().toString();
//                      System.out.println(fileName);
//                      if (fileName.matches(".*_" + interviewID
```

```
+ ".txt")) { // Match <timeslot>_<interviewID>.txt
//                                  // Print file content
//                                  System.out.println("Found file:
" + fileName);
//                                  System.out.println("Content:");
//
Files.lines(file).forEach(System.out::println);
//                                  return;
//                          }
//                          }
//                          }
//                          }
//              }
//              }
                }
                // If the file is not found
                }

                public void printAllInterviews() {}

                public void printAllInterviewsByInterviewer() {}

                private void incrementInterviewID() {
                /* TODO
                - Open save file for counter
                - if no save file, create one
                - check bytes of file
                */
                }
}
package src.HR;

import src.App;
import src.HR.src.*;
import src.Security.src.SecurityRequestScheduler;

import java.io.IOException;
import java.nio.file.Path;
import java.util.Scanner;

public class HRDepartment {

                public void start() throws Exception {
                fileStorageHR storage = new fileStorageHR();
                System.out.println("Current file path is: " +
storage.getFilepath());
                employeeRecordManager empHandler = new
employeeRecordManager(storage);
                candidateRecordManager canHandler = new
candidateRecordManager(storage);
                hiringProcess hireHandler = new hiringProcess();

                boolean loop = true;

                //start of user interaction
                while(loop) {
                /*
                TODO:
                - Add in new logic capabilities
                - Replace with container system so i dont have to
deal with the switch cases
                - Replace old logic with new updated methods
                - decide whether altering the file path can be done
by user
                - Add in method to transfer Candidate to Employee
                */
                System.out.println("Welcome to the HR
Department!");
                System.out.println("Please choose from these
options:");

                //employee
                System.out.println("1. Add Employee");
                System.out.println("2. Remove Employee");
                System.out.println("3. Retrieve Employee");
                System.out.println("4. Update Employee");
                System.out.println("5. Display All Employees");
                System.out.println("6. Display All Employees By
Department");

                //candidate
                System.out.println("7. Add Candidate");
                System.out.println("8. Remove Candidate");
                System.out.println("9. Retrieve Candidate");
                System.out.println("10. Update Candidate");
                System.out.println("11. Display All Candidates");
                System.out.println("12. Display All Candidates By
Status");

                //hiring
                System.out.println("13. Create Interview Time
Slot");
                System.out.println("14. Display Interview Time
Slot");

                SecurityRequestScheduler scheduler = new
SecurityRequestScheduler();
                scheduler.optionsPrint();



                System.out.println("0. Exit");


                Scanner input = new Scanner(System.in);
                int choice = input.nextInt();
                switch (choice) {
                case 1: //add employee TODO: extract into
methods
                //Name
                System.out.println("Enter employee name: ");
                String name = input.nextLine();
                if(name.equals(" ") || name.isEmpty() ||
name.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        name = input.nextLine();
                }
                System.out.println("Is " + name + " correct? (y/n):
");
                String answer = input.next();
                if(answer.equals("n") || answer.equals("N")) {
```

```
                        System.out.println("Please re-enter
employee name: ");
                        name = input.nextLine();
                }

                //ID
                System.out.println("Enter employeeID: ");
                String employeeId = input.next();
                if(employeeId.equals(" ") || employeeId.isEmpty() ||
employeeId.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        employeeId = input.nextLine();
                }
                System.out.println("Is " + employeeId + " correct?
(y/n): ");
                answer = input.next();
                if(answer.equals("n") || answer.equals("N")) {
                        System.out.println("Please re-enter
employeeID: ");
                        employeeId = input.nextLine();
                }

                //Department
                Department department;
                System.out.println("Enter employee department:
");
                for(int i = 0; i < Department.values().length; i++) {

System.out.println(Department.values()[i].name());
                }
                String initialDep = input.nextLine();
                if(initialDep.equals(" ") || initialDep.isEmpty() ||
initialDep.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        initialDep = input.nextLine();
                        //TODO
                        if (initialDep.contains("Human
Resources")
                        || initialDep.contains("human
resources")) {
                        department =
Department.HUMAN_RESOURCES;
                        } else {
                        department =
Department.valueOf(initialDep.toUpperCase());
                        }
                }
                else if (initialDep.contains("Human Resources")
                        || initialDep.contains("human
resources")) {
                        department =
Department.HUMAN_RESOURCES;
                }

                else {
                        department =
Department.valueOf(initialDep.toUpperCase());
                }
                System.out.println("Is " + department + " correct?
(y/n): ");
                answer = input.next();
                if(answer.equals("n") || answer.equals("N")) {
                        System.out.println("Please re-enter
Department: ");
                        initialDep = input.next();
                        if(initialDep.equals(" ") ||
initialDep.isEmpty() || initialDep.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        initialDep = input.nextLine();
                        }
                        else if (initialDep.equals("Human
Resources")
                        || initialDep.equals("human resources"))
{
                        department =
Department.HUMAN_RESOURCES;
                        }
                        else {
                        department =
Department.valueOf(initialDep.toUpperCase());
                        }
                //Department department =
Department.valueOf(initialDep.toUpperCase());
//                      System.out.println("Is " + department + "
correct? (y/n): ");
//                      answer = input.next();
//                      if(answer.equals("n") ||
answer.equals("N")) {
//                      System.out.println("Please re-enter
Department: ");
//                      initialDep = input.next();
//                      if(initialDep.equals(" ") ||
initialDep.isEmpty() || initialDep.equals("\n")) {
//                      System.out.println("Incorrect input,
please try again: ");
//                      initialDep = input.nextLine();
//                      }
//                      else if (initialDep.equals("Human
Resources")
//                      || initialDep.equals("human resources"))
{
//                      initialDep = "Human_Resources";
//                      }
//                      department =
Department.valueOf(initialDep.toUpperCase());
                }

                //Position
                System.out.println("Enter employee position: ");
                String position = input.nextLine();
                if(position.equals(" ") || position.isEmpty() ||
position.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        position = input.nextLine();
                }
                System.out.println("Is " + position + " correct?
(y/n): ");
```

```
            answer = input.next();
            if(answer.equals("n") || answer.equals("N")) {
                        System.out.println("Please re-enter
employee position: ");
                        position = input.nextLine();
            }
            //Status
            System.out.println("Enter employee employment
status (i.e. onboarding): ");
            String employmentStatus =
input.next().toUpperCase();
            if(employmentStatus.equals(" ") ||
employmentStatus.isEmpty() ||
employmentStatus.equals("\n")) {
                        System.out.println("Incorrect input,
please try again: ");
                        employmentStatus =
input.next().toUpperCase();
            }
            System.out.println("Is " + employmentStatus + "
correct? (y/n): ");
            answer = input.next();
            if(answer.equals("n") || answer.equals("N")) {
                        System.out.println("Please re-enter
employment status: ");
                        employmentStatus = input.nextLine();
            }
            //Salary
            System.out.println("Enter employee salary: ");
            int salary = input.nextInt();
            System.out.println("Is " + salary + " correct? (y/n):
");
            answer = input.next();
            if(answer.equals("n") || answer.equals("N")) {
                        System.out.println("Please re-enter
employeeID: ");
                        salary = input.nextInt();
            }

            //Adding employee
            Employee employeeHolder = new
Employee(employeeId, name, department, position,
employmentStatus, salary);
            empHandler.addEmployee(employeeHolder);
            System.out.println("Employee added successfully!
Returning to menu...\n\n\n\n");
            break;

            case 2: //remove employee
            System.out.println("Enter employee ID: ");
            String markedEmployee = input.next();
            empHandler.removeEmployee(markedEmployee);
            System.out.println("Employee removed
successfully! Returning to menu...\n\n\n\n");
            break;

            case 3: //retrieve employee
            System.out.println("Enter Employee ID: ");
            String markedEmpID = input.next();
            Path currentEmpFile =
empHandler.findEmployeeFile(markedEmpID);
storage.loadFileAndPrint(currentEmpFile.toString());
            System.out.println("Employee retrieved
successfully! Returning to menu...\n\n\n\n");
            break;

            case 4: //update employee //TODO working i think
            System.out.println("Enter employee ID: ");
            String updateEmployeeID = input.next();

empHandler.updateEmployee(updateEmployeeID);
            System.out.println("Employee updated
successfully! Returning to menu...\n\n\n\n");
            break;

            case 5: //display all employees
            System.out.println("List of All Employees: ");
            for (Department department1 :
Department.values()) {

empHandler.displayEmployeesByDepartment(department1);
            }
            System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
            break;

            case 6: //display employees in a department
            System.out.print("Please enter Department folder
to list: ");
            Department departmentFolder =
Department.valueOf(input.next().toUpperCase());
            System.out.println();

empHandler.displayEmployeesByDepartment(departmentFol
der);
            System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
            break;

            case 7: //add candidate
            System.out.println("Enter candidate name: ");
            String candidateName = input.next();
            System.out.println("Enter candidateId");
            String candidateId = input.next();
            System.out.println("Enter position candidate
applied for: ");
            String positionApplied = input.next();
            System.out.print("Enter candidate status:
\nApplied\nApproved\nHiring\nPending\nRejected\nEnter
Here: ");
            candidateStatus candidateStatus =
src.HR.src.candidateStatus.valueOf(input.next().toUpperCas
e());
            Candidate newCandidate = new
Candidate(candidateId, candidateName, positionApplied,
candidateStatus);
            canHandler.addCandidate(newCandidate);
            System.out.println("Candidate added successfully!
Returning to menu...\n\n\n\n");
            break;
```

```
            case 8: //remove candidate
            System.out.println("Enter candidate ID to be
removed: ");
            String candidateID = input.next();
            canHandler.removeCandidate(candidateID);
            System.out.println("Candidate removed
successfully! Returning to menu...\n\n\n\n");
            break;

            case 9: //retrieve candidate by id
            System.out.println("Enter Candidate ID: ");
            String candidateID3 = input.next();
            Path currentCandidateFile =
canHandler.findCandidateFile(candidateID3);

storage.loadFileAndPrint(currentCandidateFile.toString());
            break;

            case 10: //update candidate
            System.out.println("Enter Candidate ID: ");
            String candidateID2 = input.next();
            canHandler.updateCandidate(candidateID2);
            System.out.println("\nReturning to
menu...\n\n\n\n");
            break;

            case 11: //list all candidates
            System.out.println("List of All Candidates: ");
            for (src.HR.src.candidateStatus candidate :
src.HR.src.candidateStatus.values()) {

canHandler.displayCandidatesByStatus(candidate + "\n");
            }
            System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
            break;

            case 12: //list candidates by status
            System.out.print("Please enter Status folder to list:
");
            String statusFolder = input.next().toUpperCase();
            System.out.println();

canHandler.displayCandidatesByStatus(statusFolder);
            System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
            break;

            case 13:
            hireHandler.createInterview();
            System.out.println("Interview created successfully!
Returning to menu...\n\n\n\n");
            break;

            case 14:
            System.out.println("Enter Interview ID: ");
            String interviewID = input.next();
            hireHandler.printInterview(interviewID);
            System.out.println("END OF LIST, returning to
menu...\n\n\n\n");
            break;

            case 111:
            scheduler.addSecurityRequest();
            break;

            case 112:
            scheduler.showAllSecurityRequests();
            break;

            case 113:
            scheduler.deleteScheduleByID();
            break;

            case 0:
            loop = false;
            input.close();
            System.out.println("EXITING...");
            App.prompt(); //<---- Kicks you back to the main
homepage
            break;
            }
            }
            }

            public Employee getEmployee(String employeeID)
throws IOException {
            //TODO
            return null;
            }

            public Employee getEmployee(Department
department, String name) {
//          TODO: FILL OUT
            return null;
            }

}
```

Manufacturing Department[Doyle Chism]:

```
    }

    @Override
    public void setProducts(Product product) {
        this.product = product;
    }

    @Override
    public void setCustomProducts(CustomProduct customProduct) {
        this.customProduct = customProduct;
    }


    @Override
    public void collectRawMaterials(Map<String, Object> materials) {
        if (selectedDesignDetails == null || selectedDesignDetails.isEmpty())
{
```

```
package src.Manufacturing.src;

import src.Design.src.CustomDesign;
import src.Design.src.FinalDesign;
import src.Manufacturing.src.interfaces.ProductInterface;

import java.util.HashMap;
import java.util.Map;

public class CustomProduct implements ProductInterface {

    CustomDesign design;
    String name;
    String description;
    String quantity;
    String category;
    String price;

    public CustomProduct(String name) {
```

```
            System.out.println("No materials were collected");
            return;
        }
        System.out.println("Collecting raw materials for: " +
selectedDesignName);
        Object rawMaterials =
selectedDesignDetails.get(selectedDesignName); //or RawMaterials
        if (rawMaterials == null) {
            System.out.println("No materials were collected");
            return;
        }
        List<String> requiredMaterials = new ArrayList<>();
        if (rawMaterials instanceof String) {
            requiredMaterials = List.of(((String) rawMaterials).split(","));
        } else if (rawMaterials instanceof List) {
            requiredMaterials = (List<String>) rawMaterials;
        } else {
            System.out.println("Invalid raw materials were collected");
            return;
        }
        for (String material : requiredMaterials) {
            if (materials.containsKey(material)) {
                collectedMaterials.put(material,
materials.get(material).toString());
            } else {
                System.out.println("Material " + material + " does not exist");
            }
        }
        System.out.println("Collected materials: " + collectedMaterials);
    }

    @Override
    public Map<String, String> getCollectedMaterials() {
        return collectedMaterials;
    }

    public String getProductName() {
        return product.getName();
    }

}

package src.Manufacturing.src;

import src.Design.src.FinalDesign;
import src.Manufacturing.src.interfaces.ManufacturingReportInterface;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.text.ParseException;
import java.time.Duration;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.HashMap;
import java.util.Map;

public class ManufacturingReport implements
ManufacturingReportInterface {

    private LocalDateTime startTime;
    private LocalDateTime endTime;
    private LocalDateTime startCollecting;
    private LocalDateTime verificationTime;
    private String designName;
    private final DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

    public ManufacturingReport() {
    }


    @Override
    public void setStartTime(String startTime) throws ParseException {
        this.startTime = parseDateTime(startTime);

    }

    @Override
        this.name = name;
    }

    @Override
    public void setName(String name) {
        if (this.design == null) {
            this.design = new CustomDesign(name);
        } else {
            this.design.setDesignName(name);
        }
        this.name = name;
    }

    @Override
    public String getName() {
        return this.design.getDesignName();
    }


    @Override
    public void setDescription(String description) {
        this.description = description;

    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void setQuantity(String quantity) {
        this.quantity = quantity;

    }

    @Override
    public String getQuantity() {
        return quantity;
    }

    @Override
    public void setCategory(String category) {

        this.category = category;
    }

    @Override
    public String getCategory() {
        return category;
    }

    @Override
    public String displayProducts() {
        return "Design Name: " + design.getDesignName() + "\n"
                + "Description: " + description + "\n"
                + "Quantity: " + quantity + "\n"
                + "Category: " + category;
    }

    @Override
    public void setPrice(String price) {
        this.price = price;
    }

    @Override
    public String getPrice() {
        return price;
    }


}
package src.Manufacturing.src;

import src.Manufacturing.src.interfaces.MachineOperations;

public class Machines implements MachineOperations {
```

```
    public String getStartTime() {
        return startTime.format(formatter);
    }

    @Override
    public void setEndTime(String endTime) throws ParseException {
        this.endTime = parseDateTime(endTime);
    }

    @Override
    public String getEndTime() {
        return endTime.format(formatter);
    }

    @Override
    public String getCollectingMaterialsTime() {
        return startCollecting.format(formatter);
    }

    @Override
    public void setCollectingMaterialsTime(String collectingMaterialsTime)
throws ParseException {
        this.startCollecting = parseDateTime(collectingMaterialsTime);
    }

    @Override
    public String getVerifiedMaterialsTime() {
        return verificationTime.format(formatter);
    }

    @Override
    public void setVerifiedMaterialsTime(String verifiedMaterialsTime)
throws ParseException {
        this.verificationTime = parseDateTime(verifiedMaterialsTime);
    }

    @Override
    public Duration calculateTimeTakeToCollectMaterials() {
        return Duration.between(startCollecting, verificationTime);
    }

    @Override
    public Duration calculateManufacturingTime() {
        return Duration.between(startTime, endTime);
    }

    private LocalDateTime parseDateTime(String input) throws
ParseException {
        try {
            return LocalDateTime.parse(input, formatter);
        } catch (Exception e) {
            System.out.println("Invalid time format");
            throw new IllegalArgumentException("Failed to parse date-time: "
+ input, e);
        }
    }

    @Override
    public void displayManufacturingReport() {
        System.out.println("Manufacturing Report");
        System.out.println("Time when finished collecting Raw Materials: " +
(startCollecting != null ? getCollectingMaterialsTime() : "Not Set"));
        System.out.println("Time when finished verified Materials: " +
(verificationTime != null ? getVerifiedMaterialsTime() : "Not Set"));
        System.out.println("Time when starting Production: " + (startTime !=
null ? getStartTime() : "Not Set"));
        System.out.println("Time when finishing Production: " + (endTime !=
null ? getEndTime() : "Not Set"));
        //call to show the time difference between actions
        if (startCollecting != null && verificationTime != null) {
            Duration collectionToVerification =
calculateTimeTakeToCollectMaterials();
            System.out.println("Time Difference from collecting Raw Materials
to verifying Raw Materials: " +
                    collectionToVerification.toHours() + " hours " +
                    collectionToVerification.toMinutesPart() + " minutes");
        } else {
            System.out.println("Time Difference from verified Materials to
verifying Raw Materials");
    private boolean running;


    @Override
    public void startMachine() {
        this.running = true;
        System.out.println("Starting Machine");
    }

    @Override
    public boolean isRunning() {
        return running;
    }
}
package src.Manufacturing.src;

import src.App;
import src.Design.src.FinalDesign;
import src.Inventory.src.BasicStorageManage;
import src.Inventory.src.interfaces.InventoryController;

import java.util.*;

public class ManufacturingController {

    private ManufacturingManager manager;
    private HeadOfManufacturing headOfManufacturing;
    private Machines machine;
    private boolean isReady = false;
    private boolean productCreated = false;
    private final ManufacturingFileManager fileManager = new
ManufacturingFileManager();
    private Map<String, Object> selectedDesign = null;
    private Map<String, Object> selectedCSTMDesign = null;
    private String designType = null;
    private Map<String, Object> newFinalDesign = null;
    private BasicStorageManage storageManage;
    private ManufacturingReport manufacturingReport;
    private final String manRepo = "Manufacturing/repository/";


    public ManufacturingController() {
        this.headOfManufacturing = new HeadOfManufacturing();
        this.machine = new Machines();
        this.manager = new ManufacturingManager();
        this.manufacturingReport = new ManufacturingReport();
        this.storageManage = new BasicStorageManage();


    }

    public void run() throws Exception {

        Scanner sc = new Scanner(System.in);
        boolean exit = false;
        System.out.println("Welcome to Manufacturing System");

        while (!exit) {
            System.out.println("Please enter your choice");
            System.out.println("1. Receive and send specifications for
Final Design");
            System.out.println("2. Receive and send specifications for
Custom Design");
            System.out.println("3. Collect Raw Materials for Designs");
            System.out.println("4. Verify Raw Materials for Designs");
            System.out.println("5. Create Product and store in Inventory");
            System.out.println("6. View Manufacturing Report");
            System.out.println("7. Exit");

            int choice = sc.nextInt();
            sc.nextLine();
            Map<String, Object> collectedMaterials = new HashMap<>();

            switch (choice) {
                case 1:
                    System.out.println("Accessing Final Designs from
repository...");
                    System.out.println(".   .   .");
                    Map<String, Object> storedDesigns =
```

```
        }

        if (startTime != null && endTime != null) {
            Duration productionDuration = calculateManufacturingTime();
            System.out.println("Product Production Duration: " +
productionDuration.toHours() + " hours " +
                    productionDuration.toMinutesPart() + " minutes");
        } else {
            System.out.println("Time Difference from start and end time of
production.");
        }


    }

    public Map<String, Object> toMap(String designName) {
        Map<String, Object> result = new HashMap<>();

        result.put("DesignName", designName);//return name from final
Design
        result.put("StartTime", getStartTime());
        result.put("EndTime", getEndTime());
        result.put("CollectingMaterialsTime", getCollectingMaterialsTime());
        result.put("VerifiedMaterialsTime", getVerifiedMaterialsTime());
        result.put("Verification Process",
calculateTimeTakeToCollectMaterials());
        result.put("Manufacturing Process Time",
calculateManufacturingTime());
        return result;
    }

    public void saveReportToFile(String filePath, String designName) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter(filePath, true))) { // Append mode
            writer.write("Manufacturing Report\n");
            writer.write("----------------------------\n");
            writer.write("Design Name: " + designName + "\n");
            writer.write("Start Time: " + (startTime != null ? getStartTime() :
"Not Set") + "\n");
            writer.write("End Time: " + (endTime != null ? getEndTime() : "Not
Set") + "\n");
            writer.write("Collecting Materials Time: " + (startCollecting != null ?
getCollectingMaterialsTime() : "Not Set") + "\n");
            writer.write("Verification Time: " + (verificationTime != null ?
getVerifiedMaterialsTime() : "Not Set") + "\n");

//            writer.write("\nCollected Materials:\n");
//            for (Map.Entry<String, String> entry :
collectedMaterials.entrySet()) {
//                writer.write(entry.getKey() + ": " + entry.getValue() + "\n");
//            }

            writer.write("\nProcess Durations:\n");
            if (startCollecting != null && verificationTime != null) {
                writer.write("Time Taken to Collect Materials: " +
calculateTimeTakeToCollectMaterials().toHours() + " hours " +
                        calculateTimeTakeToCollectMaterials().toMinutesPart() + "
minutes\n");
            } else {
                writer.write("Time taken to collect Materials Missing Data");
            }

            if (startTime != null && endTime != null) {
                writer.write("Manufacturing Process Time: " +
calculateManufacturingTime().toHours() + " hours " +
                        calculateManufacturingTime().toMinutesPart() + "
minutes\n");
            } else {
                writer.write("Time taken for Manufacturing Process Missing
Data");
            }
            writer.write("----------------------------\n");
            System.out.println("Manufacturing report successfully saved to: "
+ filePath);
        } catch (IOException e) {
            System.err.println("Error saving manufacturing report to file: " +
e.getMessage());
        }
    }
```

```
fileManager.getFinalDesign();
                if (storedDesigns == null || storedDesigns.isEmpty()) {
                    System.out.println("No final design are found");
                    return;
                }
                System.out.println("Available Final Designs: ");
                List<String> finalDesignNames = new
ArrayList<>(storedDesigns.keySet());
                for (int i = 0; i < finalDesignNames.size(); i++) {
                    System.out.println((i + 1) + ". " +
finalDesignNames.get(i));
                }
                System.out.println("Select a Final Design: ");
                int finalDesignChoice = sc.nextInt() - 1;
                sc.nextLine();
                if (finalDesignChoice < 0 || finalDesignChoice >
finalDesignNames.size()) {
                    System.out.println("Invalid choice");
                    return;
                }

                String selectedDesignName =
finalDesignNames.get(finalDesignChoice);
                selectedDesign = (Map<String, Object>)
storedDesigns.get(selectedDesignName);
                System.out.println("Selected: " + selectedDesignName);
                selectedDesign.forEach((key, value) -> {
                    System.out.println(key + ": " + value);
                });
                //give information to the manager to then collect the raw
materials properly.
                manager.setSelectedDesign(selectedDesignName,
selectedDesign);
                System.out.println("Final Design has been passed to the
manager...");
                System.out.println(".   .   .");
                break;

            case 2:
                System.out.println("Accessing Custom Designs from
repository...");
                System.out.println(".   .   .");
                Map<String, Object> storedCustomDesigns =
fileManager.getCustomDesign();
                if (storedCustomDesigns == null ||
storedCustomDesigns.isEmpty()) {
                    System.out.println("No final design are found");
                    return;
                }
                System.out.println("Available Custom Designs: ");
                List<String> customDesignNames = new
ArrayList<>(storedCustomDesigns.keySet());
                for (int i = 0; i < customDesignNames.size(); i++) {
                    System.out.println((i + 1) + ". " +
customDesignNames.get(i));
                }
                System.out.println("Select a Custom Design: ");
                int customDesignChoice = sc.nextInt() - 1;
                sc.nextLine();
                if (customDesignChoice < 0 || customDesignChoice >
customDesignNames.size()) {
                    System.out.println("Invalid choice");
                    return;
                }
                String selectedCustomDesignName =
customDesignNames.get(customDesignChoice);
                selectedCSTMDesign = (Map<String, Object>)
storedCustomDesigns.get(selectedCustomDesignName);
                System.out.println("Selected: " +
selectedCustomDesignName);
                selectedCSTMDesign.forEach((key, value) -> {
                    System.out.println(key + ": " + value);
                });


manager.setSelectedDesign(selectedCustomDesignName,
selectedCSTMDesign);
                System.out.println("Custom Design has been passed to
the manager...");
```

```
}

package src.Manufacturing.src;

import src.Design.src.FinalDesign;
import src.Manufacturing.src.interfaces.ProductInterface;

import java.util.HashMap;
import java.util.Map;

public class Product implements ProductInterface {

    FinalDesign design;
    String name;
    String description;
    String quantity;
    String category;
    String price;

    public Product(String name) {
        this.name = name;

    }

    @Override
    public void setName(String name) {

        if (this.design == null) {
            this.design = new FinalDesign(name);
        } else {
            this.design.setDesignName(name);
        }
        this.name = name;
    }

    @Override
    public String getName() {
        return design.getDesignName();
    }

    @Override
    public void setDescription(String description) {
        this.description = description;

    }

    @Override
    public String getDescription() {
        return description;
    }

    @Override
    public void setQuantity(String quantity) {
        this.quantity = quantity;
    }

    @Override
    public String getQuantity() {
        return quantity;
    }

    @Override
    public void setCategory(String category) {
        this.category = category;
    }

    @Override
    public String getCategory() {
        return category;
    }

    @Override
    public String displayProducts() {
        return "Design Name: " + name + "\n"
                + "Description: " + description + "\n"
                + "Quantity: " + quantity + "\n"
                + "Category: " + category;
    }
```

```
                System.out.println(".   .   .");
                break;


            case 3:
                System.out.println("Collecting the Raw Materials from
Storage...");
                System.out.println("Select the design type to collect raw
materials");
                System.out.println("1. Final Design");
                System.out.println("2. Custom Design");
                int designChoice = sc.nextInt();
                sc.nextLine();
                Map<String, Object> activeDesign =
manager.getSelectedDesignDetails();

                if (activeDesign == null) {
                    System.out.println("No active design was given to the
manager to collect the materials");
                    return;
                }
                //make sure it is the correct type
                if (designChoice == 1) {
                    System.out.println("Accessing Final Designs from
repository...");
                    Map<String, Object> finalDesigns =
fileManager.getFinalDesign();
                    if (finalDesigns == null || finalDesigns.isEmpty()) {
                        System.out.println("No final design are found");
                        return;
                    }
                    System.out.println("Selected Design: ");
                    activeDesign.forEach((key, value) -> {
                        System.out.println(key + ": " + value);
                    });
                    Object requiredRawMaterials =
activeDesign.get("rawMaterials");
                    List<String> rawMaterialsList = requiredRawMaterials
instanceof String ?
                            List.of(((String) requiredRawMaterials).split(",")) :
new ArrayList<>();
                } else if (designChoice == 2) {
                    System.out.println("Accessing Custom Designs from
repository...");
                    Map<String, Object> customDesigns =
fileManager.getCustomDesign();
                    if (customDesigns == null || customDesigns.isEmpty()) {
                        System.out.println("No final design are found");
                        return;
                    }
                    System.out.println("Selected Custom Design: ");
                    activeDesign.forEach((key, value) -> {
                        System.out.println(key + ": " + value);
                    });
                    Object requiredCustomMaterials =
customDesigns.get("customDesign");
                    List<String> customMaterialList =
requiredCustomMaterials instanceof String
                            ? List.of(((String)
requiredCustomMaterials).split(",")) : new ArrayList<>();
                }

                while (true) {
                    System.out.println("Enter 'done' to exit prompt at any
time");
                    System.out.println("Collect the type of Raw Material
based on the Design: ");
                    String rawMaterial = sc.nextLine().trim();
                    if (rawMaterial.equalsIgnoreCase("done")) {
                        break;
                    }

                    System.out.println("Select Quantity of Raw Materials
based on the Design: " + rawMaterial);
                    String quantity = sc.nextLine();
                    collectedMaterials.put(rawMaterial, quantity);
                    System.out.println("You have " + quantity + " items
collected of " + rawMaterial);
                    //Enter the time at which the raw materials were
```

```
    @Override
    public void setPrice(String price) {
        this.price = price;
    }

    @Override
    public String getPrice() {
        return price;
    }


}

package src.Manufacturing.src;

import src.Design.src.CustomDesign;
import src.Design.src.DesignSketch;
import src.Design.src.FinalDesign;
import src.Design.src.MarketingDesign;
import src.TextEditor.PoorTextEditor;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

public class ManufacturingFileManager {

    private final PoorTextEditor editor = new PoorTextEditor();
    private final String manRepo = "Manufacturing/repository/";
    private final String designRepo = "Design/repository/";
    private final String inventoryRepo = "Inventory/repository/";


    private Map<String, Object> sketches = new HashMap<>();
    private Map<String, Object> finalDesign = new HashMap<>();
    private Map<String, Object> customDesign = new HashMap<>();
    private Map<String, Object> marketingDesign = new HashMap<>();
    private Map<String, Object> manufacturingReport = new
HashMap<>();


    /*
    Initialize all files based on design type
     */
    public void initialize() {
        loadFromFile("sketches.txt", sketches);
        loadFromFile("finalDesign.txt", finalDesign);
        loadFromFile("customDesign.txt", customDesign);
        loadFromFile("marketingDesign.txt", marketingDesign);
    }

    /*
    load From files and map to the editor
     */
    private void loadFromFile(String fileName, Map<String, Object> maps)
{
        File file = new File(designRepo + fileName);
        if (file.exists()) {
            editor.processTextFile(file.getPath()); //check path type
            maps.putAll(editor.getRepository());
        } else {
            System.out.println("File not found: " + file.getPath());
        }
    }

    private void loadFromManufacturingFile(String fileName, Map<String,
Object> maps) {
        File file = new File(manRepo + fileName);
        if (file.exists()) {
            editor.processTextFile(file.getPath());
            maps.putAll(editor.getRepository());
        } else {
            System.out.println("File not found: " + file.getPath());
        }
    }

    /*
    save text type to file
```

```
collected
                    System.out.println("Enter the current time at which the
Raw Materials were collected.");
                    System.out.println("(yyyy-MM-dd HH:mm:ss): )");
                    String collectingTime = sc.nextLine();

manufacturingReport.setCollectingMaterialsTime(collectingTime);
                    System.out.println("Time has been set.");
                    String reportFilePathCollect = manRepo +
"ManufacturingReport.txt";

manufacturingReport.saveReportToFile(reportFilePathCollect,
activeDesign.toString());
                }

                manager.collectRawMaterials(collectedMaterials);
                System.out.println("Raw Materials Collected:");
                collectedMaterials.forEach((rawMaterial, quantity) -> {
                    System.out.println(quantity + " items of " +
rawMaterial);
                });
                Map<String, Map<String, String>> formattedMaterials =
new HashMap<>();
                Map<String, Object> finalData =
fileManager.getFinalDesign();
                Map<String, Object> customData =
fileManager.getCustomDesign();
                if (finalData != null || customData != null ||
finalData.isEmpty() || customData.isEmpty()) {
                    finalData.forEach((key, value) -> {
                        if (value instanceof Map) {
                            @SuppressWarnings("unchecked")
                            Map<String, String> details = (Map<String,
String>) value;
                            formattedMaterials.put(key, details);
                        }
                    });
                    customData.forEach((key, value) -> {
                        if (value instanceof Map) {
                            @SuppressWarnings("unchecked")
                            Map<String, String> details = (Map<String,
String>) value;
                            formattedMaterials.put(key, details);
                        }
                    });
                }
                collectedMaterials.forEach((rawMaterial, quantity) -> {
                    Map<String, String> details = new HashMap<>();
                    formattedMaterials.computeIfAbsent(rawMaterial, k ->
new HashMap<>()).put("Quantity", String.valueOf(quantity));
//                    formattedMaterials.put(rawMaterial, details);
                });

                fileManager.saveToFile("RawMaterials.txt", new
HashMap<>(formattedMaterials));
                System.out.println("Raw Materials Saved to repository");

                break;
            case 4:
                System.out.println("Verify the Raw Materials from
Storage");

                Map<String, Object> storedRawMaterials =
fileManager.getRawMaterials();
                if (storedRawMaterials == null ||
storedRawMaterials.isEmpty()) {
                    System.out.println("You have no items collected yet.");
                    return;
                }

                System.out.println("Stored Raw Materials: ");
                storedRawMaterials.forEach((key, value) -> {
                    System.out.println("Material: " + key + ", Quantity: " +
value);
                });

//
headOfManufacturing.viewRawMaterials(manager.getCollectedMateri
als());
```

```
    */
    public void saveToFile(String fileName, Map<String, Object> maps) {

        String newFilePath;
        if (fileName.equals("Products.txt") ||
fileName.equals("CustomProducts.txt")) {
            newFilePath = inventoryRepo + fileName;
        } else {
            newFilePath = manRepo + fileName;
        }
        File file = new File(newFilePath);
        File parent = file.getParentFile();
        if (!parent.exists() && !parent.mkdirs()) {
            System.err.println("Couldn't create directory: " + parent.getPath());
            return;
        }
        Map<String, Object> existingData = new HashMap<>();
        if (file.exists()) {
            editor.processTextFile(file.getPath());
            existingData.putAll(editor.getRepository());
        }
        if (fileName.equals("RawMaterials.txt")) {
//            Map<String, Object> rawMaterialsOnly = new HashMap<>();
            maps.forEach((key, value) -> {
                if (value instanceof Map && ((Map<?, ?>)
value).containsKey("Quantity")) {
                    existingData.put(key, value);
                }
            });
//            editor.setRepository(rawMaterialsOnly);
        } else if (fileName.equals("Products.txt") ||
fileName.equals("CustomProducts.txt")) {
            maps.forEach((key, value) -> {
                if (value instanceof Map) {
                    existingData.put(key, value);
                } else {
                    Map<String, String> productData = new HashMap<>();
                    productData.put("Details", value.toString());
                    existingData.put(key, productData);
                }
            });
        } else if (fileName.equals("ManufacturingReport.txt")) {
            maps.forEach((key, value) -> {
                if (value instanceof Map) {
                    existingData.put(key, value);
                } else {
                    Map<String, String> productData = new HashMap<>();
                    productData.put("Details", value.toString());
                    existingData.put(key, productData);
                }
            });
        } else {
            existingData.putAll(maps);
        }
        editor.setRepository(existingData);
        editor.writeToTextFile(file.getPath());
    }


    /*
    save a Design Sketch
     */
    public void saveSketch(DesignSketch sketch) {
        Map<String, String> mappedSketch = sketch.mapObjects();
        System.out.println("Mapped Data: " + mappedSketch);
        sketches.put("Sketch " + sketch.getDesignName(), mappedSketch);
        saveToFile("sketches.txt", sketches);
    }

    /*
    Return the sketch
     */
    public Map<String, Object> getSketch() {
        return new HashMap<>(sketches);
    }

    public void saveFinalDesign(FinalDesign design) {
        finalDesign.put("Final Design " + design.getDesignName(),
design.mapObjects());
```

```
                for (Map.Entry<String, Object> entry :
storedRawMaterials.entrySet()) {
                    String rawMaterial = entry.getKey();
                    String quantity = entry.getValue().toString();

                    System.out.println("Do you want to verify " + quantity +
" items of " + rawMaterial + "? (Y/N)");
                    String verifyRawMaterial = sc.nextLine();

                    if (verifyRawMaterial.equals("Y")) {
                        headOfManufacturing.selectRawMaterial(quantity,
rawMaterial);
                        System.out.println("You have " + quantity + " items of
" + rawMaterial + " verified.");
                        collectedMaterials.put(rawMaterial, quantity);
                        isReady = true;
                        //Enter the verified time for which the materials were
verified.
                        System.out.println("Enter the time at which the Raw
Materials were verified.");
                        System.out.println("(yyyy-MM-dd HH:mm:ss): )");
                        String verifiedTime = sc.nextLine();

manufacturingReport.setVerifiedMaterialsTime(verifiedTime);
                        System.out.println("Time has been set.");
                        String reportFilePathRaw = manRepo +
"ManufacturingReport.txt";

manufacturingReport.saveReportToFile(reportFilePathRaw,
rawMaterial);


                    } else if (verifyRawMaterial.equals("N")) {
                        System.out.println("Material " + rawMaterial + " is not
verified");
                        isReady = false;
                    } else {
                        System.out.println("Invalid input, select Y or N");
                    }
                }
//                fileManager.saveToFile("RawMaterials.txt",
collectedMaterials);
                break;
            case 5:

                System.out.println("Start to create a product");
                System.out.println("Select the type of product to create.");
                System.out.println("1. Product");
                System.out.println("2. Custom Product");
                int productChoice = sc.nextInt();
                sc.nextLine();
                if (!isReady) {
                    System.out.println("the raw materials have not been
verified yet");
                    break;
                }
                if (!machine.isRunning()) {
                    System.out.println("Machines are not currently
running.");
                    System.out.println("Starting the machines....");
                    machine.startMachine();
                }
                if (!machine.isRunning()) {
                    System.out.println("Machines are not currently
running.");
                    return;
                }
//                FinalDesign design =
createFinalDesign(selectedDesign);
                System.out.println("Machine is running");
                System.out.println(".....");
                System.out.println("Product is being created...");
                System.out.println(".....");
                System.out.println("Enter the start time for production: ");
                System.out.println("(yyyy-MM-dd HH:mm:ss): )");
                String startTime = sc.nextLine();
                manufacturingReport.setStartTime(startTime);
                System.out.println("Time has been set.");
                String reportFilePathStart = manRepo +
```

```
        saveToFile("finalDesign.txt", finalDesign);
    }

    public Map<String, Object> getRawMaterials() {
        Map<String, Object> rawMaterials = new HashMap<>();
        loadFromManufacturingFile("rawMaterials.txt", rawMaterials);
        return rawMaterials;
    }

    public Map<String, Object> getFinalDesign() {
        Map<String, Object> finalDesigns = new HashMap<>();
        loadFromFile("finalDesign.txt", finalDesigns);
        return finalDesigns;
    }

    public void saveCustomDesign(CustomDesign design) {
        customDesign.put("Custom Design " + design.getDesignName(),
design.mapObjects());
        saveToFile("customDesign.txt", customDesign);
    }

    public Map<String, Object> getCustomDesign() {
        Map<String, Object> customDesigns = new HashMap<>();
        loadFromFile("customDesign.txt", customDesigns);
        return customDesigns;
    }

    public void saveMarketingDesign(MarketingDesign design) {
        marketingDesign.put("Marketing Design " +
design.getDesignSketchName(), design.mapObjects());
        saveToFile("marketingDesign.txt", marketingDesign);
    }

    public Map<String, Object>
getManufacturingReport(ManufacturingReport report) {
        Map<String, Object> manufacturingReport = new HashMap<>();
        loadFromFile("ManufacturingReport.txt", manufacturingReport);
        return manufacturingReport;
    }


}

package src.Manufacturing.src;

import src.Manufacturing.src.interfaces.ManagerInterface;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/*
@author
 */
public class ManufacturingManager implements ManagerInterface {

    private Map<String, String> collectedMaterials = new HashMap<>();
    private Product product;
    private String selectedDesignName;
    private Map<String, Object> selectedDesignDetails = new
HashMap<>();
    private CustomProduct customProduct;


    public void setSelectedDesign(String designName, Map<String,
Object> designDetails) {
        this.selectedDesignName = designName;
        this.selectedDesignDetails = designDetails;
    }

    public Map<String, Object> getSelectedDesignDetails() {
        return selectedDesignDetails;
    }

    @Override
    public boolean createProduct(Map<String, Object> verifiedMaterials) {

        System.out.println("Creating product with the following materials: ");
"ManufacturingReport.txt";

manufacturingReport.saveReportToFile(reportFilePathStart,
"designName");
                Map<String, Map<String, String>> productShipped = new
HashMap<>();
                if (productChoice == 1) {
                    System.out.println("Log Product Details for Inventory:
");
                    if (selectedDesign == null) {
                        System.out.println("No design was selected, select
one first");
                        break;
                    }
                    selectedDesign.forEach((key, value) -> {
                        System.out.println(key + ": " + value);
                    });
                    System.out.println("Begin listing Product information: ");
                    System.out.println("Enter Product Name: ");
                    String productName = sc.nextLine();
                    System.out.println("Enter Product Description: ");
                    String productDescription = sc.nextLine();
                    System.out.println("Enter Product Quantity: ");
                    String productQuantity = sc.nextLine();
                    System.out.println("Enter Product Category: ");
                    String productCategory = sc.nextLine();
                    System.out.println("Enter Product Price: ");
                    String productPrice = sc.nextLine();

                    Product product = new Product(productName);
                    product.setName(productName);
                    product.setDescription(productDescription);
                    product.setQuantity(productQuantity);
                    product.setCategory(productCategory);
                    product.displayProducts();
                    product.setPrice(productPrice);
                    manager.setProducts(product);

                    Map<String, Object> productDetails = new
HashMap<>();
                    productDetails.put("Name", product.getName());
                    productDetails.put("Description",
product.getDescription());
                    productDetails.put("Quantity", product.getQuantity());
                    productDetails.put("Category", product.getCategory());
                    productDetails.put("Price", product.getPrice());

                    //send to Inventory for productShipped map
                    Map<String, String> productShippedDetails = new
HashMap<>();
                    productShippedDetails.put("Quantity",
product.getQuantity());
                    productShippedDetails.put("Description",
product.getDescription());
                    productShipped.put(product.getName(),
productShippedDetails);

storageManage.unLoadShipment(productShipped);//call to inventory

                    fileManager.saveToFile("Products.txt", productDetails);
                    productCreated =
manager.createProduct(collectedMaterials);
                    System.out.println("Product created");
                    System.out.println("Enter the end time of Production: ");
                    System.out.println("(yyyy-MM-dd HH:mm:ss): )");
                    String endTime = sc.nextLine();
                    manufacturingReport.setEndTime(endTime);
                    System.out.println("Time has been set.");
                    String reportFilePath1 = manRepo +
"ManufacturingReport.txt";

manufacturingReport.saveReportToFile(reportFilePath1,product.getNa
me());
                } else if (productChoice == 2) {
                    System.out.println("Log Custom Product Details for
Inventory: ");
                    if (selectedCSTMDesign == null) {
                        System.out.println("No design was selected, select
one first");
```

```
        verifiedMaterials.forEach((material, value) ->
System.out.println(value + "items of " + material));
//         this.product = new Product(verifiedMaterials.toString());
        return true;
    }

    @Override
    public void deliverProduct(Product product) {

        if (product == null) {
            System.out.println("Product has not been created");
            return;
        }
        System.out.println("Delivered to the head of manufacturing
complete!");
```

```
                    break;
                }
                selectedCSTMDesign.forEach((key, value) -> {
                    System.out.println(key + ": " + value);
                });
                System.out.println("Enter Product Name: ");
                String productName = sc.nextLine();
                System.out.println("Enter Product Description: ");
                String productDescription = sc.nextLine();
                System.out.println("Enter Product Quantity: ");
                String productQuantity = sc.nextLine();
                System.out.println("Enter Product Category: ");
                String productCategory = sc.nextLine();
                System.out.println("Enter Product Price: ");
                String productPrice = sc.nextLine();
                CustomProduct customProduct = new
CustomProduct(productName);
                customProduct.setName(productName);
                customProduct.setDescription(productDescription);
                customProduct.setQuantity(productQuantity);
                customProduct.setCategory(productCategory);
                customProduct.setPrice(productPrice);
                customProduct.displayProducts();

                manager.setCustomProducts(customProduct);
                Map<String, Object> customProductDetails = new
HashMap<>();
                customProductDetails.put("Name",
customProduct.getName());
                customProductDetails.put("Description",
customProduct.getDescription());
                customProductDetails.put("Quantity",
customProduct.getQuantity());
                customProductDetails.put("Category",
customProduct.getCategory());
                customProductDetails.put("Price",
customProduct.getPrice());
                //sending to inventory
                Map<String, String> productShippedDetails = new
HashMap<>();
                productShippedDetails.put("Quantity",
customProduct.getQuantity());
                productShippedDetails.put("Description",
customProduct.getDescription());
                productShipped.put(customProduct.getName(),
productShippedDetails);

storageManage.unLoadShipment(productShipped);//call to inventory

                fileManager.saveToFile("CustomProducts.txt",
customProductDetails);
                productCreated =
manager.createProduct(collectedMaterials);
                System.out.println("Custom Product created");
                System.out.println("Enter the end time of production for
a Custom Product: ");
                System.out.println("(yyyy-MM-dd HH:mm:ss): )");
                String endTime = sc.nextLine();
                manufacturingReport.setEndTime(endTime);
                System.out.println("Time has been set.");
                String reportFilePath2 = manRepo +
"ManufacturingReport.txt";

manufacturingReport.saveReportToFile(reportFilePath2,customProduc
t.getName());

            }
            break;
        case 6:
            System.out.println("Manufacturing Report List");

System.out.println("-----------------------------------------------------------");
            manufacturingReport.displayManufacturingReport();


        case 7:
            System.out.println("Exit Program");
            exit = true;
//              App.prompt();
```

```
                break;

            default:
                System.out.println("Please enter a valid choice");
        }
    }

    sc.close();
  }
}
```

Design Department[Doyle Chism]:

```
package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;
import src.Design.src.interfaces.HeadOfDesignInterface;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class CustomDesign implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private String quantity;

    public CustomDesign(String designName) {
        this.designName = designName;
    }

    @Override
    public void setColor(List<String> colors) {
        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(String quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
    public List<String> getColors() {
        return colors;
    }
```

```
package src.Design.src;


//import src.App;

import src.App;
import src.Design.src.interfaces.DesignSpecifications;
import src.Design.src.interfaces.HeadOfDesignInterface;

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

public class DesignSpecificationsController {


    private HeadOfDesignTeam headOfDesignTeam;
    private List<DesignSketch> sketches;
    private FinalDesign finalDesign;
    private CustomDesign customDesign;
    private MarketingDesign marketingDesign;
    private final DesignFileManager designFileManager;

    public DesignSpecificationsController() {
        this.sketches = new ArrayList<>();
        headOfDesignTeam = new HeadOfDesignTeam(sketches); // they
ask for sketches
        this.designFileManager = new DesignFileManager();
        designFileManager.initialize();
    }


    public void run() throws Exception {


        Scanner scan = new Scanner(System.in);
        boolean exit = false;
        System.out.println("Welcome to the Design Management
System.");

        while (!exit) {
            System.out.println("1. Create a Design Sketch");
            System.out.println("2. View and Select a Design Sketch");
            System.out.println("3. Delete a Design Sketch");
            System.out.println("4. Delete a Final Design");
            System.out.println("5. Set a Final Design for Manufacturing");
            System.out.println("6. Create and Set a Final Design for
Marketing");
            System.out.println("7. Create and Set a custom Design for
Modelling");
            System.out.println("8. Exit Program");
```

```
    @Override
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public String getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }

    @Override
    public String displayAllSpecifications() {
        return
                "Display all design specifications\n" +
                        "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;

    }

    @Override
    public Map<String, String> mapObjects() {
        Map<String, String> map = new HashMap<>();
        map.put("DesignName", designName);
        map.put("DesignImage", designImage);
        map.put("DesignColors", String.join(",", colors));
        map.put("DesignRawMaterials", String.join(",", rawMaterials));
        map.put("DesignSizes", String.join(",", sizes));
        map.put("DesignQuantities", quantity);

        return map;
    }
}
```

```
package src.Design.src;

import src.TextEditor.PoorTextEditor;

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class DesignFileManager {

    private Map<String, Object> sketches = new HashMap<>();
    private Map<String, Object> finalDesign = new HashMap<>();
    private Map<String, Object> customDesign = new HashMap<>();
    private Map<String, Object> marketingDesign = new HashMap<>();


    private final PoorTextEditor editor = new PoorTextEditor();
    private final String repo = "Design/repository/";

    /*
    Initialize all files based on design type
     */
```

```
            int option = scan.nextInt();
            scan.nextLine();

            switch (option) {
                case 1:
                    System.out.println("Begin Listing the Specifications for the
design sketch.");
                    System.out.println("If entering multiple entries input as such
S,M,L with no spaces");
                    System.out.println("Enter Design Name: ");
                    String designName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String rawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for design: ");
                    String colorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String sizesNeeded = scan.nextLine();
                    System.out.println("Enter design quantity: ");
                    String designQuantity = scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String imageOfDesign = scan.nextLine();
                    //call to helper to set all specifications
                    createDesignSketch(designName, rawMaterialsNeeded,
colorsNeeded, sizesNeeded, designQuantity, imageOfDesign);
                    break;
                case 2:
//                    headOfDesignTeam.viewSketches(sketches);
                    Map<String, Object> storedSketches =
designFileManager.getSketch(); //check to see if it retrieves all the
sketches
                    if (storedSketches == null || storedSketches.isEmpty()) {
                        System.out.println("No sketches found in repository.");
                        return;
                    }
                    System.out.println("Available Design Sketches: ");
                    List<String> sketchPosition = new
ArrayList<>(storedSketches.keySet());
                    for (int i = 0; i < sketchPosition.size(); i++) {
                        System.out.println((i + 1) + ". " + sketchPosition.get(i));
                    }

                    //ask for input
                    System.out.println("Select the Design Sketch to verify by
the number");
                    int sketchNumber = scan.nextInt() - 1;
                    scan.nextLine();

                    if (sketchNumber < 0 || sketchNumber >=
sketchPosition.size()) {
                        System.out.println("Invalid sketch number");
                        return;
                    }

                    String selectedSketchName =
sketchPosition.get(sketchNumber);
//                    DesignSketch selectedSketch = (DesignSketch)
storedSketches.get(selectedSketchName);
                    Map<String, Object> selectedSketch = (Map<String,
Object>) storedSketches.get(selectedSketchName);

                    if (selectedSketch == null) {
                        System.out.println("No sketch found with name " +
selectedSketchName);
                        return;
                    }
                    System.out.println("You selected the following sketch: " +
selectedSketchName);
                    selectedSketch.forEach((key, value) -> {
                        System.out.println(key + ": " + value);
                    });

                    String sketchName = (String) selectedSketch.get("Design
Name");
                    String sketchRawMaterials = (String)
selectedSketch.get("Design Raw Materials");
                    String sketchColors = (String) selectedSketch.get("Design
Colors");
                    String sketchSizes = (String) selectedSketch.get("Design
Size");
```

```
    public void initialize() {
        loadFromFile("sketches.txt", sketches);
        loadFromFile("finalDesign.txt", finalDesign);
        loadFromFile("customDesign.txt", customDesign);
        loadFromFile("marketingDesign.txt", marketingDesign);
    }

    /*
    load From files and map to the editor
     */
    private void loadFromFile(String fileName, Map<String, Object> maps)
{
        File file = new File(repo + fileName);
        if (file.exists()) {
            editor.processTextFile(file.getAbsolutePath()); //check path type
            maps.putAll(editor.getRepository());
        }
    }

    /*
    save text type to file
     */
    public void saveToFile(String fileName, Map<String, Object> maps) {

        String newFilePath = repo + fileName;
        File file = new File(newFilePath);

        File parent = file.getParentFile();
        if (!parent.exists() && !parent.mkdirs()) {
            System.err.println("Couldn't create directory: " +
parent.getAbsolutePath());
            return;
        }
        editor.setRepository(maps);
        editor.writeToTextFile(file.getAbsolutePath());
    }

    /*
    save a Design Sketch
     */
    public void saveSketch(DesignSketch sketch) {
        Map<String, String> mappedSketch = sketch.mapObjects();
        System.out.println("Mapped Data: " + mappedSketch);
        sketches.put("Sketch " + sketch.getDesignName(), mappedSketch);
        saveToFile("sketches.txt", sketches);
    }

//    public void deleteSketchFromList(String sketchName) {
//
//        String sketchToRemove = null;
//
//
//        for (Map.Entry<String, Object> entry : sketches.entrySet()) {
//            if (entry.getKey().equals(sketchName) && entry.getValue()
instanceof DesignSketch) {
//                sketchToRemove = entry.getKey();
//                break;
//            }
//        }
//
//        // If sketch is found, remove it
//        if (sketchToRemove != null) {
//            sketches.remove(sketchToRemove); // Remove from the list in
memory
//            System.out.println("Sketch '" + sketchName + "' has been
removed from memory.");
//
//            // Update the file using DesignFileManager
//            deleteSketch(sketchToRemove);
//        } else {
//            System.out.println("Sketch '" + sketchName + "' was not found in
memory.");
//        }
//    }

    public void deleteSketch(DesignSketch sketch) {


        if (sketch == null) {
```

```
                String sketchQuantities = (String)
selectedSketch.get("Design Quantities");
                String sketchDesignImage = (String)
selectedSketch.get("Design Image");

                List<String> rawMaterialsList = sketchRawMaterials != null
&& !sketchRawMaterials.isEmpty() ?
                        List.of(sketchRawMaterials.split(",")) : new
ArrayList<>();
                List<String> colorList = sketchColors != null &&
!sketchColors.isEmpty() ?
                        List.of(sketchColors.split(",")) : new ArrayList<>();
                List<String> sizeList = sketchSizes != null &&
!sketchSizes.isEmpty() ?
                        List.of(sketchSizes.split(",")) : new ArrayList<>();


                DesignSketch newSketch = new
DesignSketch(selectedSketchName);
                newSketch.setDesignName(sketchName);
                newSketch.setRawMaterials(rawMaterialsList);
                newSketch.setColor(colorList);
                newSketch.setSizes(sizeList);
                newSketch.setQuantities(sketchQuantities);
                newSketch.setDesignImage(sketchDesignImage);
                sketches.add(newSketch);



                int sketchIndex = sketches.indexOf(newSketch);
                if (sketchIndex != -1) {
                    headOfDesignTeam.selectSketch(sketchIndex,
sketches);
                }

//                headOfDesignTeam.selectSketch(sketchNumber,
sketches);
                System.out.println("(Y/N) Do you want to verify the design
sketch?");
                String verifyDesign = scan.nextLine();
                if (verifyDesign.equals("Y")) {
                    //call to begin Final Design case 3

//                    finalDesign = new FinalDesign(selectedSketchName);
//
headOfDesignTeam.selectSketch(sketches.indexOf(newSketch),
sketches);
                    finalDesign = headOfDesignTeam.confirmFinalDesign();
                    if (finalDesign != null) {
                        designFileManager.saveFinalDesign(finalDesign);
                        System.out.println("Display Verified Design\n" +
finalDesign.displayAllSpecifications());
                        System.out.println("Sketch has been selected for a
final design.");
                    } else {
                        System.out.println("Failed to confirm final design");
                    }

                } else if (verifyDesign.equals("N")) {
                    System.out.println("Design was not verified");
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
                break;
            case 3:
                Map<String, Object> deletedSketch =
designFileManager.getSketch(); //check to see if it retrieves all the
sketches
                if (deletedSketch == null || deletedSketch.isEmpty()) {
                    System.out.println("No sketches found in repository.");
                    return;
                }
                System.out.println("Available Design Sketches: ");
                List<String> deletedPosition = new
ArrayList<>(deletedSketch.keySet());
                for (int i = 0; i < deletedPosition.size(); i++) {
                    System.out.println((i + 1) + ". " + deletedPosition.get(i));
                }

                //ask for input
```

```
            System.out.println("sketch is null");
            return;
        }
        String designName = sketch.getDesignName();
        String key = "Sketch " + designName;
        File file = new File(repo + key);
        if (!file.exists()) {
            System.out.println("The file does not exist.");
            return;
        }
        List<String> allLines = new ArrayList<>();
        try {
            allLines = Files.readAllLines(file.toPath());
        } catch (IOException e) {
            System.err.println("Error reading file: " + e.getMessage());
            return;
        }

        if (sketches.containsKey(key)) {
            sketches.remove(key);
            editor.removeArrayItem(designName);
            saveToFile("sketches.txt", sketches);
            System.out.println("Sketch: " + designName + " has been
deleted.");
        } else {
            System.out.println("Sketch: " + designName + " does not exist.");
        }

    }


    public void deleteFinalDesign(FinalDesign finalDesign) {

    }

    /*
    Return the sketch
     */
    public Map<String, Object> getSketch() {
        return new HashMap<>(sketches);
    }

    public void saveFinalDesign(FinalDesign design) {
        finalDesign.put("Final Design " + design.getDesignName(),
design.mapObjects());
        saveToFile("finalDesign.txt", finalDesign);
    }

    public Map<String, Object> getFinalDesign() {
        return new HashMap<>(finalDesign);
    }

    public void saveCustomDesign(CustomDesign design) {
        customDesign.put("Custom Design " + design.getDesignName(),
design.mapObjects());
        saveToFile("customDesign.txt", customDesign);
    }

    public Map<String, Object> getCustomDesign() {
        return new HashMap<>(customDesign);
    }

    public void saveMarketingDesign(MarketingDesign design) {
        marketingDesign.put("Marketing Design " +
design.getDesignSketchName(), design.mapObjects());
        saveToFile("marketingDesign.txt", marketingDesign);
    }

    public Map<String, Object> getMarketingDesign() {
        return new HashMap<>(marketingDesign);
    }

    public void sendDataToRepo() {
        File f = new File(repo + "sketches.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "sketches.txt");
            sketches = editor.getRepository();
        }
        f = new File(repo + "finalDesign.txt");
            System.out.println("Select the Design Sketch to delete by
the number");
            int deletedNumber = scan.nextInt() - 1;
            scan.nextLine();

            if (deletedNumber < 0 || deletedNumber >=
deletedPosition.size()) {
                System.out.println("Invalid sketch number");
                return;
            }

            String deletedSketchName =
deletedPosition.get(deletedNumber);
//              DesignSketch selectedSketch = (DesignSketch)
deletedSketch.get(deletedSketchName);
            Map<String, Object> selectedDeletedSketch =
(Map<String, Object>) deletedSketch.get(deletedSketchName);

            if (selectedDeletedSketch == null) {
                System.out.println("No sketch found with name " +
deletedSketchName);
                return;
            }
            System.out.println("You selected the following sketch: " +
deletedSketchName);
            selectedDeletedSketch.forEach((key, value) -> {
                System.out.println(key + ": " + value);
            });

            String newSketchName = (String)
selectedDeletedSketch.get("Design Name");
            String deletedSketchRawMaterials = (String)
selectedDeletedSketch.get("Design Raw Materials");
            String deletedSketchColors = (String)
selectedDeletedSketch.get("Design Colors");
            String deletedSketchSizes = (String)
selectedDeletedSketch.get("Design Size");
            String deletedSketchQuantities = (String)
selectedDeletedSketch.get("Design Quantities");
            String deletedSketchDesignImage = (String)
selectedDeletedSketch.get("Design Image");

            List<String> rawMaterialsListDeleted =
deletedSketchRawMaterials != null &&
!deletedSketchRawMaterials.isEmpty() ?
                    List.of(deletedSketchRawMaterials.split(",")) : new
ArrayList<>();
            List<String> colorListDeleted = deletedSketchColors != null
&& !deletedSketchColors.isEmpty() ?
                    List.of(deletedSketchColors.split(",")) : new
ArrayList<>();
            List<String> sizeListDeleted = deletedSketchSizes != null
&& !deletedSketchSizes.isEmpty() ?
                    List.of(deletedSketchSizes.split(",")) : new
ArrayList<>();


            DesignSketch newSketchDeleted = new
DesignSketch(deletedSketchName);
            newSketchDeleted.setDesignName(newSketchName);

newSketchDeleted.setRawMaterials(rawMaterialsListDeleted);
            newSketchDeleted.setColor(colorListDeleted);
            newSketchDeleted.setSizes(sizeListDeleted);
            newSketchDeleted.setQuantities(deletedSketchQuantities);

newSketchDeleted.setDesignImage(deletedSketchDesignImage);
            sketches.add(newSketchDeleted);

            int sketchIndexDeleted =
sketches.indexOf(newSketchDeleted);
            if (sketchIndexDeleted != -1) {
                headOfDesignTeam.selectSketch(sketchIndexDeleted,
sketches);

            }
            //Prompt User to delete items
            System.out.println("(Y/N) Do you want to delete this design
sketch?");
```

```
        if (f.exists()) {
            editor.processTextFile(repo + "finalDesign.txt");
            finalDesign = editor.getRepository();
        }
        f = new File(repo + "customDesign.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "customDesign.txt");
            customDesign = editor.getRepository();
        }
        f = new File(repo + "marketingDesign.txt");
        if (f.exists()) {
            editor.processTextFile(repo + "marketingDesign.txt");
            marketingDesign = editor.getRepository();

        }
    }


}

package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;

public class DesignSketch implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private String quantity;

    public DesignSketch(String designName) {
        this.designName = designName;
    }

    @Override
    public void setColor(List<String> colors) {

        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(String quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
    public List<String> getColors() {
        return colors;
    }

    @Override
```

```
                String deletedResponse = scan.nextLine().trim();

                DesignSketch sketchToDelete = null;
                for (DesignSketch sketch : sketches) {
                    if
(sketch.getDesignName().equals(deletedSketchName)) {
                        sketchToDelete = sketch;
                        break;
                    }
                }
                if (deletedResponse.equalsIgnoreCase("Y")) {
                    // Call deleteSketch with the DesignSketch object
                    if (newSketchDeleted != null) {
                        designFileManager.deleteSketch(sketchToDelete);
                        sketches.removeIf(s ->
s.getDesignName().equals(newSketchDeleted.getDesignName())); //
Remove from local memory list
                        System.out.println("The selected design sketch has
been deleted.");
                    }
                } else if (deletedResponse.equalsIgnoreCase("N")) {
                    System.out.println("The design sketch was not
deleted.");
                    return;
                } else {
                    System.out.println("Invalid input. No action taken.");
                    return;
                }

                break;


            case 4:
                Map<String, Object> deletedFinalDesigns =
designFileManager.getFinalDesign();
                if (deletedFinalDesigns == null ||
deletedFinalDesigns.isEmpty()) {
                    System.out.println("No sketches found in repository.");
                    return;
                }
                System.out.println("Available Final Designs: ");
                List<String> deletedFinalDesignNames = new
ArrayList<>(deletedFinalDesigns.keySet());
                for (int i = 0; i < deletedFinalDesignNames.size(); i++) {
                    System.out.println((i + 1) + ". " +
deletedFinalDesignNames.get(i));
                }

                System.out.println("Select a Final Design for
Manufacturing");
                int designNumberDeleted = scan.nextInt() - 1;
                scan.nextLine();

                if (designNumberDeleted < 0 || designNumberDeleted >=
deletedFinalDesignNames.size()) {
                    System.out.println("Invalid design number");
                    return;
                }
                String selectedFinalDesignNameDeleted =
deletedFinalDesignNames.get(designNumberDeleted);
                Map<String, Object> selectedFinalDesignDeleted =
(Map<String, Object>)
deletedFinalDesigns.get(selectedFinalDesignNameDeleted);

                finalDesign = new
FinalDesign(selectedFinalDesignNameDeleted);
                finalDesign.setDesignName((String)
selectedFinalDesignDeleted.get("DesignName"));
                finalDesign.setRawMaterials(List.of((String)
selectedFinalDesignDeleted.get("DesignRawMaterials")));
                finalDesign.setColor(List.of((String)
selectedFinalDesignDeleted.get("DesignColors")));
                finalDesign.setSizes(List.of((String)
selectedFinalDesignDeleted.get("DesignSizes")));
                finalDesign.setQuantities((String)
selectedFinalDesignDeleted.get("DesignQuantities"));
                finalDesign.setDesignImage((String)
selectedFinalDesignDeleted.get("DesignImage"));
```

```
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public String getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }

    @Override
    public String displayAllSpecifications() {
        return
                "Display all design specifications\n" +
                        "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;
    }

    @Override
    public Map<String, String> mapObjects() {
        Map<String, String> map = new HashMap<>();

        map.put("Design Name", Objects.toString(this.designName, "null"));
        map.put("Design Raw Materials", this.rawMaterials != null ?
String.join(",", this.rawMaterials) : "null");
        map.put("Design Size", this.sizes != null ? String.join(",", this.sizes) :
"null");
        map.put("Design Quantities", Objects.toString(this.quantity, "null"));
        map.put("Design Colors", this.colors != null ? String.join(",",
this.colors) : "null");
        map.put("Design Image", Objects.toString(this.designImage, "null"));

//        map.put("DesignName", designName);
//        map.put("DesignImage", designImage);
//        map.put("DesignColors", String.join(",", colors));
//        map.put("DesignRawMaterials", String.join(",", rawMaterials));
//        map.put("DesignSizes", String.join(",", sizes));
//        map.put("DesignQuantities", quantity);

        return map;
    }
}
    @Override
    public FinalDesign confirmFinalDesign() {

        if (selectedSketch == null) {
            System.out.println("No sketch selected");
            return null;
        }
        FinalDesign finalDesign =
FinalDesign.fromDesignSketch(selectedSketch);

        finalDesign.setDesignName(selectedSketch.getDesignName());
        finalDesign.setColor(selectedSketch.getColors());
        finalDesign.setDesignImage(selectedSketch.getDesignImage());
        finalDesign.setSizes(selectedSketch.getSizes());
        finalDesign.setQuantities(selectedSketch.getQuantities());
        finalDesign.setRawMaterials(selectedSketch.getRawMaterials());
        finalDesign.displayAllSpecifications();

        return finalDesign;
                System.out.println("Current Final Design Specifications: \n"
+ finalDesign.displayAllSpecifications());
                //Prompt User to delete items
                System.out.println("(Y/N) Do you want to delete this design
sketch?");
                String deletedFinal = scan.nextLine().trim();

                DesignSketch designToDelete = null;
                for (DesignSketch sketch : sketches) {
                    if
(sketch.getDesignName().equals(finalDesign.getDesignName())) {
                        designToDelete = sketch;
                        break;
                    }
                }
                if (deletedFinal.equalsIgnoreCase("Y")) {
                    // Call deleteSketch with the DesignSketch object
                    if (finalDesign != null) {
                        designFileManager.deleteSketch(designToDelete);
                        sketches.removeIf(s ->
s.getDesignName().equals(finalDesign.getDesignName())); // Remove
from local memory list
                        System.out.println("The selected design sketch has
been deleted.");
                    }
                } else if (deletedFinal.equalsIgnoreCase("N")) {
                    System.out.println("The design sketch was not
deleted.");
                    return;
                } else {
                    System.out.println("Invalid input. No action taken.");
                    return;
                }

                break;

            case 5:
                Map<String, Object> storedFinalDesigns =
designFileManager.getFinalDesign();
                if (storedFinalDesigns == null ||
storedFinalDesigns.isEmpty()) {
                    System.out.println("No sketches found in repository.");
                    return;
                }
                System.out.println("Available Final Designs: ");
                List<String> finalDesignNames = new
ArrayList<>(storedFinalDesigns.keySet());
                for (int i = 0; i < finalDesignNames.size(); i++) {
                    System.out.println((i + 1) + ". " +
finalDesignNames.get(i));
                }

                System.out.println("Select a Final Design for
Manufacturing");
                int designNumber = scan.nextInt() - 1;
                scan.nextLine();

                if (designNumber < 0 || designNumber >=
finalDesignNames.size()) {
                    System.out.println("Invalid design number");
                    return;
                }
                String selectedFinalDesignName =
finalDesignNames.get(designNumber);
                Map<String, Object> selectedFinalDesign = (Map<String,
Object>) storedFinalDesigns.get(selectedFinalDesignName);

                finalDesign = new FinalDesign(selectedFinalDesignName);
                finalDesign.setDesignName((String)
selectedFinalDesign.get("DesignName"));
                finalDesign.setRawMaterials(List.of((String)
selectedFinalDesign.get("DesignRawMaterials")));
                finalDesign.setColor(List.of((String)
selectedFinalDesign.get("DesignColors")));
                finalDesign.setSizes(List.of((String)
selectedFinalDesign.get("DesignSizes")));
                finalDesign.setQuantities((String)
selectedFinalDesign.get("DesignQuantities"));
```

```
    }

    @Override
    public MarketingDesign confirmMarketingDesign() {

        if (marketingDesign == null) {
            System.out.println("No marketing design selected");
            return null;
        }
        MarketingDesign market = new MarketingDesign(finalDesign);

        market.setDesignSketchName(finalDesign.getDesignName());
        market.setPrice(marketingDesign.getPrice());

market.setProductDescription(marketingDesign.getProductDescription());
        market.setSeasonType(marketingDesign.getSeasonType());
        market.setTargetAudience(marketingDesign.getTargetAudience());
        market.displayAllSpecifications();

        return market;
    }

    @Override
    public CustomDesign confirmCustomDesign() {

        if (customDesign == null) {
            System.out.println("No custom design selected");
            return null;
        }
        CustomDesign custom = new
CustomDesign(customDesign.getDesignName());

        custom.setDesignName(customDesign.getDesignName());
        custom.setColor(customDesign.getColors());
        custom.setDesignImage(customDesign.getDesignImage());
        custom.setSizes(customDesign.getSizes());
        custom.setQuantities(customDesign.getQuantities());
        custom.setRawMaterials(customDesign.getRawMaterials());
        custom.displayAllSpecifications();

        return custom;
    }

}

package src.Design.src;

import src.Design.src.interfaces.MarketingDesignSpecifications;

import java.util.HashMap;
import java.util.Map;

public class MarketingDesign implements MarketingDesignSpecifications
{

    private FinalDesign finalDesign;
    private String targetAudience;
    private String price;
    private String description;
    private String seasonType;
    private String marketingDesign;

    public MarketingDesign(FinalDesign finalDesign) {
        this.finalDesign = finalDesign;

    }

    @Override
    public String getDesignSketchName() {
        return finalDesign.getDesignName();
    }

    @Override
    public void setDesignSketchName(String designSketchName) {
        this.finalDesign.setDesignName(designSketchName);
    }


    @Override
                finalDesign.setDesignImage((String)
selectedFinalDesign.get("DesignImage"));


                System.out.println("Current Final Design Specifications: \n"
+ finalDesign.displayAllSpecifications());
                System.out.println("Do you want to modify the current final
design?");
                System.out.println("(Y/N)");
                String modifyDesign = scan.nextLine();
                if (modifyDesign.equals("Y")) {
                    System.out.println("Set a Final Design and set
specifications of the final design");
                    System.out.println("Enter Design Name: ");
                    String finalDesignName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String finalRawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for design:
");
                    String finalColorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String finalSizesNeeded = scan.nextLine();
                    System.out.println("Enter design  quantity: ");
                    String finalDesignQuantity = scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String finalImageOfDesign = scan.nextLine();
                    setFinalDesign(finalDesignName,
finalRawMaterialsNeeded, finalColorsNeeded, finalSizesNeeded,
finalDesignQuantity, finalImageOfDesign);
                } else if (modifyDesign.equals("N")) {
                    System.out.println("No changes are needed in the Final
Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
                break;
            case 6:

                Map<String, Object> storedMarketingDesigns =
designFileManager.getFinalDesign();
                if (storedMarketingDesigns == null ||
storedMarketingDesigns.isEmpty()) {
                    System.out.println("No sketches found in repository.");
                    return;
                }
                System.out.println("Available Final Designs: ");
                List<String> finalMarketingNames = new
ArrayList<>(storedMarketingDesigns.keySet());
                for (int i = 0; i < finalMarketingNames.size(); i++) {
                    System.out.println((i + 1) + ". " +
finalMarketingNames.get(i));
                }

                System.out.println("Select a Final Design for Marketing");
                int marketingNumber = scan.nextInt() - 1;
                scan.nextLine();

                if (marketingNumber < 0 || marketingNumber >=
finalMarketingNames.size()) {
                    System.out.println("Invalid design number");
                    return;
                }
                String selectedMarketingDesignName =
finalMarketingNames.get(marketingNumber);
                Map<String, Object> selectedMarketingDesign =
(Map<String, Object>)
storedMarketingDesigns.get(selectedMarketingDesignName);

                finalDesign = new
FinalDesign(selectedMarketingDesignName);
                finalDesign.setDesignName((String)
selectedMarketingDesign.get("DesignName"));
                finalDesign.setRawMaterials(List.of((String)
selectedMarketingDesign.get("DesignRawMaterials")));
                finalDesign.setColor(List.of((String)
selectedMarketingDesign.get("DesignColors")));
                finalDesign.setSizes(List.of((String)
selectedMarketingDesign.get("DesignSizes")));
```

```
    public void setTargetAudience(String targetAudience) {
        this.targetAudience = targetAudience;

    }

    @Override
    public String getTargetAudience() {
        return targetAudience;
    }

    @Override
    public void setPrice(String price) {
        this.price = price;

    }

    @Override
    public String getPrice() {
        return price;
    }

    @Override
    public void setSeasonType(String seasonType) {
        this.seasonType = seasonType;
    }

    @Override
    public String getSeasonType() {
        return seasonType;
    }

    @Override
    public void setProductDescription(String productDescription) {
        this.description = productDescription;
    }

    @Override
    public String getProductDescription() {
        return description;
    }

    @Override
    public String displayAllSpecifications() {
        return "Display all specifications \n" + "Design sketch: " +
finalDesign.getDesignName() +
                "\n" + "Target Audience: " + targetAudience + "\n"
                + "Price: " + price + "\n" + "Season type: " + seasonType +
                "\n" + "Product description: " + description + "\n";
    }

    @Override
    public Map<String, Object> mapObjects() {
        Map<String, Object> map = new HashMap<>();
        map.put("sketch", finalDesign != null ? finalDesign.getDesignName()
: "");
        map.put("targetAudience", targetAudience);
        map.put("price", price);
        map.put("seasonType", seasonType);

        return map;
    }

}
```

```
                finalDesign.setQuantities((String)
selectedMarketingDesign.get("DesignQuantities"));
                finalDesign.setDesignImage((String)
selectedMarketingDesign.get("DesignImage"));
                System.out.println("Current Final Design Specifications: \n"
+ finalDesign.displayAllSpecifications());

                marketingDesign = new MarketingDesign(finalDesign);

                System.out.println("Do you want to begin setting the
marketing Design?");
                System.out.println("(Y/N)");
                String modifyDesign2 = scan.nextLine();
                if (modifyDesign2.equals("Y")) {
                    System.out.println("Set Marketing Specifications based
on Final Design: ");
                    System.out.println("Enter Marketing Design Name: ");
                    String marketingDesignName = scan.nextLine();
                    System.out.println("Enter Target Audience: ");
                    String audience = scan.nextLine();
                    System.out.println("Enter Marketing Price: ");
                    String price = scan.nextLine();
                    System.out.println("Enter Design Description: ");
                    String description = scan.nextLine();
                    System.out.println("Enter Season Type: ");
                    String seasonType = scan.nextLine();
                    setMarketingDesign(marketingDesignName, audience,
price, description, seasonType);
//
designFileManager.saveMarketingDesign(marketingDesign);
                    System.out.println("Selected Marketing Design
Specifications: \n" + marketingDesign.displayAllSpecifications());


                } else if (modifyDesign2.equals("N")) {
                    System.out.println("No changes are needed in the Final
Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
                break;
            case 7:
                System.out.println("Do you want to create a custom Design
for modelling?");
                System.out.println("(Y/N)");
                String response = scan.nextLine();
                if (response.equals("Y")) {
                    System.out.println("Set a Custom Design and set
specifications for the final design");
                    System.out.println("Enter Design Name: ");
                    String finalDesignName = scan.nextLine();
                    System.out.println("Enter raw materials needed: ");
                    String finalRawMaterialsNeeded = scan.nextLine();
                    System.out.println("Enter the colors you want for design:
");
                    String finalColorsNeeded = scan.nextLine();
                    System.out.println("Enter sizes for design: ");
                    String finalSizesNeeded = scan.nextLine();
                    System.out.println("Enter design quantity: ");
                    String finalDesignQuantity = scan.nextLine();
                    System.out.println("Enter image of the design: ");
                    String finalImageOfDesign = scan.nextLine();
                    //have to change to set custom design
                    setCustomDesign(finalDesignName,
finalRawMaterialsNeeded, finalColorsNeeded, finalSizesNeeded,
finalDesignQuantity, finalImageOfDesign);

                } else if (response.equals("N")) {
                    System.out.println("No changes are needed in the Final
Design");
                    break;
                } else {
                    System.out.println("Invalid Input. Select Y or N");
                }
                break;
            case 8:
                System.out.println("Exit Program");
                exit = true;
```

```
//                  App.prompt();
                    break;

                default:
                    System.out.println("Invalid option");
            }
        }
        scan.close();
    }

    //lists all the things needed to create a new sketch which will then be
reviewed
    public void createDesignSketch(String designName, String
rawMaterialsNeeded, String colorsNeeded, String sizesNeeded, String
designQuantity, String imageOfDesign) {

        List<String> rawMaterials = List.of(rawMaterialsNeeded.split(","));
        List<String> colors = List.of(colorsNeeded.split(","));
        List<String> sizes = List.of(sizesNeeded.split(","));
        //make sure I am calling this correctly
        DesignSketch sketch = new DesignSketch(designName);
        sketch.setDesignName(designName);
        sketch.setRawMaterials(rawMaterials);
        sketch.setColor(colors);
        sketch.setSizes(sizes);
        sketch.setQuantities(designQuantity);
        sketch.setDesignImage(imageOfDesign);
        sketches.add(sketch);
        sketch.displayAllSpecifications();
        designFileManager.saveSketch(sketch);
        System.out.println("Sketch created and added to a list of sketches
waiting to be approved");

    }

    public void setFinalDesign(String finalDesignName, String
finalRawMaterialsNeeded, String finalColorsNeeded, String
finalSizesNeeded, String finalDesignQuantity, String
finalImageOfDesign) {

        List<String> rawMaterials =
List.of(finalRawMaterialsNeeded.split(","));
        List<String> colors = List.of(finalColorsNeeded.split(","));
        List<String> sizes = List.of(finalSizesNeeded.split(","));

        finalDesign = new FinalDesign(finalDesignName);

        finalDesign.setDesignName(finalDesignName);
        finalDesign.setRawMaterials(rawMaterials);
        finalDesign.setColor(colors);
        finalDesign.setSizes(sizes);
        finalDesign.setQuantities(finalDesignQuantity);
        finalDesign.setDesignImage(finalImageOfDesign);
        designFileManager.saveFinalDesign(finalDesign);
        System.out.println("Final Design has been set with all
specifications");
        finalDesign.displayAllSpecifications();

    }

    public void setCustomDesign(String customDesignName, String
rawMaterialsNeeded, String colorsNeeded, String sizeNeeded, String
designQuantity, String imageOfDesign) {

        List<String> rawMaterials = List.of(rawMaterialsNeeded.split(","));
        List<String> colors = List.of(colorsNeeded.split(","));
        List<String> sizes = List.of(sizeNeeded.split(","));

        customDesign = new CustomDesign(customDesignName);
        customDesign.setRawMaterials(rawMaterials);
        customDesign.setColor(colors);
        customDesign.setSizes(sizes);
        customDesign.setQuantities(designQuantity);
        customDesign.setDesignImage(imageOfDesign);
        customDesign.displayAllSpecifications();
        designFileManager.saveCustomDesign(customDesign);
        System.out.println("Custom Design has been set with all
specifications");
        customDesign.displayAllSpecifications();
```

```
    }

    public void setMarketingDesign(String sketch, String targetAudience,
String price, String description, String seasonType) {

        marketingDesign = new MarketingDesign(finalDesign);
        marketingDesign.setDesignSketchName(sketch);
        marketingDesign.setPrice(price);
        marketingDesign.setProductDescription(description);
        marketingDesign.setSeasonType(seasonType);
        marketingDesign.setTargetAudience(targetAudience);
        designFileManager.saveMarketingDesign(marketingDesign);
        System.out.println("The Marketing Design has been set with all
specifications");
        marketingDesign.displayAllSpecifications();

    }


}
package src.Design.src;

import src.Design.src.interfaces.DesignSpecifications;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class FinalDesign implements DesignSpecifications {

    private String designName;
    private String designImage;
    private List<String> colors;
    private List<String> rawMaterials;
    private List<String> sizes;
    private String quantity;


    public FinalDesign(String designName) {
        this.designName = designName;
    }


    public static FinalDesign fromDesignSketch(DesignSketch sketch) {
        return new FinalDesign(sketch.getDesignName());
    }

    @Override
    public void setColor(List<String> colors) {
        this.colors = colors;
    }

    @Override
    public void setRawMaterials(List<String> rawMaterials) {
        this.rawMaterials = rawMaterials;
    }

    @Override
    public void setSizes(List<String> sizes) {
        this.sizes = sizes;
    }

    @Override
    public void setQuantities(String quantities) {
        this.quantity = quantities;
    }

    @Override
    public void setDesignName(String designName) {
        this.designName = designName;
    }

    @Override
    public void setDesignImage(String image) {
        this.designImage = image;
    }

    @Override
```

```
    public List<String> getColors() {
        return colors;
    }

    @Override
    public List<String> getRawMaterials() {
        return rawMaterials;
    }

    @Override
    public List<String> getSizes() {
        return sizes;
    }

    @Override
    public String getQuantities() {
        return quantity;
    }

    @Override
    public String getDesignName() {
        return designName;
    }

    @Override
    public String getDesignImage() {
        return designImage;
    }

    @Override
    public String displayAllSpecifications() {

        return
                "Design Name: " + designName + "\n" +
                        "Design Image: " + designImage + "\n" +
                        "Design Colors: " + colors + "\n" +
                        "Design Raw Materials: " + rawMaterials + "\n" +
                        "Design Sizes: " + sizes + "\n" +
                        "Design Quantities: " + quantity;
    }

    @Override
    public Map<String, String> mapObjects() {
        Map<String, String> map = new HashMap<>();
        map.put("DesignName", designName);
        map.put("DesignImage", designImage);
        map.put("DesignColors", String.join(",", colors));
        map.put("DesignRawMaterials", String.join(", ", rawMaterials));
        map.put("DesignSizes", String.join(", ", sizes));
        map.put("DesignQuantities", quantity);
        return map;
    }
}
package src.Design.src;

import src.Design.src.interfaces.*;

import java.util.List;

public class HeadOfDesignTeam implements HeadOfDesignInterface {


    private List<DesignSketch> allSketches;
    private DesignSketch selectedSketch;
    private CustomDesign customDesign;
    private MarketingDesign marketingDesign;
    private FinalDesign finalDesign;


    public HeadOfDesignTeam(List<DesignSketch> sketches) {
        this.allSketches = sketches;
    }


    @Override
    public void viewSketches(List<DesignSketch> sketches) {

        if (sketches.isEmpty()) {
            System.out.println("No sketch selected");
```

```
        return;
      }
      System.out.println("Select a sketch");
      for (int i = 0; i < sketches.size(); i++) {
        System.out.println((i + 1) + ". " +
sketches.get(i).getDesignName());
      }
    }

    @Override
    public void selectSketch(int sketchIndex, List<DesignSketch>
sketches) {

      if (sketches == null || sketches.isEmpty()) {
        System.out.println("No sketch selected");
        return;
      }
      if (sketchIndex >= 0 && sketchIndex < sketches.size()) {
        selectedSketch = sketches.get(sketchIndex);
        System.out.println("Selcted sketch was: " +
selectedSketch.getDesignName());
      } else {
        System.out.println("Incorrect Sketch selected, Try Again");
      }
    }
```

Public Relations Department [Kenny]:

```
/**
 * @author Kenny
 */

package src.PublicRelations.src;

import src.TextEditor.PoorTextEditor;

import java.io.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class ActivityScheduler implements
src.PublicRelations.src.interfaces.ActivityScheduler {

    private final String PRPlanningEmployeesDir =
"src/PublicRelations/repository/planningEmployees/";
    private final String PRReviewEmployeesDir =
"src/PublicRelations/repository/reviewEmployees/";
    private final String activityScheduleDir =
"src/PublicRelations/repository/activitySchedules/";
    private final String printedActivitySchedulesDir =
"src/PublicRelations/repository/printedActivitySchedules/";

    PoorTextEditor editor = new PoorTextEditor();

    // sorts content schedules by deadline first, then by status
    PriorityQueue<ActivitySchedules>
damageControlActivitySchedulesPriorityQueue = new
PriorityQueue<>(
            Comparator.<ActivitySchedules,
    LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
        );

    PriorityQueue<ActivitySchedules>
    planningActivitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ActivitySchedules,
    LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
        );

    PriorityQueue<ActivitySchedules>
    reviewingActivitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ActivitySchedules,
    LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
    DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
        );
```

```
    PriorityQueue<ActivitySchedules>
revisingActivitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ActivitySchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
    );

    PriorityQueue<ActivitySchedules>
readyActivitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ActivitySchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
    );

    PriorityQueue<ActivitySchedules>
allActivitySchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ActivitySchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparing(ActivitySchedules::getStatus)
    );
    /**
     * Repository to store all content schedules
     */
    Map<String, ActivitySchedules> activitySchedulesRepository =
new LinkedHashMap<>();

    // sorts employees by division, then position, then rating in
descending order
    PriorityQueue<PRPlanningEmployee>
availablePRPlanningEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(PRPlanningEmployee::getDivision)
                    .thenComparing(PRPlanningEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRPlanningEmployee>
allPRPlanningEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(PRPlanningEmployee::getDivision)
                    .thenComparing(PRPlanningEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRReviewEmployee>
availablePRReviewEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(PRReviewEmployee::getDivision)
                    .thenComparing(PRReviewEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRReviewEmployee>
allPRReviewEmployeePriorityQueue = new PriorityQueue<>(
            Comparator.comparing(PRReviewEmployee::getDivision)
                    .thenComparing(PRReviewEmployee::getPosition)
                    .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    /**
     * Repository to store all PR employees
     */
    Map<String, PRPlanningEmployee>
PRPlanningEmployeeRepository = new LinkedHashMap<>();
    Map<String, PRReviewEmployee> PRReviewEmployeeRepository
= new LinkedHashMap<>();

    @Override
    public boolean addPREmployee() {
        return false;
    }

    @Override
    public boolean deletePREmployee(String employeeID) {
        return false;
    }

    @Override
    public boolean showFreePlanningEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRPlanningEmployee> tempQueue = new
PriorityQueue<>(availablePRPlanningEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            planningEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showAllPlanningEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRPlanningEmployee> tempQueue = new
PriorityQueue<>(allPRPlanningEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            planningEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showFreeReviewEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRReviewEmployee> tempQueue = new
PriorityQueue<>(availablePRReviewEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            reviewEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showAllReviewEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }
```

```
        PriorityQueue<PRReviewEmployee> tempQueue = new
PriorityQueue<>(allPRReviewEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            reviewEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean createActivity() {

        Scanner scan = new Scanner(System.in);

        String scheduleID = "socialActivity-" + IDGenerator();

        System.out.println("Enter social activity status: (Recommended:
No status | Damage Control |---| Others: Planning | Reviewing |
Revising | Ready)");
        String status = scan.nextLine();

        String department = "Department of Public Relations";

        System.out.println("Enter social activity name: ");
        String activityName = scan.nextLine();

        System.out.println("Enter social activity type: " );
        String activityType = scan.nextLine();

        System.out.println("Enter social activity objective: ");
        String objective = scan.nextLine();

        System.out.println("Enter social activity target audience: ");
        String targetAudience = scan.nextLine();

        System.out.println("Enter social activity description: ");
        String description = scan.nextLine();

        System.out.println("Enter social activity tasks: ");
        String tasks = scan.nextLine();

        System.out.println("Enter social activity location: (Country - City -
Specific Location)");
        String location = scan.nextLine();

        System.out.println("Enter social activity duration: (MM-dd-yyyy -
MM-dd-yyyy)");
        String duration = scan.nextLine();

        System.out.println("Enter social activity deadline:
(MM-dd-yyyy)");
        String deadline = scan.nextLine();

        String dateScheduled = dateIssuer();

        ActivitySchedules schedule = new ActivitySchedules(
                scheduleID,
                activityName,
                activityType,
                objective,
                targetAudience,
                description,
                tasks,
                department,
                location,
                duration,
                deadline,
                status,
                dateScheduled,
                null);

        // assigning planning employee to schedule
        if (!addPlanningEmployeeToSchedule(schedule)){

            System.out.println("Planning employee assignment for activity
schedule ID: " + schedule.getScheduleID() + " was cancelled");
            return false;
        }

        // assigning review employee to schedule
        if (!addReviewEmployeeToSchedule(schedule)){

            System.out.println("Review employee assignment for activity
schedule ID: " + schedule.getScheduleID() + " was cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(activityScheduleToHashmap(schedule));
        editor.writeToTextFile(activityScheduleDir +
schedule.getScheduleID() + ".txt");

        // writing printed schedule to repository
        printActivitySchedule(schedule);

        return true;
    }

    @Override
    public boolean deleteActivity(String activityID) {

        if (!retrieveAllActivities()) return false;

        if (!retrieveAllPREmployees()) return false;

        if (!activitySchedulesRepository.containsKey(activityID)){

            System.out.println("Activity schedule ID: " + activityID + " not
found");
            return false;
        }

        ActivitySchedules schedule =
activitySchedulesRepository.get(activityID);

        if (schedule.getPlanningTeamAssign().isEmpty()||
schedule.getReviewTeamAssign().isEmpty()){

            System.out.println("Incomplete employee assignment for
activity schedule ID: " + activityID);
            return false;
        }

        /**
         * Freeing planning employees current assignment
         */
        List<String> planningEmployees =
schedule.getPlanningTeamAssign();
        editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
        for (String s : planningEmployees){
            editor.setValue(s, "currentAssignment", "free");
        }
        editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");

        /**
         * Freeing review employees current assignment
         */
```

```
        List<String> reviewEmployees =
schedule.getReviewTeamAssign();
        editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
        for (String s : reviewEmployees){
            editor.setValue(s, "currentAssignment", "free");
        }
        editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");

        /**
         * Deleting content schedule
         */
        File fileToDelete = new File(activityScheduleDir,
schedule.getFileName());

        if (fileToDelete.exists()){

            if (fileToDelete.delete()){

                System.out.println("Activity schedule of ID: " + activityID + "
successfully deleted");
            }
            else {

                System.out.println("Failed to delete activity schedule ID: " +
activityID);
                return false;
            }
        }
        else {

            System.out.println("Activity schedule with ID: " + activityID + "
not found");
            return false;
        }
        return true;
    }

    @Override
    public boolean editActivity(String activityID) {

        if (!retrieveAllActivities()){
            return false;
        }

        if (!activitySchedulesRepository.containsKey(activityID)){

            System.out.println("Activity schedule with ID: " + activityID + "
does not exists");
            return false;
        }

        ActivitySchedules schedule =
activitySchedulesRepository.get(activityID);

        Scanner scan = new Scanner(System.in);
        boolean exit = false;

        System.out.println("Editing activity schedule ID: " +
schedule.getScheduleID());
        while (!exit){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>>>>>>>");
            System.out.println("Select a category to edit: ");
            System.out.println("1. Activity Name | Current: " +
schedule.getActivityName());
            System.out.println("2. Activity Type | Current: " +
schedule.getActivityType());
            System.out.println("3. Objective | Current: " +
schedule.getObjective());
            System.out.println("4. Target Audience | Current: " +
schedule.getTargetAudience());
            System.out.println("5. Description | Current: " +
schedule.getDescription());
            System.out.println("6. Tasks | Current: " +
schedule.getTasks());
            System.out.println();
            System.out.println("7. Location | Current: " +
schedule.getLocation());
            System.out.println("8. Duration | Current: " +
schedule.getDuration());
            System.out.println("9. Deadline | Current: " +
schedule.getDeadline());
            System.out.println("10. Status | Current: " +
schedule.getStatus());
            System.out.println("11. Planning Team Assignment");
            System.out.println("12. Review Team Assignment");
            System.out.println("0. Save & Exit");

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<\n");

            String choice = scan.nextLine();

            switch (choice) {

                case "1" -> {
                    System.out.println("Enter social activity name: ");
                    schedule.setActivityName(scan.nextLine());
                }
                case "2" -> {
                    System.out.println("Enter social activity type: " );
                    schedule.setActivityType(scan.nextLine());
                }
                case "3" -> {
                    System.out.println("Enter social activity objective: ");
                    schedule.setObjective(scan.nextLine());
                }
                case "4" -> {
                    System.out.println("Enter social activity target audience:
");
                    schedule.setTargetAudience(scan.nextLine());
                }
                case "5" -> {
                    System.out.println("Enter social activity description: ");
                    schedule.setDescription(scan.nextLine());
                }
                case "6" -> {
                    System.out.println("Enter social activity tasks: ");
                    schedule.setTasks(scan.nextLine());
                }
                case "7" -> {
                    System.out.println("Enter social activity location:
(Country - City - Specific Location)");
                    schedule.setLocation(scan.nextLine());
                }
                case "8" -> {
                    System.out.println("Enter social activity duration:
(MM-dd-yyyy - MM-dd-yyyy)");
                    schedule.setDuration(scan.nextLine());
                }
                case "9" -> {
                    System.out.println("Enter social activity deadline:
(MM-dd-yyyy)");
                    schedule.setDeadline(scan.nextLine());
                }
```

```
            case "10" -> {
                System.out.println("Enter social activity status: (No status
| Damage Control | Planning | Reviewing | Revising | Ready)");
                schedule.setStatus(scan.nextLine());
            }
            case "11" -> {
                addPlanningEmployeeToSchedule(schedule);
            }
            case "12" -> {
                addReviewEmployeeToSchedule(schedule);
            }
            case "0" -> {
                exit = true;
            }
            default -> System.out.println("Invalid choice. Try again");
        }
    }

    // writing schedule to repository
    editor.setRepository(activityScheduleToHashmap(schedule));
    editor.writeToTextFile(activityScheduleDir +
schedule.getFileName());

    // writing printed schedule to repository
    printActivitySchedule(schedule);

    System.out.println("Update successful for activity schedule ID: "
+ activityID);
    return true;
  }

  @Override
  public boolean showAllActivities() {

    if (!retrieveAllActivities()){
      return false;
    }

    if (allActivitySchedulesPriorityQueue.isEmpty()){
      System.out.println("Nothing was found");
      return false;
    }

    PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(allActivitySchedulesPriorityQueue);

    while (!tempQueue.isEmpty()){
      activityScheduleToText(tempQueue.poll());
    }
    return true;
  }

  @Override
  public boolean showDamageControlActivities() {

    if (!retrieveAllActivities()){
      return false;
    }

    if (damageControlActivitySchedulesPriorityQueue.isEmpty()){
      System.out.println("Nothing was found");
      return false;
    }

    PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(damageControlActivitySchedulesPriorityQueue);

    while (!tempQueue.isEmpty()){
      activityScheduleToText(tempQueue.poll());
    }
    return true;
  }

  @Override
  public boolean showPlanningActivities() {

    if (!retrieveAllActivities()){
      return false;
    }

    if (planningActivitySchedulesPriorityQueue.isEmpty()){
      System.out.println("Nothing was found");
      return false;
    }

    PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(planningActivitySchedulesPriorityQueue);

    while (!tempQueue.isEmpty()){
      activityScheduleToText(tempQueue.poll());
    }
    return true;
  }

  @Override
  public boolean showReviewingActivities() {

    if (!retrieveAllActivities()){
      return false;
    }

    if (reviewingActivitySchedulesPriorityQueue.isEmpty()){
      System.out.println("Nothing was found");
      return false;
    }

    PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(reviewingActivitySchedulesPriorityQueue);

    while (!tempQueue.isEmpty()){
      activityScheduleToText(tempQueue.poll());
    }
    return true;
  }

  @Override
  public boolean showRevisingContents() {

    if (!retrieveAllActivities()){
      return false;
    }

    if (revisingActivitySchedulesPriorityQueue.isEmpty()){
      System.out.println("Nothing was found");
      return false;
    }

    PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(revisingActivitySchedulesPriorityQueue);

    while (!tempQueue.isEmpty()){
      activityScheduleToText(tempQueue.poll());
    }
    return true;
  }

  @Override
  public boolean showReadyContents() {

    if (!retrieveAllActivities()){
```

```
            return false;
        }

        if (readyActivitySchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ActivitySchedules> tempQueue = new
PriorityQueue<>(readyActivitySchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            activityScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showActivityByID(String activityID) {

        if (!retrieveAllActivities()){
            return false;
        }

        if (!activitySchedulesRepository.containsKey(activityID)){

            System.out.println("Activity schedule with ID: " + activityID + "
not found");
            return false;
        }


activityScheduleToText(activitySchedulesRepository.get(activityID));
        return true;
    }

    /**
     * To add planning employee to a schedule
     * @param schedule given schedule
     * @return true if successful, false otherwise
     */
    private boolean
addPlanningEmployeeToSchedule(ActivitySchedules schedule){

        if (!retrieveAllPREmployees()){
            return false;
        }

        List<String> selectedPlanningEmployeeIDs = new ArrayList<>();
        if (schedule.getPlanningTeamAssign() != null &&
!schedule.getPlanningTeamAssign().isEmpty()){
            selectedPlanningEmployeeIDs =
schedule.getPlanningTeamAssign();
        }

        Scanner scan = new Scanner(System.in);

        boolean endProgram = false;
        boolean returnValue = false;

        while (!endProgram){

            System.out.println("Planning Employee Assignment Manager:
");
            System.out.println("1. Add Planning Employee by ID");
            System.out.println("2. Remove Planning Employee by ID");
            System.out.println("3. Show Scheduled Planning
Employees");
            System.out.println("4. Complete Schedule");
            System.out.println("5. Cancel Schedule");
            System.out.println("0. Exit");

            String choice = scan.nextLine();

            switch (choice){

                case "1" -> {

                    System.out.println("Enter planning employee ID: ");
                    String employeeID = scan.nextLine();

                    if
(!PRPlanningEmployeeRepository.containsKey(employeeID)){

                        System.out.println("Planning employee with ID: " +
employeeID + " does not exists");
                        break;
                    }

                    PRPlanningEmployee employee =
PRPlanningEmployeeRepository.get(employeeID);

                    if (!employee.getCurrentAssignment().equals("free")){

                        System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                                "Overwrite? (y/n)\n" +
                                "!!!WARNING!!! This process is irreversible!");

                        String yesNo = scan.nextLine();

                        if (yesNo.equals("y")){


employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

                            if
(!selectedPlanningEmployeeIDs.contains(employee.getEmployeeID()
)){


selectedPlanningEmployeeIDs.add(employee.getEmployeeID());
                            }

                            // updating employee assignments to file
                            editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                            editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                            editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                            editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                        }
                    }
                    else {


availablePRPlanningEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedPlanningEmployeeIDs.add(employee.getEmployeeID());

                        // updating employee assignments to file
                        editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
```

```
                editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
            }
        }
        case "2" -> {

            if (selectedPlanningEmployeeIDs.isEmpty()){

                System.out.println("No planning employees to remove
from this schedule");
                break;
            }

            System.out.println("Enter planning employee ID to
remove: ");
            String removeID = scan.nextLine();

            if (!selectedPlanningEmployeeIDs.contains(removeID)){

                System.out.println("Planning employee ID: " +
removeID + " not assigned to this schedule");
                break;
            }

            selectedPlanningEmployeeIDs.remove(removeID);
            PRPlanningEmployee employee =
PRPlanningEmployeeRepository.get(removeID);
            employee.setCurrentAssignment("free");

availablePRPlanningEmployeePriorityQueue.add(employee);

            // updating employee assignments to file
            editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
            editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
            editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
        }
        case "3" -> {

            if (selectedPlanningEmployeeIDs.isEmpty()){

                System.out.println("No planning employees to show
from this schedule");
                break;
            }

            System.out.println("Selected planning employees by ID:
");
            for (String s : selectedPlanningEmployeeIDs){
                System.out.println(s);
            }
            System.out.println();
        }
        case "4" -> {
            if (selectedPlanningEmployeeIDs.isEmpty()){

                System.out.println("Have at least one assigned
employee");
                break;
            }


schedule.setPlanningTeamAssign(selectedPlanningEmployeeIDs);
            System.out.println("Planning assignment created");
            endProgram = true;
            returnValue = true;
        }
        case "5" -> {

            for (String s : selectedPlanningEmployeeIDs){

                PRPlanningEmployee resetEmployee =
PRPlanningEmployeeRepository.get(s);
                resetEmployee.setCurrentAssignment("free");

availablePRPlanningEmployeePriorityQueue.add(resetEmployee);

                // updating employee assignment to files
                editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
            }
            return returnValue;
        }
        case "0" -> {

            if (selectedPlanningEmployeeIDs.isEmpty()){

                endProgram = true;
            }
            else {

                System.out.println("Remove all assigned planning
employees or cancel before exiting");
            }
        }
        default -> System.out.println("Invalid choice. Try again");
      }
    }
    return returnValue;
  }

  /**
   * To add review employee to a schedule
   * @param schedule given schedule
   * @return true if successful, false otherwise
   */
  private boolean
addReviewEmployeeToSchedule(ActivitySchedules schedule){

    if (!retrieveAllPREmployees()){
      return false;
    }

    List<String> selectedReviewEmployeeIDs = new ArrayList<>();
    if (schedule.getReviewTeamAssign() != null &&
!schedule.getReviewTeamAssign().isEmpty()){
      selectedReviewEmployeeIDs =
schedule.getReviewTeamAssign();
    }

    Scanner scan = new Scanner(System.in);

    boolean endProgram = false;
    boolean returnValue = false;

    while (!endProgram){

      System.out.println("Review Employee Assignment Manager:
");
      System.out.println("1. Add Review Employee by ID");
      System.out.println("2. Remove Review Employee by ID");
      System.out.println("3. Show Scheduled Review Employees");
```

```
        System.out.println("4. Complete Schedule");
        System.out.println("5. Cancel Schedule");
        System.out.println("0. Exit");

        String choice = scan.nextLine();

        switch (choice){

            case "1" -> {

                System.out.println("Enter review employee ID: ");
                String employeeID = scan.nextLine();

                if
(!PRReviewEmployeeRepository.containsKey(employeeID)){

                    System.out.println("Review employee with ID: " +
employeeID + " does not exists");
                    break;
                }

                PRReviewEmployee employee =
PRReviewEmployeeRepository.get(employeeID);

                if (!employee.getCurrentAssignment().equals("free")){

                    System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                            "Overwrite? (y/n)\n" +
                            "!!!WARNING!!! This process is irreversible!");

                    String yesNo = scan.nextLine();

                    if (yesNo.equals("y")){


employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

                        if
(!selectedReviewEmployeeIDs.contains(employee.getEmployeeID()))
{


selectedReviewEmployeeIDs.add(employee.getEmployeeID());
                        }

                        // updating employee assignments to file
                        editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                        editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                        editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                        editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                    }
                }
                else {


availablePRReviewEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedReviewEmployeeIDs.add(employee.getEmployeeID());

                    // updating employee assignments to file
                    editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                    editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                    editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                }
            }
            case "2" -> {

                if (selectedReviewEmployeeIDs.isEmpty()){

                    System.out.println("No review employees to remove
from this schedule");
                    break;
                }

                System.out.println("Enter review employee ID to remove:
");
                String removeID = scan.nextLine();

                if (!selectedReviewEmployeeIDs.contains(removeID)){

                    System.out.println("Review employee ID: " +
removeID + " not assigned to this schedule");
                    break;
                }

                selectedReviewEmployeeIDs.remove(removeID);
                PRReviewEmployee employee =
PRReviewEmployeeRepository.get(removeID);
                employee.setCurrentAssignment("free");

availablePRReviewEmployeePriorityQueue.add(employee);

                // updating employee assignments to file
                editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
            }
            case "3" -> {

                if (selectedReviewEmployeeIDs.isEmpty()){

                    System.out.println("No review employees to show
from this schedule");
                    break;
                }

                System.out.println("Selected review employees by ID: ");
                for (String s : selectedReviewEmployeeIDs){
                    System.out.println(s);
                }
                System.out.println();
            }
            case "4" -> {
                if (selectedReviewEmployeeIDs.isEmpty()){

                    System.out.println("Have at least one assigned
employee");
                    break;
                }


schedule.setReviewTeamAssign(selectedReviewEmployeeIDs);
                System.out.println("Planning assignment created");
                endProgram = true;
```

```
                returnValue = true;
            }
            case "5" -> {

                for (String s : selectedReviewEmployeeIDs){

                    PRReviewEmployee resetEmployee =
PRReviewEmployeeRepository.get(s);
                    resetEmployee.setCurrentAssignment("free");

availablePRReviewEmployeePriorityQueue.add(resetEmployee);

                    // updating employee assignment to files
                    editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                    editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                    editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                }
                return returnValue;
            }
            case "0" -> {

                if (selectedReviewEmployeeIDs.isEmpty()){

                    endProgram = true;
                }
                else {

                    System.out.println("Remove all assigned review
employees or cancel before exiting");
                }
            }
            default -> System.out.println("Invalid choice. Try again");
        }
    }
    return returnValue;
  }

  /**
   * Retrieve all activity schedules
   * @return true if successful, false otherwise
   */
  private boolean retrieveAllActivities() {

    damageControlActivitySchedulesPriorityQueue.clear();
    planningActivitySchedulesPriorityQueue.clear();
    reviewingActivitySchedulesPriorityQueue.clear();
    revisingActivitySchedulesPriorityQueue.clear();
    readyActivitySchedulesPriorityQueue.clear();
    allActivitySchedulesPriorityQueue.clear();
    activitySchedulesRepository.clear();

    File[] scheduleTextFiles =
retrieveActivityFiles(activityScheduleDir);

    if (scheduleTextFiles == null){
      return false;
    }

    for (File f : scheduleTextFiles){

      editor.processTextFile(activityScheduleDir + f.getName());
      String[] schedules = editor.getArrayNames();

      for (String s : schedules){

        ActivitySchedules schedule = new ActivitySchedules(s,
            editor.retrieveValue(s, "activityName"),
            editor.retrieveValue(s, "activityType"),
            editor.retrieveValue(s, "objective"),
            editor.retrieveValue(s, "targetAudience"),
            editor.retrieveValue(s, "description"),
            editor.retrieveValue(s, "tasks"),
            editor.retrieveValue(s, "department"),
            editor.retrieveValue(s, "location"),
            editor.retrieveValue(s, "duration"),
            editor.retrieveValue(s, "deadline"),
            editor.retrieveValue(s, "status"),
            editor.retrieveValue(s, "dateScheduled"),
            f.getName());

        List<String> planningEmployees = new ArrayList<>();
        int num = 0;
        String planningEmployeeID = "employeePlanning" + num;
        while ((editor.retrieveValue(s, planningEmployeeID)) !=
null){

          planningEmployees.add(editor.retrieveValue(s,
planningEmployeeID));
          num++;
          planningEmployeeID = "employeePlanning" + num;
        }
        schedule.setPlanningTeamAssign(planningEmployees);

        List<String> reviewEmployees = new ArrayList<>();
        int num1 = 0;
        String reviewEmployeeID = "employeeReview" + num1;
        while ((editor.retrieveValue(s, reviewEmployeeID)) != null){

          reviewEmployees.add(editor.retrieveValue(s,
reviewEmployeeID));
          num1++;
          reviewEmployeeID = "employeeReview" + num1;
        }
        schedule.setReviewTeamAssign(reviewEmployees);

        if (schedule.getScheduleID() == null ||
schedule.getActivityName() == null ||
            schedule.getActivityType() == null ||
schedule.getObjective() == null ||
            schedule.getTargetAudience() == null ||
schedule.getDescription() == null ||
            schedule.getTasks() == null || schedule.getDepartment()
== null ||
            schedule.getLocation() == null || schedule.getDuration()
== null ||
            schedule.getDeadline() == null || schedule.getStatus() ==
null ||
            schedule.getDateScheduled() == null){

          System.out.println("Invalid schedule data detected");
          return false;
        }

        String status = schedule.getStatus();
        switch (status){
          case "Damage Control" ->
damageControlActivitySchedulesPriorityQueue.add(schedule);
          case "Planning" ->
planningActivitySchedulesPriorityQueue.add(schedule);
          case "Reviewing" ->
reviewingActivitySchedulesPriorityQueue.add(schedule);
          case "Revising" ->
revisingActivitySchedulesPriorityQueue.add(schedule);
          case "Ready" ->
readyActivitySchedulesPriorityQueue.add(schedule);
        }
```

```
            allActivitySchedulesPriorityQueue.add(schedule);
            activitySchedulesRepository.put(s, schedule);
        }
      }
      return true;
   }

   /**
    * Retrieve schedule file list
    * @param fileDirectory given directory
    * @return schedule file list
    */
   private File[] retrieveActivityFiles(String fileDirectory){

      File directory = new File(fileDirectory);
      File[] textFiles = null;

      if (directory.exists() && directory.isDirectory()){

         // grab list of text files
         FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
         textFiles = directory.listFiles(textFileFilter);
      }
      else {

         System.out.println("Repository not found for directory: " +
fileDirectory);
         return null;
      }

      if (textFiles != null) {

         if (textFiles.length == 0){
            System.out.println("No activity schedules found");
            return null;
         }
      }
      else {
         System.out.println("No activity schedules found");
         return null;
      }
      return textFiles;
   }

   /**
    * Retrieve all PR Planning and Review Employees
    * @return true if successful, false otherwise
    */
   private boolean retrieveAllPREmployees(){

      // clearing repositories and queues
      availablePRPlanningEmployeePriorityQueue.clear();
      availablePRReviewEmployeePriorityQueue.clear();
      allPRPlanningEmployeePriorityQueue.clear();
      allPRReviewEmployeePriorityQueue.clear();
      PRPlanningEmployeeRepository.clear();
      PRReviewEmployeeRepository.clear();

      // saving Planning employees
      File[] planningTextFiles =
retrieveEmployeeFiles(PRPlanningEmployeesDir);

      if (planningTextFiles == null){
         return false;
      }

      for (File f : planningTextFiles){

         editor.processTextFile(PRPlanningEmployeesDir +
f.getName());
         String[] employees = editor.getArrayNames();

         for (String s : employees){

            PRPlanningEmployee employee = new
PRPlanningEmployee(s,
                  editor.retrieveValue(s,"name"),
                  editor.retrieveValue(s, "division"),
                  editor.retrieveValue(s, "position"),
                  editor.retrieveValue(s, "department"),
                  editor.retrieveValue(s, "rating"),
                  editor.retrieveValue(s, "yearsOfExperience"),
                  editor.retrieveValue(s, "previousAssignment"),
                  editor.retrieveValue(s, "currentAssignment"));

            if (employee.getEmployeeID() == null ||
employee.getName() == null ||
                  employee.getDivision() == null ||
employee.getPosition() == null ||
                  employee.getDepartment() == null ||
employee.getRating() == null ||
                  employee.getYearsOfExperience() == null ||
employee.getPreviousAssignment() == null ||
                  employee.getCurrentAssignment() == null) {

               System.out.println("Invalid employee data detected");
               return false;
            }

            if (employee.getCurrentAssignment().equals("free")){

availablePRPlanningEmployeePriorityQueue.add(employee);
            }
            allPRPlanningEmployeePriorityQueue.add(employee);
            PRPlanningEmployeeRepository.put(s, employee);
         }
      }

      // saving Review employees
      File[] reviewTextFiles =
retrieveEmployeeFiles(PRReviewEmployeesDir);

      if (reviewTextFiles == null){
         return false;
      }

      for (File f : reviewTextFiles){

         editor.processTextFile(PRReviewEmployeesDir +
f.getName());
         String[] employees = editor.getArrayNames();

         for (String s : employees){

            PRReviewEmployee employee = new
PRReviewEmployee(s,
                  editor.retrieveValue(s,"name"),
                  editor.retrieveValue(s, "division"),
                  editor.retrieveValue(s, "position"),
                  editor.retrieveValue(s, "department"),
                  editor.retrieveValue(s, "rating"),
                  editor.retrieveValue(s, "yearsOfExperience"),
                  editor.retrieveValue(s, "previousAssignment"),
                  editor.retrieveValue(s, "currentAssignment"));

            if (employee.getEmployeeID() == null ||
employee.getName() == null ||
```

```
                    employee.getDivision() == null ||
employee.getPosition() == null ||
                    employee.getDepartment() == null ||
employee.getRating() == null ||
                    employee.getYearsOfExperience() == null ||
employee.getPreviousAssignment() == null ||
                    employee.getCurrentAssignment() == null) {

                System.out.println("Invalid employee data detected");
                return false;
            }

            if (employee.getCurrentAssignment().equals("free")){

availablePRReviewEmployeePriorityQueue.add(employee);
            }
            allPRReviewEmployeePriorityQueue.add(employee);
            PRReviewEmployeeRepository.put(s, employee);
        }
    }
    return true;
  }

  /**
   * Retrieve employee file list
   * @param fileDirectory given directory
   * @return employee file list
   */
  private File[] retrieveEmployeeFiles(String fileDirectory){

    File directory = new File(fileDirectory);
    File[] textFiles = null;

    if (directory.exists() && directory.isDirectory()){

      // grab list of text files
      FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
      textFiles = directory.listFiles(textFileFilter);
    }
    else {

      System.out.println("Repository not found for directory: " +
fileDirectory);
      return null;
    }

    if (textFiles != null) {

      if (textFiles.length == 0){
        System.out.println("No employees found");
        return null;
      }
    }
    else {
      System.out.println("No employees found");
      return null;
    }
    return textFiles;
  }

  /**
   * Converts ActivitySchedules object into nested HashMap
   * @param schedule specified ActivitySchedule
   * @return HashMap of ActivitySchedule
   */
  private Map<String, Object>
activityScheduleToHashmap(ActivitySchedules schedule){

    Map<String, Object> innerMap = new LinkedHashMap<>();
    innerMap.put("activityName", schedule.getActivityName());
    innerMap.put("activityType", schedule.getActivityType());
    innerMap.put("objective", schedule.getObjective());
    innerMap.put("targetAudience", schedule.getTargetAudience());
    innerMap.put("description", schedule.getDescription());
    innerMap.put("tasks", schedule.getTasks());
    innerMap.put("department", schedule.getDepartment());
    innerMap.put("location", schedule.getLocation());
    innerMap.put("duration", schedule.getDuration());
    innerMap.put("deadline", schedule.getDeadline());
    innerMap.put("status", schedule.getStatus());
    innerMap.put("dateScheduled", schedule.getDateScheduled());

    int num = 0;
    for (String s : schedule.getPlanningTeamAssign()){
      String employee = "employeePlanning" + num;
      innerMap.put(employee, s);
      num++;
    }

    int num1 = 0;
    for (String s : schedule.getReviewTeamAssign()){
      String employee = "employeeReview" + num1;
      innerMap.put(employee, s);
      num1++;
    }

    Map<String, Object> outerMap = new LinkedHashMap<>();
    outerMap.put(schedule.getScheduleID(), innerMap);

    return outerMap;
  }

  /**
   * Creates printable content schedule
   * @param schedule specific content schedule
   */
  private void printActivitySchedule(ActivitySchedules schedule){

    BufferedWriter writer = null;

    try {
      writer = new BufferedWriter(new
FileWriter(printedActivitySchedulesDir + schedule.getScheduleID() +
".txt"));

writer.write("==========================================
============================\n");
      writer.write("ACTIVITY SCHEDULE\n");
      writer.write("Schedule ID: " + schedule.getScheduleID() +
"\n");
      writer.write("Status: " + schedule.getStatus() + " (No status |
Planning | Reviewing | Revising | Ready | Damage Control)\n");
      writer.write("Department: " + schedule.getDepartment() +
"\n");

writer.write("----------------------------------------------------------------------\n");
      writer.write("Activity Name: " + schedule.getActivityName() +
"\n");
      writer.write("Activity Type: " + schedule.getActivityType() +
"\n");
      writer.write("Objective: " + schedule.getObjective() + "\n");
      writer.write("Target Audience: " +
schedule.getTargetAudience() + "\n");
      writer.write("Description: " + schedule.getDescription() + "\n");
      writer.write("Tasks: " + schedule.getTasks() + "\n");

writer.write("----------------------------------------------------------------------\n");
      writer.write("Location: " + schedule.getLocation() + "\n");
      writer.write("Duration: " + schedule.getDuration() + "\n");
```

```
            writer.write("Deadline: " + schedule.getDeadline() + "\n");
            writer.write("Date Scheduled: " +
schedule.getDateScheduled() + "\n");

writer.write("--------------------------------------------------------------------\n");
            writer.write("Assigned Planning Employees:\n");
            for (String s : schedule.getPlanningTeamAssign()){
                writer.write(s + "\n");
            }
            writer.write("Assigned Review Employees:\n");
            for (String s : schedule.getReviewTeamAssign()){
                writer.write(s + "\n");
            }

writer.write("============================================
==========================\n");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Pretty print planning employee
     * @param employee specific planning employee
     */
    private void planningEmployeeToText(PRPlanningEmployee
employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("----------------------------------------------------");
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<\n");
    }

    /**
     * Pretty print review employee
     * @param employee specific review employee
     */
    private void reviewEmployeeToText(PRReviewEmployee
employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("----------------------------------------------------");
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<\n");
    }

    /**
     * Pretty print activity schedules
     * @param schedule specific activity schedule
     */
    private void activityScheduleToText(ActivitySchedules schedule){


System.out.println("====================================
==================");
        System.out.println("Schedule ID: " + schedule.getScheduleID());
        System.out.println("Status: " + schedule.getStatus() + " (No
status | Planning | Reviewing | Revising | Ready | Damage Control)");
        System.out.println("Department: " + schedule.getDepartment());
        System.out.println("----------------------------------------------------");
        System.out.println("Activity Name: " +
schedule.getActivityType());
        System.out.println("Activity Type: " + schedule.getActivityType());
        System.out.println("Objective: " + schedule.getObjective());
        System.out.println("Target Audience: " +
schedule.getTargetAudience());
        System.out.println("Description: " + schedule.getDescription());
        System.out.println("Tasks: " + schedule.getTasks());
        System.out.println("----------------------------------------------------");
        System.out.println("Location: " + schedule.getLocation());
        System.out.println("Duration: " + schedule.getDuration());
        System.out.println("Deadline: " + schedule.getDeadline());
        System.out.println("Date Scheduled: " +
schedule.getDateScheduled());
        System.out.println("----------------------------------------------------");
        System.out.println("Assigned Planning Employees:");
        for (String s : schedule.getPlanningTeamAssign()){
            System.out.println(s);
        }
        System.out.println("----------------------------------------------------");
        System.out.println("Assigned Review Employees:");
        for (String s : schedule.getReviewTeamAssign()){
            System.out.println(s);
        }

System.out.println("====================================
==================\n");
    }

    /**
     * Generates IDs for activity schedules
     * @return generated activity schedule ID
     */
    private String IDGenerator(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMddyyyyHHmmss");
```

```java
        return timeNow.format(formatter);
    }

    /**
     * Returns current time in MM-dd-yyyy-HH-mm-ss format when
called
     * @return date in MM-dd-yyyy-HH-mm-ss format
     */
    private String dateIssuer(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy-HH-mm-ss");

        return timeNow.format(formatter);
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

import src.App;

import java.util.Scanner;

public class ActivitySchedulerController implements
src.PublicRelations.src.interfaces.ActivitySchedulerController {

    private ActivitySchedulerController controller;

    private ActivityScheduler scheduler = null;

    public ActivitySchedulerController (ActivitySchedulerController
controller){
        this.controller = controller;
    }

    public ActivitySchedulerController(){};

    /**
     * Run the security program
     * @throws Exception
     */
    public void run() throws Exception {

        System.out.println("Welcome to the Social Activity Scheduler
System");

        boolean exit = false;

        while (!exit){

            Scanner scan = new Scanner(System.in);


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>");
            System.out.println("Select An Action: ");
            System.out.println("1. Add New Public Relations Employee");
            System.out.println("2. Delete Public Relations Employee by
ID");
            System.out.println();
            System.out.println("3. Show Available Planning Employees");
            System.out.println("4. Show All Planning Employees");
            System.out.println("5. Show Available Review Employees");
            System.out.println("6. Show All Review Employees");
            System.out.println();
            System.out.println("7. Create Activity Schedule");
            System.out.println("8. Delete Activity Schedule");
            System.out.println("9. Edit Existing Activity Schedule");
            System.out.println();
            System.out.println("10. Show All Activity Schedules");
            System.out.println("11. Show Activity Schedules In Damage
Control");
            System.out.println("12. Show Activity Schedules In Planning");
            System.out.println("13. Show Activity Schedules In
Reviewing");
            System.out.println("14. Show Activity Schedules In Revising");
            System.out.println("15. Show Activity Schedules In Ready");
            System.out.println("16. Show Activity Schedule By ID");
            System.out.println();
            System.out.println("0. Exit program");

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<");

            String choice = scan.nextLine();

            switch (choice){

                case "1" -> {
                    addPREmployee();
                    System.out.println();
                }
                case "2" -> {
                    System.out.println("Enter Public Relations Employee ID
to delete: ");
                    String deleteID = scan.nextLine();
                    deletePREmployee(deleteID);
                    System.out.println();
                }
                case "3" -> {
                    showFreePlanningEmployees();
                    System.out.println();
                }
                case "4" -> {
                    showAllPlanningEmployees();
                    System.out.println();
                }
                case "5" -> {
                    showFreeReviewEmployees();
                    System.out.println();
                }
                case "6" -> {
                    showAllReviewEmployees();
                    System.out.println();
                }
                case "7" -> {
                    createActivity();
                    System.out.println();
                }
                case "8" -> {
                    System.out.println("Enter activity schedule ID to delete:
");
                    String deleteScheduleID = scan.nextLine();
                    deleteActivity(deleteScheduleID);
                    System.out.println();
                }
                case "9" -> {
                    System.out.println("Enter activity schedule ID to modify:
");
                    String activityID = scan.nextLine();
                    editActivity(activityID);
                    System.out.println();
                }
                case "10" -> {
                    showAllActivities();
```

```
                System.out.println();
            }
            case "11" -> {
                showDamageControlActivities();
                System.out.println();
            }
            case "12" -> {
                showPlanningActivities();
                System.out.println();
            }
            case "13" -> {
                showReviewingActivities();
                System.out.println();
            }
            case "14" -> {
                showRevisingContents();
                System.out.println();
            }
            case "15" -> {
                showReadyContents();
                System.out.println();
            }
            case "16" -> {
                System.out.println("Enter activity schedule ID to view: ");
                String activityViewID = scan.nextLine();
                showActivityByID(activityViewID);
                System.out.println();
            }
            case "0" -> {
                System.out.println("Closing Program");
                App.prompt();
            }
            default -> System.out.println("Invalid choice. Try again");
        }
      }
    }

    @Override
    public boolean addPREmployee() {
        return this.getSchedulerInstance().addPREmployee();
    }

    @Override
    public boolean deletePREmployee(String employeeID) {
        return
this.getSchedulerInstance().deletePREmployee(employeeID);
    }

    @Override
    public boolean showFreePlanningEmployees() {
        return
this.getSchedulerInstance().showFreePlanningEmployees();
    }

    @Override
    public boolean showAllPlanningEmployees() {
        return this.getSchedulerInstance().showAllPlanningEmployees();
    }

    @Override
    public boolean showFreeReviewEmployees() {
        return
this.getSchedulerInstance().showFreeReviewEmployees();
    }

    @Override
    public boolean showAllReviewEmployees() {
        return this.getSchedulerInstance().showAllReviewEmployees();
    }

    @Override
    public boolean createActivity() {
        return this.getSchedulerInstance().createActivity();
    }

    @Override
    public boolean deleteActivity(String activityID) {
        return this.getSchedulerInstance().deleteActivity(activityID);
    }

    @Override
    public boolean editActivity(String activityID) {
        return this.getSchedulerInstance().editActivity(activityID);
    }

    @Override
    public boolean showAllActivities() {
        return this.getSchedulerInstance().showAllActivities();
    }

    @Override
    public boolean showDamageControlActivities() {
        return
this.getSchedulerInstance().showDamageControlActivities();
    }

    @Override
    public boolean showPlanningActivities() {
        return this.getSchedulerInstance().showPlanningActivities();
    }

    @Override
    public boolean showReviewingActivities() {
        return this.getSchedulerInstance().showReviewingActivities();
    }

    @Override
    public boolean showRevisingContents() {
        return this.getSchedulerInstance().showRevisingContents();
    }

    @Override
    public boolean showReadyContents() {
        return this.getSchedulerInstance().showReadyContents();
    }

    @Override
    public boolean showActivityByID(String activityID) {
        return this.getSchedulerInstance().showActivityByID(activityID);
    }

    /**
     * To define one instance of the class ActivityScheduler to
     * call ActivityScheduler methods
     * @return ActivityScheduler instance
     */
    private ActivityScheduler getSchedulerInstance(){

        if (scheduler == null){
            scheduler = new ActivityScheduler();
        }
        return scheduler;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;
```

```
import java.util.List;

public class ActivitySchedules {

    private String scheduleID, activityName, activityType, objective,
targetAudience, description,
                tasks, department, location, duration, deadline, status,
dateScheduled, fileName;
    private List<String> planningTeamAssign, reviewTeamAssign;

    public ActivitySchedules(){}

    public ActivitySchedules(String scheduleID, String activityName,
String activityType,
                        String objective, String targetAudience, String
description,
                        String tasks, String department, String location,
                        String duration, String deadline, String status,
                        String dateIssued, String fileName){

        this.scheduleID = scheduleID;
        this.activityName = activityName;
        this.activityType = activityType;
        this.objective = objective;
        this.targetAudience = targetAudience;
        this.description = description;
        this.tasks = tasks;
        this.department = department;
        this.location = location;
        this.duration = duration;
        this.deadline = deadline;
        this.status = status;
        this.dateScheduled = dateIssued;
        this.fileName = fileName;
    }

    public String getScheduleID() {
        return scheduleID;
    }

    public String getActivityName() {
        return activityName;
    }

    public String getActivityType() {
        return activityType;
    }

    public String getObjective() {
        return objective;
    }

    public String getTargetAudience() {
        return targetAudience;
    }

    public String getDescription() {
        return description;
    }

    public String getTasks() {
        return tasks;
    }

    public String getDepartment() {
        return department;
    }

    public String getLocation() {
        return location;
    }

    public String getDuration() {
        return duration;
    }

    public String getDeadline() {
        return deadline;
    }

    public String getStatus() {
        return status;
    }

    public String getDateScheduled() {
        return dateScheduled;
    }

    public String getFileName() {
        return fileName;
    }

    public List<String> getPlanningTeamAssign() {
        return planningTeamAssign;
    }

    public List<String> getReviewTeamAssign() {
        return reviewTeamAssign;
    }

    public void setScheduleID(String scheduleID) {
        this.scheduleID = scheduleID;
    }

    public void setActivityName(String activityName) {
        this.activityName = activityName;
    }

    public void setActivityType(String activityType) {
        this.activityType = activityType;
    }

    public void setObjective(String objective) {
        this.objective = objective;
    }

    public void setTargetAudience(String targetAudience) {
        this.targetAudience = targetAudience;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setTasks(String tasks) {
        this.tasks = tasks;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setLocation(String location) {
        this.location = location;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }
```

```
    public void setDeadline(String deadline) {
        this.deadline = deadline;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public void setDateScheduled(String dateScheduled) {
        this.dateScheduled = dateScheduled;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public void setPlanningTeamAssign(List<String>
planningTeamAssign) {
        this.planningTeamAssign = planningTeamAssign;
    }

    public void setReviewTeamAssign(List<String> reviewTeamAssign)
{
        this.reviewTeamAssign = reviewTeamAssign;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

public class ContentRequests {

    String requestID, theme, contentCategory, platform, duration,
deadline,
            description, tasks, department, priority, specialReqs,
dateIssued,
            resolved, fileName;

    public ContentRequests(){}

    public ContentRequests(String requestID, String theme, String
contentCategory,
                           String platform, String duration, String deadline,
                           String description, String tasks, String department,
                           String priority, String specialReqs, String dateIssued,
                           String resolved, String fileName){

        this.requestID = requestID;
        this.theme = theme;
        this.contentCategory = contentCategory;
        this.platform = platform;
        this.duration = duration;
        this.deadline = deadline;
        this.description = description;
        this.tasks = tasks;
        this.department = department;
        this.priority = priority;
        this.specialReqs = specialReqs;
        this.dateIssued = dateIssued;
        this.resolved = resolved;
        this.fileName = fileName;
    }

    public String getRequestID() {
        return requestID;
    }

public String getTheme() {
    return theme;
}

public String getContentCategory() {
    return contentCategory;
}

public String getPlatform() {
    return platform;
}

public String getDuration() {
    return duration;
}

public String getDeadline() {
    return deadline;
}

public String getDescription() {
    return description;
}

public String getTasks() {
    return tasks;
}

public String getDepartment() {
    return department;
}

public String getPriority() {
    return priority;
}

public String getSpecialReqs() {
    return specialReqs;
}

public String getDateIssued() {
    return dateIssued;
}

public String getResolved() {
    return resolved;
}

public String getFileName() {
    return fileName;
}

public void setRequestID(String requestID) {
    this.requestID = requestID;
}

public void setTheme(String theme) {
    this.theme = theme;
}

public void setContentCategory(String contentCategory) {
    this.contentCategory = contentCategory;
}

public void setPlatform(String platform) {
    this.platform = platform;
}

public void setDuration(String duration) {
```

```
        this.duration = duration;
    }

    public void setDeadline(String deadline) {
        this.deadline = deadline;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setTasks(String tasks) {
        this.tasks = tasks;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setPriority(String priority) {
        this.priority = priority;
    }

    public void setSpecialReqs(String specialReqs) {
        this.specialReqs = specialReqs;
    }

    public void setDateIssued(String dateIssued) {
        this.dateIssued = dateIssued;
    }

    public void setResolved(String resolved) {
        this.resolved = resolved;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

import src.TextEditor.PoorTextEditor;

import java.io.*;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class ContentScheduler implements
src.PublicRelations.src.interfaces.ContentScheduler {

    private final String departmentRequestDir =
"src/PublicRelations/repository/departmentRequests/";
    private final String PRPlanningEmployeesDir =
"src/PublicRelations/repository/planningEmployees/";
    private final String PRReviewEmployeesDir =
"src/PublicRelations/repository/reviewEmployees/";
    private final String contentSchedulesDir =
"src/PublicRelations/repository/contentSchedules/";
    private final String printedContentSchedulesDir =
"src/PublicRelations/repository/printedContentSchedules/";

    PoorTextEditor editor = new PoorTextEditor();

    // sorts content requests by deadline first, then by priority
    PriorityQueue<ContentRequests>
pendingContentRequestsPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentRequests,
LocalDate>comparing(request -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(request.getDeadline(), formatter);
            }).thenComparingInt(request ->
-Integer.parseInt(request.getPriority()))
    );
    PriorityQueue<ContentRequests>
allContentRequestsPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentRequests,
LocalDate>comparing(request -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(request.getDeadline(), formatter);
            }).thenComparingInt(request ->
-Integer.parseInt(request.getPriority()))
    );
    /**
     * Repository to store all content requests
     */
    Map<String, ContentRequests> contentRequestsRepository = new
LinkedHashMap<>();

    // sorts content schedules by deadline first, then by priority
    PriorityQueue<ContentSchedules>
planningContentSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentSchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparingInt(schedule ->
-Integer.parseInt(schedule.getPriority()))
    );

    PriorityQueue<ContentSchedules>
reviewingContentSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentSchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparingInt(schedule ->
-Integer.parseInt(schedule.getPriority()))
    );

    PriorityQueue<ContentSchedules>
revisingContentSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentSchedules,
LocalDate>comparing(schedule -> {

                DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
                return LocalDate.parse(schedule.getDeadline(), formatter);
            }).thenComparingInt(schedule ->
-Integer.parseInt(schedule.getPriority()))
    );

    PriorityQueue<ContentSchedules>
readyContentSchedulesPriorityQueue = new PriorityQueue<>(
            Comparator.<ContentSchedules,
LocalDate>comparing(schedule -> {
```

```
            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
            return LocalDate.parse(schedule.getDeadline(), formatter);
        }).thenComparingInt(schedule ->
-Integer.parseInt(schedule.getPriority()))
    );

    PriorityQueue<ContentSchedules>
allContentSchedulesPriorityQueue = new PriorityQueue<>(
        Comparator.<ContentSchedules,
LocalDate>comparing(schedule -> {

            DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy");
            return LocalDate.parse(schedule.getDeadline(), formatter);
        }).thenComparingInt(schedule ->
-Integer.parseInt(schedule.getPriority()))
    );
    /**
     * Repository to store all content schedules
     */
    Map<String, ContentSchedules> contentSchedulesRepository =
new LinkedHashMap<>();

    // sorts employees by division, then position, then rating in
descending order
    PriorityQueue<PRPlanningEmployee>
availablePRPlanningEmployeePriorityQueue = new PriorityQueue<>(
        Comparator.comparing(PRPlanningEmployee::getDivision)
                .thenComparing(PRPlanningEmployee::getPosition)
                .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRPlanningEmployee>
allPRPlanningEmployeePriorityQueue = new PriorityQueue<>(
        Comparator.comparing(PRPlanningEmployee::getDivision)
                .thenComparing(PRPlanningEmployee::getPosition)
                .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRReviewEmployee>
availablePRReviewEmployeePriorityQueue = new PriorityQueue<>(
        Comparator.comparing(PRReviewEmployee::getDivision)
                .thenComparing(PRReviewEmployee::getPosition)
                .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    PriorityQueue<PRReviewEmployee>
allPRReviewEmployeePriorityQueue = new PriorityQueue<>(
        Comparator.comparing(PRReviewEmployee::getDivision)
                .thenComparing(PRReviewEmployee::getPosition)
                .thenComparing((e1, e2) ->
e2.getRating().compareTo(e1.getRating()))
    );
    /**
     * Repository to store all PR employees
     */
    Map<String, PRPlanningEmployee>
PRPlanningEmployeeRepository = new LinkedHashMap<>();
    Map<String, PRReviewEmployee> PRReviewEmployeeRepository
= new LinkedHashMap<>();

    @Override
    public boolean addPREmployee() {
        return false;
    }

    @Override
    public boolean deletePREmployee(String employeeID) {
        return false;
    }

    @Override
    public boolean showFreePlanningEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRPlanningEmployee> tempQueue = new
PriorityQueue<>(availablePRPlanningEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            planningEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showAllPlanningEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRPlanningEmployee> tempQueue = new
PriorityQueue<>(allPRPlanningEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            planningEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showFreeReviewEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRReviewEmployee> tempQueue = new
PriorityQueue<>(availablePRReviewEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            reviewEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showAllReviewEmployees() {

        if (!retrieveAllPREmployees()){
            return false;
        }

        PriorityQueue<PRReviewEmployee> tempQueue = new
PriorityQueue<>(allPRReviewEmployeePriorityQueue);

        while (!tempQueue.isEmpty()){
            reviewEmployeeToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showPendingRequests() {
```

```
        if (!retrieveAllRequests()){
            return false;
        }

        PriorityQueue<ContentRequests> tempQueue = new
PriorityQueue<>(pendingContentRequestsPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentRequestToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showAllRequests() {

        if (!retrieveAllRequests()){
            return false;
        }

        PriorityQueue<ContentRequests> tempQueue = new
PriorityQueue<>(allContentRequestsPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentRequestToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean createContentSchedule(String requestID) {

        if (!retrieveAllRequests()){
            return false;
        }

        if (!contentRequestsRepository.containsKey(requestID)){

            System.out.println("Content request ID: " + requestID + " not
found");
            return false;
        }

        if
(Boolean.parseBoolean(contentRequestsRepository.get(requestID).g
etResolved())){

            System.out.println("Security request ID: " + requestID + " has
already been resolved. Edit it instead");
            return false;
        }

        ContentRequests request =
contentRequestsRepository.get(requestID);
        contentRequestToText(request);

        ContentSchedules schedule = new ContentSchedules(
                IDGenerator() + "-" + request.getRequestID(),
                request.getRequestID(),
                request.getTheme(),
                request.getContentCategory(),
                request.getPlatform(),
                request.getDuration(),
                request.getDeadline(),
                request.getDescription(),
                request.getTasks(),
                request.getDepartment(),
                request.getPriority(),
                "no status",
                dateIssuer(),
                null);

        // assigning planning employee to schedule
        if (!addPlanningEmployeeToSchedule(schedule)){

            System.out.println("Planning employee schedule for request
ID: " + requestID + " was cancelled");
            return false;
        }

        // assigning review employee to schedule
        if (!addReviewEmployeeToSchedule(schedule)){

            System.out.println("Review employee schedule for request ID:
" + requestID + " was cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(contentScheduleToHashmap(schedule));
        editor.writeToTextFile(contentSchedulesDir +
schedule.getScheduleID() + ".txt");

        // writing printed schedule to repository
        printContentSchedule(schedule);

        // removing specific request from pending request queue
        pendingContentRequestsPriorityQueue.remove(request);
        // removing specific request in all queue
        allContentRequestsPriorityQueue.remove(request);
        // setting resolve status of content request to true
        request.setResolved("true");
        // adding back changed specific request
        allContentRequestsPriorityQueue.add(request);

        // editing content request resolved status in files
        editor.processTextFile(departmentRequestDir +
request.getFileName());
        editor.setValue(request.getRequestID(), "resolved", "true");
        editor.writeToTextFile(departmentRequestDir +
request.getFileName());

        return true;
    }

    @Override
    public boolean deleteContentSchedule(String contentID) {

        if (!retrieveAllSchedules()) return false;

        if (!retrieveAllPREmployees()) return false;

        if (!retrieveAllRequests()) return false;

        if (!contentSchedulesRepository.containsKey(contentID)){

            System.out.println("Content schedule ID: " + contentID + " not
found");
            return false;
        }

        ContentSchedules schedule =
contentSchedulesRepository.get(contentID);

        if (schedule.getPlanningTeamAssign().isEmpty()||
schedule.getReviewTeamAssign().isEmpty()){

            System.out.println("Incomplete employee assignment for
content schedule ID: " + contentID);
            return false;
```

```
        }

        /**
         * Freeing planning employees current assignment
         */
        List<String> planningEmployees =
schedule.getPlanningTeamAssign();
        editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
        for (String s : planningEmployees){
            editor.setValue(s, "currentAssignment", "free");
        }
        editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");

        /**
         * Freeing review employees current assignment
         */
        List<String> reviewEmployees =
schedule.getReviewTeamAssign();
        editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
        for (String s : reviewEmployees){
            editor.setValue(s, "currentAssignment", "free");
        }
        editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");

        /**
         * Setting content request resolved status back to false
         */
        ContentRequests request =
contentRequestsRepository.get(schedule.getRequestID());
        editor.processTextFile(departmentRequestDir +
request.getFileName());
        editor.setValue(schedule.getRequestID(), "resolved", "false");
        editor.writeToTextFile(departmentRequestDir +
request.getFileName());

        /**
         * Deleting content schedule
         */
        File fileToDelete = new File(contentSchedulesDir,
schedule.getFileName());

        if (fileToDelete.exists()){

            if (fileToDelete.delete()){

                System.out.println("Content schedule of ID: " + contentID +
" successfully deleted");
            }
            else {

                System.out.println("Failed to delete content schedule ID: " +
contentID);
                return false;
            }
        }
        else {

            System.out.println("Content schedule with ID: " + contentID +
" not found");
            return false;
        }
        return true;
    }

    @Override
    public boolean editContentScheduleAssignment(String contentID) {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (!contentSchedulesRepository.containsKey(contentID)){

            System.out.println("Content schedule with ID: " + contentID +
" does not exists");
            return false;
        }

        ContentSchedules schedule =
contentSchedulesRepository.get(contentID);
        if (!addPlanningEmployeeToSchedule(schedule)){
            System.out.println("Planning employee update cancelled");
            return false;
        }
        if (!addReviewEmployeeToSchedule(schedule)){
            System.out.println("Review employee update cancelled");
            return false;
        }

        // writing schedule to repository
        editor.setRepository(contentScheduleToHashmap(schedule));
        editor.writeToTextFile(contentSchedulesDir +
schedule.getFileName());

        // writing printed schedule to repository
        printContentSchedule(schedule);

        System.out.println("Update successful for content schedule ID: "
+ contentID);
        return true;
    }

    @Override
    public boolean editContentScheduleStatus(String contentID) {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (!contentSchedulesRepository.containsKey(contentID)){

            System.out.println("Content schedule with ID: " + contentID +
" does not exists");
            return false;
        }

        ContentSchedules schedule =
contentSchedulesRepository.get(contentID);
        System.out.println("Edit status for content schedule ID: " +
contentID + "\n(Planning | Reviewing | Revising | Ready)");
        Scanner scan = new Scanner(System.in);
        schedule.setStatus(scan.nextLine());

        // writing schedule to repository
        editor.setRepository(contentScheduleToHashmap(schedule));
        editor.writeToTextFile(contentSchedulesDir +
schedule.getFileName());

        // writing printed schedule to repository
        printContentSchedule(schedule);

        System.out.println("Update successful for content schedule ID: "
+ contentID);
        return true;
    }
```

```
    @Override
    public boolean showAllContentSchedules() {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (allContentSchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ContentSchedules> tempQueue = new
PriorityQueue<>(allContentSchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showPlanningContentSchedules() {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (planningContentSchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ContentSchedules> tempQueue = new
PriorityQueue<>(planningContentSchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showReviewingContentSchedules() {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (reviewingContentSchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ContentSchedules> tempQueue = new
PriorityQueue<>(reviewingContentSchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showRevisingContentSchedules() {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (revisingContentSchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ContentSchedules> tempQueue = new
PriorityQueue<>(revisingContentSchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showReadyContentSchedules() {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (readyContentSchedulesPriorityQueue.isEmpty()){
            System.out.println("Nothing was found");
            return false;
        }

        PriorityQueue<ContentSchedules> tempQueue = new
PriorityQueue<>(readyContentSchedulesPriorityQueue);

        while (!tempQueue.isEmpty()){
            contentScheduleToText(tempQueue.poll());
        }
        return true;
    }

    @Override
    public boolean showContentScheduleByID(String contentID) {

        if (!retrieveAllSchedules()){
            return false;
        }

        if (!contentSchedulesRepository.containsKey(contentID)){

            System.out.println("Content schedule with ID: " + contentID +
" not found");
            return false;
        }


contentScheduleToText(contentSchedulesRepository.get(contentID));
        return true;
    }

    /**
     * To add planning employee to a schedule
     * @param schedule given schedule
     * @return true if successful, false otherwise
     */
    private boolean
addPlanningEmployeeToSchedule(ContentSchedules schedule){

        if (!retrieveAllPREmployees()){
            return false;
        }

        List<String> selectedPlanningEmployeeIDs = new ArrayList<>();
        if (schedule.getPlanningTeamAssign() != null &&
!schedule.getPlanningTeamAssign().isEmpty()){
```

```
        selectedPlanningEmployeeIDs =
schedule.getPlanningTeamAssign();
    }

    Scanner scan = new Scanner(System.in);

    boolean endProgram = false;
    boolean returnValue = false;

    while (!endProgram){

        System.out.println("Planning Employee Assignment Manager:
");
        System.out.println("1. Add Planning Employee by ID");
        System.out.println("2. Remove Planning Employee by ID");
        System.out.println("3. Show Scheduled Planning
Employees");
        System.out.println("4. Complete Schedule");
        System.out.println("5. Cancel Schedule");
        System.out.println("0. Exit");

        String choice = scan.nextLine();

        switch (choice){

            case "1" -> {

                System.out.println("Enter planning employee ID: ");
                String employeeID = scan.nextLine();

                if
(!PRPlanningEmployeeRepository.containsKey(employeeID)){

                    System.out.println("Planning employee with ID: " +
employeeID + " does not exists");
                    break;
                }

                PRPlanningEmployee employee =
PRPlanningEmployeeRepository.get(employeeID);

                if (!employee.getCurrentAssignment().equals("free")){

                    System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                            "Overwrite? (y/n)\n" +
                            "!!!WARNING!!! This process is irreversible!");

                    String yesNo = scan.nextLine();

                    if (yesNo.equals("y")){


employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

                        if
(!selectedPlanningEmployeeIDs.contains(employee.getEmployeeID()
)){


selectedPlanningEmployeeIDs.add(employee.getEmployeeID());
                        }

                        // updating employee assignments to file
                        editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                        editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                        editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                        editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                    }
                }
                else {


availablePRPlanningEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedPlanningEmployeeIDs.add(employee.getEmployeeID());

                    // updating employee assignments to file
                    editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                    editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                    editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                }
            }
            case "2" -> {

                if (selectedPlanningEmployeeIDs.isEmpty()){

                    System.out.println("No planning employees to remove
from this schedule");
                    break;
                }

                System.out.println("Enter planning employee ID to
remove: ");
                String removeID = scan.nextLine();

                if (!selectedPlanningEmployeeIDs.contains(removeID)){

                    System.out.println("Planning employee ID: " +
removeID + " not assigned to this schedule");
                    break;
                }

                selectedPlanningEmployeeIDs.remove(removeID);
                PRPlanningEmployee employee =
PRPlanningEmployeeRepository.get(removeID);
                employee.setCurrentAssignment("free");

availablePRPlanningEmployeePriorityQueue.add(employee);

                // updating employee assignments to file
                editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
            }
            case "3" -> {

                if (selectedPlanningEmployeeIDs.isEmpty()){

                    System.out.println("No planning employees to show
from this schedule");
                    break;
                }
```

```
                System.out.println("Selected planning employees by ID:
");
                for (String s : selectedPlanningEmployeeIDs){
                    System.out.println(s);
                }
                System.out.println();
            }
            case "4" -> {
                if (selectedPlanningEmployeeIDs.isEmpty()){

                    System.out.println("Have at least one assigned
employee");
                    break;
                }


schedule.setPlanningTeamAssign(selectedPlanningEmployeeIDs);
                System.out.println("Planning assignment created");
                endProgram = true;
                returnValue = true;
            }
            case "5" -> {

                for (String s : selectedPlanningEmployeeIDs){

                    PRPlanningEmployee resetEmployee =
PRPlanningEmployeeRepository.get(s);
                    resetEmployee.setCurrentAssignment("free");

availablePRPlanningEmployeePriorityQueue.add(resetEmployee);

                    // updating employee assignment to files
                    editor.processTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                    editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                    editor.writeToTextFile(PRPlanningEmployeesDir +
"PRPlanningEmployeeList.txt");
                }
                return returnValue;
            }
            case "0" -> {

                if (selectedPlanningEmployeeIDs.isEmpty()){

                    endProgram = true;
                }
                else {

                    System.out.println("Remove all assigned planning
employees or cancel before exiting");
                }
            }
            default -> System.out.println("Invalid choice. Try again");
        }
    }
    return returnValue;
}

/**
 * To add review employee to a schedule
 * @param schedule given schedule
 * @return true if successful, false otherwise
 */
private boolean
addReviewEmployeeToSchedule(ContentSchedules schedule){

    if (!retrieveAllPREmployees()){
        return false;
    }

    List<String> selectedReviewEmployeeIDs = new ArrayList<>();
    if (schedule.getReviewTeamAssign() != null &&
!schedule.getReviewTeamAssign().isEmpty()){
        selectedReviewEmployeeIDs =
schedule.getReviewTeamAssign();
    }

    Scanner scan = new Scanner(System.in);

    boolean endProgram = false;
    boolean returnValue = false;

    while (!endProgram){

        System.out.println("Review Employee Assignment Manager:
");
        System.out.println("1. Add Review Employee by ID");
        System.out.println("2. Remove Review Employee by ID");
        System.out.println("3. Show Scheduled Review Employees");
        System.out.println("4. Complete Schedule");
        System.out.println("5. Cancel Schedule");
        System.out.println("0. Exit");

        String choice = scan.nextLine();

        switch (choice){

            case "1" -> {

                System.out.println("Enter review employee ID: ");
                String employeeID = scan.nextLine();

                if
(!PRReviewEmployeeRepository.containsKey(employeeID)){

                    System.out.println("Review employee with ID: " +
employeeID + " does not exists");
                    break;
                }

                PRReviewEmployee employee =
PRReviewEmployeeRepository.get(employeeID);

                if (!employee.getCurrentAssignment().equals("free")){

                    System.out.println("This employee has already been
assigned to :" + employee.getCurrentAssignment() + "\n" +
                            "Overwrite? (y/n)\n" +
                            "!!!WARNING!!! This process is irreversible!");

                    String yesNo = scan.nextLine();

                    if (yesNo.equals("y")){


employee.setPreviousAssignment(employee.getCurrentAssignment()
);

employee.setCurrentAssignment(schedule.getScheduleID());

                        if
(!selectedReviewEmployeeIDs.contains(employee.getEmployeeID()))
{


selectedReviewEmployeeIDs.add(employee.getEmployeeID());
                        }

                        // updating employee assignments to file
```

```
                    editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                    editor.setValue(employee.getEmployeeID(),
"previousAssignment", employee.getPreviousAssignment());
                    editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                    editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                }
            }
            else {


availablePRReviewEmployeePriorityQueue.remove(employee);

employee.setCurrentAssignment(schedule.getScheduleID());

selectedReviewEmployeeIDs.add(employee.getEmployeeID());

                // updating employee assignments to file
                editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
                editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
            }
        }
        case "2" -> {

            if (selectedReviewEmployeeIDs.isEmpty()){

                System.out.println("No review employees to remove
from this schedule");
                break;
            }

            System.out.println("Enter review employee ID to remove:
");
            String removeID = scan.nextLine();

            if (!selectedReviewEmployeeIDs.contains(removeID)){

                System.out.println("Review employee ID: " +
removeID + " not assigned to this schedule");
                break;
            }

            selectedReviewEmployeeIDs.remove(removeID);
            PRReviewEmployee employee =
PRReviewEmployeeRepository.get(removeID);
            employee.setCurrentAssignment("free");

availablePRReviewEmployeePriorityQueue.add(employee);

            // updating employee assignments to file
            editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
            editor.setValue(employee.getEmployeeID(),
"currentAssignment", employee.getCurrentAssignment());
            editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
        }
        case "3" -> {

            if (selectedReviewEmployeeIDs.isEmpty()){

                System.out.println("No review employees to show
from this schedule");
                break;
            }

            System.out.println("Selected review employees by ID: ");
            for (String s : selectedReviewEmployeeIDs){
                System.out.println(s);
            }
            System.out.println();
        }
        case "4" -> {
            if (selectedReviewEmployeeIDs.isEmpty()){

                System.out.println("Have at least one assigned
employee");
                break;
            }


schedule.setReviewTeamAssign(selectedReviewEmployeeIDs);
            System.out.println("Planning assignment created");
            endProgram = true;
            returnValue = true;
        }
        case "5" -> {

            for (String s : selectedReviewEmployeeIDs){

                PRReviewEmployee resetEmployee =
PRReviewEmployeeRepository.get(s);
                resetEmployee.setCurrentAssignment("free");

availablePRReviewEmployeePriorityQueue.add(resetEmployee);

                // updating employee assignment to files
                editor.processTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
                editor.setValue(resetEmployee.getEmployeeID(),
"currentAssignment", resetEmployee.getCurrentAssignment());
                editor.writeToTextFile(PRReviewEmployeesDir +
"PRReviewEmployeeList.txt");
            }
            return returnValue;
        }
        case "0" -> {

            if (selectedReviewEmployeeIDs.isEmpty()){

                endProgram = true;
            }
            else {

                System.out.println("Remove all assigned review
employees or cancel before exiting");
            }
        }
        default -> System.out.println("Invalid choice. Try again");
      }
    }
    return returnValue;
  }

  /**
   * Retrieve all content schedules
   * @return true if successful, false otherwise
   */
  private boolean retrieveAllSchedules() {

    planningContentSchedulesPriorityQueue.clear();
    reviewingContentSchedulesPriorityQueue.clear();
    revisingContentSchedulesPriorityQueue.clear();
    readyContentSchedulesPriorityQueue.clear();
```

```
        allContentSchedulesPriorityQueue.clear();
        contentSchedulesRepository.clear();

        File[] scheduleTextFiles =
retrieveScheduleFiles(contentSchedulesDir);

        if (scheduleTextFiles == null){
            return false;
        }

        for (File f : scheduleTextFiles){

            editor.processTextFile(contentSchedulesDir + f.getName());
            String[] schedules = editor.getArrayNames();

            for (String s : schedules){

                ContentSchedules schedule = new ContentSchedules(s,
                        editor.retrieveValue(s, "requestID"),
                        editor.retrieveValue(s, "theme"),
                        editor.retrieveValue(s, "contentCategory"),
                        editor.retrieveValue(s, "platform"),
                        editor.retrieveValue(s, "duration"),
                        editor.retrieveValue(s, "deadline"),
                        editor.retrieveValue(s, "description"),
                        editor.retrieveValue(s, "tasks"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "priority"),
                        editor.retrieveValue(s, "status"),
                        editor.retrieveValue(s, "dateScheduled"),
                        f.getName());

                List<String> planningEmployees = new ArrayList<>();
                int num = 0;
                String planningEmployeeID = "employeePlanning" + num;
                while ((editor.retrieveValue(s, planningEmployeeID)) !=
null){

                    planningEmployees.add(editor.retrieveValue(s,
planningEmployeeID));
                    num++;
                    planningEmployeeID = "employeePlanning" + num;
                }
                schedule.setPlanningTeamAssign(planningEmployees);

                List<String> reviewEmployees = new ArrayList<>();
                int num1 = 0;
                String reviewEmployeeID = "employeeReview" + num1;
                while ((editor.retrieveValue(s, reviewEmployeeID)) != null){

                    reviewEmployees.add(editor.retrieveValue(s,
reviewEmployeeID));
                    num1++;
                    reviewEmployeeID = "employeeReview" + num1;
                }
                schedule.setReviewTeamAssign(reviewEmployees);

                if (schedule.getScheduleID() == null || schedule.getTheme()
== null ||
                        schedule.getContentCategory() == null ||
schedule.getPlatform() == null ||
                        schedule.getDuration() == null || schedule.getDeadline()
== null ||
                        schedule.getDescription() == null || schedule.getTasks()
== null ||
                        schedule.getDepartment() == null ||
schedule.getPriority() == null ||
                        schedule.getStatus() == null ||
schedule.getDateScheduled() == null ||
                        schedule.getRequestID() == null ||
schedule.getPlanningTeamAssign().isEmpty() ||
                        schedule.getReviewTeamAssign().isEmpty()){

                    System.out.println("Invalid schedule data detected");
                    return false;
                }

                String status = schedule.getStatus();
                switch (status){
                    case "Planning" ->
planningContentSchedulesPriorityQueue.add(schedule);
                    case "Reviewing" ->
reviewingContentSchedulesPriorityQueue.add(schedule);
                    case "Revising" ->
revisingContentSchedulesPriorityQueue.add(schedule);
                    case "Ready" ->
readyContentSchedulesPriorityQueue.add(schedule);
                }

                allContentSchedulesPriorityQueue.add(schedule);
                contentSchedulesRepository.put(s, schedule);
            }
        }
        return true;
    }

    /**
     * Retrieve schedule file list
     * @param fileDirectory given directory
     * @return schedule file list
     */
    private File[] retrieveScheduleFiles(String fileDirectory){

        File directory = new File(fileDirectory);
        File[] textFiles = null;

        if (directory.exists() && directory.isDirectory()){

            // grab list of text files
            FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
            textFiles = directory.listFiles(textFileFilter);
        }
        else {

            System.out.println("Repository not found for directory: " +
fileDirectory);
            return null;
        }

        if (textFiles != null) {

            if (textFiles.length == 0){
                System.out.println("No content schedules found");
                return null;
            }
        }
        else {
            System.out.println("No content schedules found");
            return null;
        }
        return textFiles;
    }

    /**
     * Retrieve all content requests
     * @return true if successful, false otherwise
     */
    private boolean retrieveAllRequests() {
```

```
        // clearing repositories and queues
        pendingContentRequestsPriorityQueue.clear();
        allContentRequestsPriorityQueue.clear();
        contentRequestsRepository.clear();

        File[] requestTextFiles =
retrieveRequestFiles(departmentRequestDir);

        if (requestTextFiles == null){
            return false;
        }

        for (File f : requestTextFiles){

            editor.processTextFile(departmentRequestDir + f.getName());
            String[] requests = editor.getArrayNames();

            for (String s : requests){

                ContentRequests request = new ContentRequests(s,
                        editor.retrieveValue(s, "theme"),
                        editor.retrieveValue(s, "contentCategory"),
                        editor.retrieveValue(s, "platform"),
                        editor.retrieveValue(s, "duration"),
                        editor.retrieveValue(s, "deadline"),
                        editor.retrieveValue(s, "description"),
                        editor.retrieveValue(s, "tasks"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "priority"),
                        editor.retrieveValue(s, "specialReqs"),
                        editor.retrieveValue(s, "dateIssued"),
                        editor.retrieveValue(s, "resolved"),
                        (f.getName()));

                if (request.getTheme() == null ||
request.getContentCategory() == null ||
                        request.getPlatform() == null || request.getDuration() ==
null ||
                        request.getDeadline() == null || request.getDescription()
== null ||
                        request.getTasks() == null || request.getDepartment() ==
null ||
                        request.getPriority() == null || request.getSpecialReqs()
== null ||
                        request.getDateIssued() == null || request.getResolved()
== null){

                    System.out.println("Invalid content request detected");
                    return false;
                }

                if (!Boolean.parseBoolean(request.getResolved())){
                    pendingContentRequestsPriorityQueue.add(request);
                }
                allContentRequestsPriorityQueue.add(request);
                contentRequestsRepository.put(s,request);
            }
        }
        return true;
    }

    /**
     * Retrieve request file list
     * @param fileDirectory given directory
     * @return request file list
     */
    private File[] retrieveRequestFiles(String fileDirectory) {

        File directory = new File(fileDirectory);
        File[] textFiles = null;

        if (directory.exists() && directory.isDirectory()){

            // grab list of text files
            FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
            textFiles = directory.listFiles(textFileFilter);
        }
        else {

            System.out.println("Repository not found for directory: " +
fileDirectory);
            return null;
        }

        if (textFiles != null) {

            if (textFiles.length == 0){
                System.out.println("No content requests found");
                return null;
            }
        }
        else {
            System.out.println("No content requests found");
            return null;
        }
        return textFiles;
    }

    /**
     * Retrieve all PR Planning and Review Employees
     * @return true if successful, false otherwise
     */
    private boolean retrieveAllPREmployees(){

        // clearing repositories and queues
        availablePRPlanningEmployeePriorityQueue.clear();
        availablePRReviewEmployeePriorityQueue.clear();
        allPRPlanningEmployeePriorityQueue.clear();
        allPRReviewEmployeePriorityQueue.clear();
        PRPlanningEmployeeRepository.clear();
        PRReviewEmployeeRepository.clear();

        // saving Planning employees
        File[] planningTextFiles =
retrieveEmployeeFiles(PRPlanningEmployeesDir);

        if (planningTextFiles == null){
            return false;
        }

        for (File f : planningTextFiles){

            editor.processTextFile(PRPlanningEmployeesDir +
f.getName());
            String[] employees = editor.getArrayNames();

            for (String s : employees){

                PRPlanningEmployee employee = new
PRPlanningEmployee(s,
                        editor.retrieveValue(s,"name"),
                        editor.retrieveValue(s, "division"),
                        editor.retrieveValue(s, "position"),
                        editor.retrieveValue(s, "department"),
                        editor.retrieveValue(s, "rating"),
                        editor.retrieveValue(s, "yearsOfExperience"),
                        editor.retrieveValue(s, "previousAssignment"),
                        editor.retrieveValue(s, "currentAssignment"));
```

```
            if (employee.getEmployeeID() == null ||
employee.getName() == null ||
                    employee.getDivision() == null ||
employee.getPosition() == null ||
                    employee.getDepartment() == null ||
employee.getRating() == null ||
                    employee.getYearsOfExperience() == null ||
employee.getPreviousAssignment() == null ||
                    employee.getCurrentAssignment() == null) {

                System.out.println("Invalid employee data detected");
                return false;
            }

            if (employee.getCurrentAssignment().equals("free")){

availablePRPlanningEmployeePriorityQueue.add(employee);
            }
            allPRPlanningEmployeePriorityQueue.add(employee);
            PRPlanningEmployeeRepository.put(s, employee);
        }
    }

    // saving Review employees
    File[] reviewTextFiles =
retrieveEmployeeFiles(PRReviewEmployeesDir);

    if (reviewTextFiles == null){
        return false;
    }

    for (File f : reviewTextFiles){

        editor.processTextFile(PRReviewEmployeesDir +
f.getName());
        String[] employees = editor.getArrayNames();

        for (String s : employees){

            PRReviewEmployee employee = new
PRReviewEmployee(s,
                    editor.retrieveValue(s,"name"),
                    editor.retrieveValue(s, "division"),
                    editor.retrieveValue(s, "position"),
                    editor.retrieveValue(s, "department"),
                    editor.retrieveValue(s, "rating"),
                    editor.retrieveValue(s, "yearsOfExperience"),
                    editor.retrieveValue(s, "previousAssignment"),
                    editor.retrieveValue(s, "currentAssignment"));

            if (employee.getEmployeeID() == null ||
employee.getName() == null ||
                    employee.getDivision() == null ||
employee.getPosition() == null ||
                    employee.getDepartment() == null ||
employee.getRating() == null ||
                    employee.getYearsOfExperience() == null ||
employee.getPreviousAssignment() == null ||
                    employee.getCurrentAssignment() == null) {

                System.out.println("Invalid employee data detected");
                return false;
            }

            if (employee.getCurrentAssignment().equals("free")){

availablePRReviewEmployeePriorityQueue.add(employee);
            }
            allPRReviewEmployeePriorityQueue.add(employee);
            PRReviewEmployeeRepository.put(s, employee);
        }
    }
    return true;
}

/**
 * Retrieve employee file list
 * @param fileDirectory given directory
 * @return employee file list
 */
private File[] retrieveEmployeeFiles(String fileDirectory){

    File directory = new File(fileDirectory);
    File[] textFiles = null;

    if (directory.exists() && directory.isDirectory()){

        // grab list of text files
        FilenameFilter textFileFilter = ((dir, name) ->
name.toLowerCase().endsWith(".txt"));
        textFiles = directory.listFiles(textFileFilter);
    }
    else {

        System.out.println("Repository not found for directory: " +
fileDirectory);
        return null;
    }

    if (textFiles != null) {

        if (textFiles.length == 0){
            System.out.println("No employees found");
            return null;
        }
    }
    else {
        System.out.println("No employees found");
        return null;
    }
    return textFiles;
}

/**
 * Converts ContentSchedules object into nested HashMap
 * @param schedule specified ContentSchedule
 * @return HashMap of ContentSchedule
 */
private Map<String, Object>
contentScheduleToHashmap(ContentSchedules schedule){

    Map<String, Object> innerMap = new LinkedHashMap<>();
    innerMap.put("requestID", schedule.getRequestID());
    innerMap.put("theme", schedule.getTheme());
    innerMap.put("contentCategory",
schedule.getContentCategory());
    innerMap.put("platform", schedule.getPlatform());
    innerMap.put("duration", schedule.getDuration());
    innerMap.put("deadline", schedule.getDeadline());
    innerMap.put("description", schedule.getDescription());
    innerMap.put("tasks", schedule.getTasks());
    innerMap.put("department", schedule.getDepartment());
    innerMap.put("priority", schedule.getPriority());
    innerMap.put("status", schedule.getStatus());
    innerMap.put("dateScheduled", schedule.getDateScheduled());

    int num = 0;
    for (String s : schedule.getPlanningTeamAssign()){
        String employee = "employeePlanning" + num;
```

```
            innerMap.put(employee, s);
            num++;
        }

        int num1 = 0;
        for (String s : schedule.getReviewTeamAssign()){
            String employee = "employeeReview" + num1;
            innerMap.put(employee, s);
            num1++;
        }

        Map<String, Object> outerMap = new LinkedHashMap<>();
        outerMap.put(schedule.getScheduleID(), innerMap);

        return outerMap;
    }

    /**
     * Creates printable content schedule
     * @param schedule specific content schedule
     */
    private void printContentSchedule(ContentSchedules schedule){

        BufferedWriter writer = null;

        try {
            writer = new BufferedWriter(new
FileWriter(printedContentSchedulesDir + schedule.getScheduleID() +
".txt"));

writer.write("==========================================
============================\n");
            writer.write("CONTENT SCHEDULE\n");
            writer.write("Schedule ID: " + schedule.getScheduleID() + " is
associated with Request ID: " + schedule.getRequestID() + "\n");

writer.write("---------------------------------------------------------------------\n");
            writer.write("Status: " + schedule.getStatus() + "\n");
            writer.write("Priority: " + schedule.getPriority() + " (1 = Urgent,
0 = Normal)\n");
            writer.write("Department: " + schedule.getDepartment() +
"\n");

writer.write("---------------------------------------------------------------------\n");
            writer.write("Content Category: " +
schedule.getContentCategory() + "\n");
            writer.write("Theme: " + schedule.getTheme() + "\n");
            writer.write("Platform: " + schedule.getPlatform() + "\n");
            writer.write("Description: " + schedule.getDescription() + "\n");
            writer.write("Tasks: " + schedule.getTasks() + "\n");

writer.write("---------------------------------------------------------------------\n");
            writer.write("Duration: " + schedule.getDuration() + "\n");
            writer.write("Deadline: " + schedule.getDeadline() + "\n");
            writer.write("Date Scheduled: " +
schedule.getDateScheduled() + "\n");

writer.write("---------------------------------------------------------------------\n");
            writer.write("Assigned Planning Employees:\n");
            for (String s : schedule.getPlanningTeamAssign()){
                writer.write(s + "\n");
            }
            writer.write("Assigned Review Employees:\n");
            for (String s : schedule.getReviewTeamAssign()){
                writer.write(s + "\n");
            }

writer.write("==========================================
============================\n");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
                if (writer != null) {
                    writer.close();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * Pretty print content requests
     * @param request specific content request
     */
    private void contentRequestToText(ContentRequests request) {


System.out.println("==================================
================");
        System.out.println("Security Request ID: " +
request.getRequestID());
        System.out.println("Priority: " + request.getPriority() + " (0 =
Normal | 1 = Urgent)");
        System.out.println("Department: " + request.getDepartment());
        System.out.println();
        System.out.println("Content Category: " +
request.getContentCategory());
        System.out.println("Theme: " + request.getTheme());
        System.out.println("Platform: " + request.getPlatform());
        System.out.println("Description: " + request.getDescription());
        System.out.println("Tasks: " + request.getTasks());
        System.out.println();
        System.out.println("Duration: " + request.getDuration());
        System.out.println("Deadline: " + request.getDeadline());
        System.out.println("Special Requests: " +
request.getSpecialReqs());
        System.out.println();
        System.out.println("Date Issued: " + request.getDateIssued());
        System.out.println("Resolve Status: " + request.getResolved());

System.out.println("==================================
================\n");
    }

    /**
     * Pretty print content schedules
     * @param schedule specific content schedule
     */
    private void contentScheduleToText(ContentSchedules schedule){


System.out.println("==================================
================");
        System.out.println("Schedule ID: " + schedule.getScheduleID());
        System.out.println("is associated with");
        System.out.println("Request ID: " + schedule.getRequestID());
        System.out.println("-----------------------------------------------------");
        System.out.println("Status: " + schedule.getStatus() + " (No
status | Planning | Reviewing | Revising | Ready)");
        System.out.println("Priority: " + schedule.getPriority() + " (0 =
Normal | 1 = Urgent)");
        System.out.println("Department: " + schedule.getDepartment());
        System.out.println("-----------------------------------------------------");
        System.out.println("Content Category: " +
schedule.getContentCategory());
        System.out.println("Theme: " + schedule.getTheme());
        System.out.println("Platform: " + schedule.getPlatform());
        System.out.println("Description: " + schedule.getDescription());
```

```
        System.out.println("Tasks: " + schedule.getTasks());
        System.out.println("-------------------------------------------------------");
        System.out.println("Duration: " + schedule.getDuration());
        System.out.println("Deadline: " + schedule.getDeadline());
        System.out.println("Date Scheduled: " +
schedule.getDateScheduled());
        System.out.println("-------------------------------------------------------");
        System.out.println("Assigned Planning Employees:");
        for (String s : schedule.getPlanningTeamAssign()){
            System.out.println(s);
        }
        System.out.println("-------------------------------------------------------");
        System.out.println("Assigned Review Employees:");
        for (String s : schedule.getReviewTeamAssign()){
            System.out.println(s);
        }

System.out.println("=====================================
==================\n");
    }

    /**
     * Pretty print planning employee
     * @param employee specific planning employee
     */
    private void planningEmployeeToText(PRPlanningEmployee
employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("-------------------------------------------------------");
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<\n");
    }

    /**
     * Pretty print review employee
     * @param employee specific review employee
     */
    private void reviewEmployeeToText(PRReviewEmployee
employee){


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>>>>>");
        System.out.println("Employee ID: " +
employee.getEmployeeID());
        System.out.println("Name: " + employee.getName());
        System.out.println("Division: " + employee.getDivision());
        System.out.println("Position: " + employee.getPosition());
        System.out.println("Department: " + employee.getDepartment());
        System.out.println("-------------------------------------------------------");
        System.out.println("Rating: " + employee.getRating());
        System.out.println("Years of Experience: " +
employee.getYearsOfExperience());
        System.out.println("Previous Assignment: " +
employee.getPreviousAssignment());
        System.out.println("Current Assignment: " +
employee.getCurrentAssignment());

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<\n");
    }

    /**
     * Generates IDs for content schedules
     * @return generated content schedule ID
     */
    private String IDGenerator(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MMddyyyyHHmmss");

        return timeNow.format(formatter);
    }

    /**
     * Returns current time in MM-dd-yyyy-HH-mm-ss format when
called
     * @return date in MM-dd-yyyy-HH-mm-ss format
     */
    private String dateIssuer(){

        LocalDateTime timeNow = LocalDateTime.now();
        DateTimeFormatter formatter =
DateTimeFormatter.ofPattern("MM-dd-yyyy-HH-mm-ss");

        return timeNow.format(formatter);
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

import src.App;

import java.util.Scanner;

public class ContentSchedulerController implements
src.PublicRelations.src.interfaces.ContentSchedulerController {

    private ContentSchedulerController controller;

    private ContentScheduler scheduler = null;

    public ContentSchedulerController (ContentSchedulerController
controller){
        this.controller = controller;
    }

    public ContentSchedulerController(){}

    /**
     * Run the security program
     * @throws Exception
     */
    public void run() throws Exception {

        System.out.println("Welcome to the Social Content Scheduler
System");
```

```
        boolean exit = false;

        while (!exit){

            Scanner scan = new Scanner(System.in);


System.out.println(">>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
>>>>>>>>>>>>>");
            System.out.println("Select An Action: ");
            System.out.println("1. Add New Public Relations Employee");
            System.out.println("2. Delete Public Relations Employee by
ID");
            System.out.println();
            System.out.println("3. Show Available Planning Employees");
            System.out.println("4. Show All Planning Employees");
            System.out.println("5. Show Available Review Employees");
            System.out.println("6. Show All Review Employees");
            System.out.println();
            System.out.println("7. Show Pending Content Requests");
            System.out.println("8. Show All Content Requests");
            System.out.println();
            System.out.println("9. Create Content Schedule");
            System.out.println("10. Delete Content Schedule");
            System.out.println("11. Edit Existing Content Schedule Team
Assignment");
            System.out.println("12. Edit Existing Content Schedule
Status");
            System.out.println();
            System.out.println("13. Show All Content Schedules");
            System.out.println("14. Show Content Schedules In
Planning");
            System.out.println("15. Show Content Schedules In
Reviewing");
            System.out.println("16. Show Content Schedules In
Revising");
            System.out.println("17. Show Content Schedules In Ready");
            System.out.println("18. Show Content Schedule By ID");
            System.out.println();
            System.out.println("0. Exit program");

System.out.println("<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<");

            String choice = scan.nextLine();

            switch (choice){

                case "1" -> {
                    addPREmployee();
                    System.out.println();
                }
                case "2" -> {
                    System.out.println("Enter Public Relations Employee ID
to delete: ");
                    String deleteID = scan.nextLine();
                    deletePREmployee(deleteID);
                    System.out.println();
                }
                case "3" -> {
                    showFreePlanningEmployees();
                    System.out.println();
                }
                case "4" -> {
                    showAllPlanningEmployees();
                    System.out.println();
                }
                case "5" -> {
                    showFreeReviewEmployees();
                    System.out.println();
                }
                case "6" -> {
                    showAllReviewEmployees();
                    System.out.println();
                }
                case "7" -> {
                    showPendingRequests();
                    System.out.println();
                }
                case "8" -> {
                    showAllRequests();
                    System.out.println();
                }
                case "9" -> {
                    System.out.println("Enter content request ID to make
schedule for: ");
                    String requestID = scan.nextLine();
                    createContentSchedule(requestID);
                    System.out.println();
                }
                case "10" -> {
                    System.out.println("Enter content schedule ID to delete:
");
                    String deleteScheduleID = scan.nextLine();
                    deleteContentSchedule(deleteScheduleID);
                    System.out.println();
                }
                case "11" -> {
                    System.out.println("Enter content schedule ID to modify
assignments: ");
                    String assignScheduleID = scan.nextLine();
                    editContentScheduleAssignment(assignScheduleID);
                    System.out.println();
                }
                case "12" -> {
                    System.out.println("Enter content schedule ID to modify
status: ");
                    String statusScheduleID = scan.nextLine();
                    editContentScheduleStatus(statusScheduleID);
                    System.out.println();
                }
                case "13" -> {
                    showAllContentSchedules();
                    System.out.println();
                }
                case "14" -> {
                    showPlanningContentSchedules();
                    System.out.println();
                }
                case "15" -> {
                    showReviewingContentSchedules();
                    System.out.println();
                }
                case "16" -> {
                    showRevisingContentSchedules();
                    System.out.println();
                }
                case "17" -> {
                    showReadyContentSchedules();
                    System.out.println();
                }
                case "18" -> {
                    System.out.println("Enter content schedule ID to view: ");
                    String showScheduleID = scan.nextLine();
                    showContentScheduleByID(showScheduleID);
                    System.out.println();
                }
                case "0" -> {
                    System.out.println("Closing Program");
                    App.prompt();
```

```
            }
            default -> System.out.println("Invalid choice. Try again");
          }
        }
    }

    @Override
    public boolean addPREmployee() {
        return this.getSchedulerInstance().addPREmployee();
    }

    @Override
    public boolean deletePREmployee(String employeeID) {
        return
this.getSchedulerInstance().deletePREmployee(employeeID);
    }

    @Override
    public boolean showFreePlanningEmployees() {
        return
this.getSchedulerInstance().showFreePlanningEmployees();
    }

    @Override
    public boolean showAllPlanningEmployees() {
        return this.getSchedulerInstance().showAllPlanningEmployees();
    }

    @Override
    public boolean showFreeReviewEmployees() {
        return
this.getSchedulerInstance().showFreeReviewEmployees();
    }

    @Override
    public boolean showAllReviewEmployees() {
        return this.getSchedulerInstance().showAllReviewEmployees();
    }

    @Override
    public boolean showPendingRequests() {
        return this.getSchedulerInstance().showPendingRequests();
    }

    @Override
    public boolean showAllRequests() {
        return this.getSchedulerInstance().showAllRequests();
    }

    @Override
    public boolean createContentSchedule(String requestID) {
        return
this.getSchedulerInstance().createContentSchedule(requestID);
    }

    @Override
    public boolean deleteContentSchedule(String contentID) {
        return
this.getSchedulerInstance().deleteContentSchedule(contentID);
    }

    @Override
    public boolean editContentScheduleAssignment(String contentID) {
        return
this.getSchedulerInstance().editContentScheduleAssignment(contentI
D);
    }

    @Override
    public boolean editContentScheduleStatus(String contentID) {
        return
this.getSchedulerInstance().editContentScheduleStatus(contentID);
    }

    @Override
    public boolean showAllContentSchedules() {
        return this.getSchedulerInstance().showAllContentSchedules();
    }

    @Override
    public boolean showPlanningContentSchedules() {
        return
this.getSchedulerInstance().showPlanningContentSchedules();
    }

    @Override
    public boolean showReviewingContentSchedules() {
        return
this.getSchedulerInstance().showReviewingContentSchedules();
    }

    @Override
    public boolean showRevisingContentSchedules() {
        return
this.getSchedulerInstance().showRevisingContentSchedules();
    }

    @Override
    public boolean showReadyContentSchedules() {
        return
this.getSchedulerInstance().showReadyContentSchedules();
    }

    @Override
    public boolean showContentScheduleByID(String contentID) {
        return
this.getSchedulerInstance().showContentScheduleByID(contentID);
    }

    /**
     * To define one instance of the class ContentScheduler to
     * call ContentScheduler methods
     * @return ContentScheduler instance
     */
    private ContentScheduler getSchedulerInstance(){

        if (scheduler == null){
            scheduler = new ContentScheduler();
        }
        return scheduler;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

import java.util.List;

public class ContentSchedules {

    String scheduleID, requestID, theme, contentCategory, platform,
duration, deadline,
            description, tasks, department, priority, status,
dateScheduled, fileName;
    private List<String> planningTeamAssign, reviewTeamAssign;

    public ContentSchedules(){}
```

```
    public ContentSchedules(String scheduleID, String requestID,
String theme,
                            String contentCategory, String platform, String
duration,
                            String deadline, String description, String tasks,
                            String department, String priority, String status,
                            String dateIssued, String fileName){

        this.scheduleID = scheduleID;
        this.requestID = requestID;
        this.theme = theme;
        this.contentCategory = contentCategory;
        this.platform = platform;
        this.duration = duration;
        this.deadline = deadline;
        this.description = description;
        this.tasks = tasks;
        this.department = department;
        this.priority = priority;
        this.status = status;
        this.dateScheduled = dateIssued;
        this.fileName = fileName;
    }

    public String getScheduleID() {
        return scheduleID;
    }

    public String getRequestID() {
        return requestID;
    }

    public String getTheme() {
        return theme;
    }

    public String getContentCategory() {
        return contentCategory;
    }

    public String getPlatform() {
        return platform;
    }

    public String getDuration() {
        return duration;
    }

    public String getDeadline() {
        return deadline;
    }

    public String getDescription() {
        return description;
    }

    public String getTasks() {
        return tasks;
    }

    public String getDepartment() {
        return department;
    }

    public String getPriority() {
        return priority;
    }

    public String getStatus() {
        return status;
    }

    public String getDateScheduled() {
        return dateScheduled;
    }

    public String getFileName() {
        return fileName;
    }

    public List<String> getPlanningTeamAssign() {
        return planningTeamAssign;
    }

    public List<String> getReviewTeamAssign() {
        return reviewTeamAssign;
    }

    public void setScheduleID(String scheduleID) {
        this.scheduleID = scheduleID;
    }

    public void setRequestID(String requestID) {
        this.requestID = requestID;
    }

    public void setTheme(String theme) {
        this.theme = theme;
    }

    public void setContentCategory(String contentCategory) {
        this.contentCategory = contentCategory;
    }

    public void setPlatform(String platform) {
        this.platform = platform;
    }

    public void setDuration(String duration) {
        this.duration = duration;
    }

    public void setDeadline(String deadline) {
        this.deadline = deadline;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    public void setTasks(String tasks) {
        this.tasks = tasks;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setPriority(String priority) {
        this.priority = priority;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public void setDateScheduled(String dateScheduled) {
        this.dateScheduled = dateScheduled;
    }
```

```
    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public void setPlanningTeamAssign(List<String>
planningTeamAssign) {
        this.planningTeamAssign = planningTeamAssign;
    }

    public void setReviewTeamAssign(List<String> reviewTeamAssign)
{
        this.reviewTeamAssign = reviewTeamAssign;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

public class PRPlanningEmployee {

    private String employeeID, name, division, position, department,
rating,
                yearsOfExperience, previousAssignment,
currentAssignment;

    public PRPlanningEmployee(){}

    public PRPlanningEmployee (String employeeID, String name,
String division,
                        String position, String department, String rating,
                        String yearsOfExperience, String
previousAssignment, String currentAssignment){

        this.employeeID = employeeID;
        this.name = name;
        this.division = division;
        this.position = position;
        this.department = department;
        this.rating = rating;
        this.yearsOfExperience = yearsOfExperience;
        this.previousAssignment = previousAssignment;
        this.currentAssignment = currentAssignment;
    }

    public String getEmployeeID() {
        return employeeID;
    }

    public String getName() {
        return name;
    }

    public String getDivision() {
        return division;
    }

    public String getPosition() {
        return position;
    }

    public String getDepartment() {
        return department;
    }

    public String getRating() {
        return rating;
    }

    public String getYearsOfExperience() {
        return yearsOfExperience;
    }

    public String getPreviousAssignment() {
        return previousAssignment;
    }

    public String getCurrentAssignment() {
        return currentAssignment;
    }

    public void setEmployeeID(String employeeID) {
        this.employeeID = employeeID;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDivision(String division) {
        this.division = division;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setRating(String rating) {
        this.rating = rating;
    }

    public void setYearsOfExperience(String yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }

    public void setPreviousAssignment(String previousAssignment) {
        this.previousAssignment = previousAssignment;
    }

    public void setCurrentAssignment(String currentAssignment) {
        this.currentAssignment = currentAssignment;
    }
}

/**
 * @author Kenny
 */

package src.PublicRelations.src;

public class PRReviewEmployee {

    private String employeeID, name, division, position, department,
rating,
                yearsOfExperience, previousAssignment,
currentAssignment;

    public PRReviewEmployee(){}

    public PRReviewEmployee (String employeeID, String name,
String division,
                        String position, String department, String rating,
```

```
                    String yearsOfExperience, String
previousAssignment, String currentAssignment){

        this.employeeID = employeeID;
        this.name = name;
        this.division = division;
        this.position = position;
        this.department = department;
        this.rating = rating;
        this.yearsOfExperience = yearsOfExperience;
        this.previousAssignment = previousAssignment;
        this.currentAssignment = currentAssignment;
    }

    public String getEmployeeID() {
        return employeeID;
    }

    public String getName() {
        return name;
    }

    public String getDivision() {
        return division;
    }

    public String getPosition() {
        return position;
    }

    public String getDepartment() {
        return department;
    }

    public String getRating() {
        return rating;
    }

    public String getYearsOfExperience() {
        return yearsOfExperience;
    }

    public String getPreviousAssignment() {
        return previousAssignment;
    }

    public String getCurrentAssignment() {
        return currentAssignment;
    }

    public void setEmployeeID(String employeeID) {
        this.employeeID = employeeID;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setDivision(String division) {
        this.division = division;
    }

    public void setPosition(String position) {
        this.position = position;
    }

    public void setDepartment(String department) {
        this.department = department;
    }

    public void setRating(String rating) {
        this.rating = rating;
    }

    public void setYearsOfExperience(String yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }

    public void setPreviousAssignment(String previousAssignment) {
        this.previousAssignment = previousAssignment;
    }

    public void setCurrentAssignment(String currentAssignment) {
        this.currentAssignment = currentAssignment;
    }
}
```

Sale Department:

```
public class BasicSalesController implements
SalesController {
   SalesManagement sale=new
BasicSalesManage();
   SalesInventoryRequest sir=new
SalesInventoryRequest();


   public void run()
   {
      SalesCommand scd=new
SalesCommand();

      System.out.println("Welcome to the Sales
Management System");

      System.out.println("Enter 'help' to see
available commands");
      System.out.println("Enter 'exit' to leave the
sales Management System");
      System.out.println("Enter the command for
your task");

      while(true) {
         Scanner sc = new Scanner(System.in);
         System.out.println("Enter command: ");
         String command = sc.nextLine();
         scd.perform(command);
      }
   }

   public void changePrice(int price, String
pname)
   {
      sir.changePrice(price,pname);
   }

   public void registerRetailer(String name,
String location) {
      sale.registerRetailer(name, location);
   }


   public void addOrder(int rid, String date,
Map<String, Integer> products) {
      sale.addOrder(rid, date, products);
   }

   public void printReceipt(int orderId)
   {
     sale.printReceipt(orderId);
   }
   public void returnOrder(int rid, int oid,
Map<String, Integer> rproducts, int days)
   {
     sale.returnOrder(rid,oid,rproducts,days);
   }

   public void viewRetailers()
   {
      sale.viewRetailers();
   }

   public void viewOrders(){
      sale.viewOrders();
   }

   public void viewAvaProducts()
   {
      sir.viewAvaProducts();
   }

   public void viewRegisteredProducts()
   {
      sir.viewProducts();
```

```
public class SaleAnalyzer {

   SalesInventoryRequest sir= new
SalesInventoryRequest();

   private int thresholdCount = 10;
   DateManagement dm =new DateManagement();

   public int NumOfProductSold(Map<String,
Map<String, String>> orders, String pname)
   {
      int count =0;
      for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
      {
         Map<String, String> odetail = item.getValue();
            if(odetail.containsKey(pname)) {
               count =
Integer.parseInt(odetail.get(pname));
            }
      }
      return count;

   }

   public int NumOfProductInDay(Map<String,
Map<String, String>> orders, String pname, String date)
   {
      int count =0;
      for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
      {
         Map<String, String> odetail = item.getValue();
         if(odetail.get("date").equals(date)) {
            if(odetail.containsKey(pname)) {
               count =
Integer.parseInt(odetail.get(pname));
            }
         }
      }
      return count;
   }

   public int NumOfProductInMonth(Map<String,
Map<String, String>> orders, String pname, int month,
int year)
   {
      int count =0;
      for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
      {
         Map<String, String> odetail = item.getValue();
         if( dm.isInMonth(odetail.get("date"),month,year))
{
            if(odetail.containsKey(pname)) {
               count =
Integer.parseInt(odetail.get(pname));
            }
         }
      }
      return count;
   }

   public int NumOfProductInYear(Map<String,
Map<String, String>> orders, String pname, int year)
   {
      int count =0;

      for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
      {
         Map<String, String> odetail = item.getValue();
         if( dm.isInYear(odetail.get("date"), year)) {
            if(odetail.containsKey(pname)) {
               count =
```

```
    }

    public void NumOfProductSold(String pname)
    {
        sale.NumOfProductSold(pname);
    }
    public void NumOfProductInDay(String
pname, String date){
        sale.NumOfProductInDay(pname,date);
    }
    public void NumOfProductInMonth(String
pname, int month, int year){

sale.NumOfProductInMonth(pname,month,year);
    }
    public void NumOfProductInYear(String
pname, int year){
        sale.NumOfProductInYear(pname,year);
    }
    public void MonthSaleReport(int month, int
year){
        sale.MonthSaleReport(month,year);
    }
    public void DaySaleReport(String date){
        sale.DaySaleReport(date);
    }
    public void YearSaleReport(int year){
        sale.YearSaleReport(year);
    }
    public void priceManipulate(String pname){
        sale.priceManipulate(pname);
    }
    public void trackRetailerReward(int rid){
        sale.trackRetailerReward(rid);
    }

    public void getRetailerPoints(int rid)
    {
        System.out.println("Available Reward
Points "+sale.getRetailerPoints(rid));
    }

}
public class BasicSalesManage implements
SalesManagement {

    // id to details
    private Map<String, Map<String, String>>
retailers;

    // id to details
    private Map<String, Map<String, String>>
orders;

    private SalesInventoryRequest sir = new
SalesInventoryRequest();
    private SaleAnalyzer sa = new
SaleAnalyzer();

    private RewardRetailer rewardRetailer = new
RewardRetailer();
    private String location;
    private String repoPath =
"src/Sales/repository/";

    PoorTextEditor textEditor;

    public BasicSalesManage() {
        this.retailers = new HashMap<>();
        this.orders = new HashMap<>();
        location = "Main";

        set();
    }

    private void set() {
        String rPath = repoPath + "Retailers.txt";
        String oPath = repoPath + "Orders.txt";
Integer.parseInt(odetail.get(pname));
                }
            }
        }
        return count;
    }

    public void MonthSaleReport(Map<String,
Map<String, String>> orders, int month, int year){

        Map<String, Map<String, String>> regProds = new
HashMap<>(sir.regProd());

        for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
        {
            Map<String, String> odetail = item.getValue();
            if( dm.isInMonth(odetail.get("date"),month,year))
{

                for(Map.Entry<String, String> pdetail:
odetail.entrySet())
                {
                    String pname = pdetail.getKey();
                    if(regProds.containsKey(pname))
                    {
                        regProds.get(pname).put("count",
pdetail.getValue());
                    }

                }

            }
        }
        printSaleReport(regProds,"Sale Report for "+
month +" month" + year +" year");
    }

    public void DaySaleReport(Map<String, Map<String,
String>> orders, String date){

        Map<String, Map<String, String>> regProds = new
HashMap<>(sir.regProd());

        for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
        {
            Map<String, String> odetail = item.getValue();
            if( odetail.get("date").equals(date)) {

                for(Map.Entry<String, String> pdetail:
odetail.entrySet())
                {
                    String pname = pdetail.getKey();
                    if(regProds.containsKey(pname))
                    {
                        regProds.get(pname).put("count",
pdetail.getValue());
                    }

                }

            }
        }
        printSaleReport(regProds,"Sale Report for sales
on"+ date);
    }

    public void YearSaleReport(Map<String, Map<String,
String>> orders, int year){

        Map<String, Map<String, String>> regProds = new
HashMap<>(sir.regProd());

        for(Map.Entry<String, Map<String, String>> item :
orders.entrySet())
        {
            Map<String, String> odetail = item.getValue();
            if( dm.isInYear(odetail.get("date"),year)) {
```

```
        try {
            File rFile = new File(rPath);
            if (rFile.exists()) {
                textEditor = new PoorTextEditor();
                textEditor.processTextFile(rPath);
                retailers =
textEditor.getRepositoryString();
            }
        } catch (Exception e) {
            System.out.println("Error processing
Retailers.txt");

        }
        try {
            File oFile = new File(oPath);
            if (oFile.exists()) {
                textEditor = new PoorTextEditor();
                textEditor.processTextFile(oPath);
                orders =
textEditor.getRepositoryString();
            }
        } catch (Exception e) {
            System.out.println("Error processing
Retailers.txt");
        }
    }


    public void addOrder(int rid, String date,
Map<String, Integer> products) {

        int size = orders.size();
        String id = String.valueOf(rid);
        if (retailers.containsKey(id)) {

            Map<String, String> odetail =
sir.addPToOrder(products);
            odetail.put("retailerId",
String.valueOf(rid));
            odetail.put("date", date);

            odetail.put("appliedRewardPoints", "0");

            int avaPoints = getRetailerPoints(rid);
            System.out.println("You have " +
avaPoints + " reward points");

            String wantToApply = "";

            Scanner sc = new Scanner(System.in);
            System.out.println("Do you want use any
points to pay this order (yes/no): ");
            wantToApply = sc.nextLine();

            if (wantToApply.equals("yes")) {
                System.out.println("How many Points
do you want to use?");

                int pointUse = sc.nextInt();
                sc.nextLine();
                while (!removePointRetailer(pointUse,
rid)) {
                    System.out.println("Requested
Points exceed the available points as you only
have " + avaPoints);
                    System.out.println("Enter Points
with in available limit to apply");
                    pointUse = sc.nextInt();
                    sc.nextLine();
                }
                odetail.put("appliedRewardPoints",
String.valueOf(pointUse));

            } else {
                System.out.println("Proceeding with
out reward points");
            }

            orders.put(String.valueOf(size + 1),
                for(Map.Entry<String, String> pdetail:
odetail.entrySet())
                {
                    String pname = pdetail.getKey();
                    if(regProds.containsKey(pname))
                    {
                        regProds.get(pname).put("count",
pdetail.getValue());
                    }

                }

            }
        }

        printSaleReport(regProds,"Sale Report for "+ year
+" year");
    }

    public void printSaleReport(Map<String, Map<String,
String>> regProds, String title)
    {
        String maxSoldname = "";
        String minSoldname = "";
        int maxSoldCount = Integer.MIN_VALUE;
        int minSoldCount = Integer.MAX_VALUE;

System.out.println("----------------------------------------------
-------------------------------------------------------------");
        System.out.printf("%40s %n", title);

System.out.println("----------------------------------------------
-------------------------------------------------------------");
        System.out.printf("%-30s %-30s%n", "product
name", "quantity sold");
        for(Map.Entry<String, Map<String, String>>
pdeatil: regProds.entrySet())
        {
            String pname = pdeatil.getKey();
            String count = pdeatil.getValue().get("count");
            if(Integer.parseInt(count) > maxSoldCount)
            {
                maxSoldname = pname;
                maxSoldCount = Integer.parseInt(count);
            }
            if(Integer.parseInt(count) < minSoldCount)
            {
                minSoldname=pname;
                minSoldCount= Integer.parseInt(count);
            }
            System.out.printf("%-30s %-30s%n", pname,
count);
        }

System.out.println("----------------------------------------------
-------------------------------------------------------------");
        System.out.printf("%40s %n", "Report Analysis");

System.out.println("----------------------------------------------
-------------------------------------------------------------");
        System.out.printf("%-30s -----> %-10s%n", "Name
of Highest Demand Product : "+ maxSoldname,
"quantity sold : "+ maxSoldCount);
        System.out.printf("%-30s -----> %-10s%n", "Name
of Lowest Demand Product : "+ minSoldname, "quantity
sold : "+ minSoldCount);

System.out.println("----------------------------------------------
-------------------------------------------------------------");

        saveSaleReportToFile(regProds,title);
    }


    public void saveSaleReportToFile(Map<String,
Map<String, String>> regProds, String title) {
        try (BufferedWriter writer = new BufferedWriter(new
FileWriter("src/Sales/repository/SaleReports/"+toCamel
```

```
odetail);

            System.out.println("order placed by id: "
+ size + 1);
            int points = calReward(odetail);
            System.out.println("Earned Reward
Points with this order :" + points);

            System.out.println(" Reward Scheme:
every 100 bucks purchase earn 5 reward points
where 1 point equivalent to 1 buck");
            System.out.println(" You can use these
points for any your future orders.");
            textEditor = new PoorTextEditor();
            textEditor.setRepositoryStrings(orders);
            textEditor.writeToTextFile(repoPath +
"Orders.txt");

            // to update reward points and order
details
            set();
            addPointRetailer(points, rid);

            System.out.println("use command to print
receipt if needed using the order id");
        } else {
            System.out.println("Retailer id does not
exist. Register the retailer!");
        }

    }

    public void NumOfProductSold(String pname)
{

        System.out.println(" The total quantity of
"+pname +" sold across all sales is
"+sa.NumOfProductSold(orders, pname));
    }

    public void NumOfProductInDay(String
pname, String date) {

        System.out.println(" The total quantity of
"+pname +" sold on " + date +" is
"+sa.NumOfProductInDay(orders, pname,
date));
    }

    public void NumOfProductInMonth(String
pname, int month, int year) {
        System.out.println(" The total quantity of
"+pname +" sold during " + month +" month "+
year+ " year is
"+sa.NumOfProductInMonth(orders, pname,
month, year));
    }

    public void NumOfProductInYear(String
pname, int year) {
        System.out.println(" The total quantity of
"+pname +" sold during " + year +" year is
"+sa.NumOfProductInYear(orders, pname,
year));
    }

    public void MonthSaleReport(int month, int
year) {
        sa.MonthSaleReport(orders, month, year);
    }

    public void DaySaleReport(String date) {
        sa.DaySaleReport(orders, date);
    }

    public void YearSaleReport(int year) {
        sa.YearSaleReport(orders, year);
    }
    public void priceManipulate(String pname)
Case(title)+".txt"))) {
            String maxSoldName = "";
            String minSoldName = "";
            int maxSoldCount = Integer.MIN_VALUE;
            int minSoldCount = Integer.MAX_VALUE;


writer.write("----------------------------------------------------------
-----------------------------------------------\n");
            writer.write(String.format("%40s %n", title));

writer.write("----------------------------------------------------------
-----------------------------------------------\n");
            writer.write(String.format("%-30s %-30s%n",
"product name", "quantity sold"));

            for (Map.Entry<String, Map<String, String>>
pDetail : regProds.entrySet()) {
                String pName = pDetail.getKey();
                String count = pDetail.getValue().get("count");
                if (Integer.parseInt(count) > maxSoldCount) {
                    maxSoldName = pName;
                    maxSoldCount = Integer.parseInt(count);
                }
                if (Integer.parseInt(count) < minSoldCount) {
                    minSoldName = pName;
                    minSoldCount = Integer.parseInt(count);
                }
                writer.write(String.format("%-30s %-30s%n",
pName, count));
            }


writer.write("----------------------------------------------------------
-----------------------------------------------\n");
            writer.write(String.format("%40s %n", "Report
Analysis"));

writer.write("----------------------------------------------------------
-----------------------------------------------\n");
            writer.write(String.format("%-30s ----->
%-10s%n", "Name of Highest Demand Product : " +
maxSoldName, "quantity sold : " + maxSoldCount));
            writer.write(String.format("%-30s ----->
%-10s%n", "Name of Lowest Demand Product : " +
minSoldName, "quantity sold : " + minSoldCount));

writer.write("----------------------------------------------------------
-----------------------------------------------\n");

        } catch (IOException e) {
            System.err.println("Error while writing to file: " +
e.getMessage());
        }
    }


    public void priceManipulate(Map<String, Map<String,
String>> orders, String pname)
    {
        System.out.println(" The following recommendation
considered entire sales history" +
                "and default threshold count (10) to make
decision : ");
        if(NumOfProductSold(orders,pname) >=
thresholdCount)
        {
          System.out.println("Recommended to increase
the price");

        }else{
            System.out.println("Recommended to increase
the price");
        }

System.out.println("----------------------------------------------
-----------------------------------------------------------");
        System.out.println("Note : Threshold Count is the
minimum count of product required to be sold to rise the
```

```
    {
        sa.priceManipulate(orders, pname);
    }

    public void trackRetailerReward(int rid)
    {
        Scanner sc = new Scanner(System.in);

        System.out.println("Specify how you want
to track rewards (By 'order' or 'month' or 'year'):
");
        String opt =sc.nextLine();
        String wantToCont;
       do{
          switch (opt){
             case "order":
                System.out.println("Enter order ID
for reward analysis :");
                int oid = sc.nextInt();
                sc.nextLine();

rewardRetailer.rewardByOrderId(rid,oid,orders);
             case "month":
                System.out.println("Enter the
Month(MM) for reward analysis :");
                int month = sc.nextInt();
                sc.nextLine();
                System.out.println("Enter the
Year(YYYY) for reward analysis :");
                int year = sc.nextInt();
                sc.nextLine();

rewardRetailer.rewardByMonth(rid,month,year,or
ders);
             case "year":
                System.out.println("Enter the
Year(YYYY) for reward analysis :");
                int year1 = sc.nextInt();
                sc.nextLine();

rewardRetailer.rewardByYear(rid,year1,orders);
             default:
                System.out.println("Select correct
option('order' or 'month' or 'year'): ");
                opt =sc.nextLine();
                wantToCont = "yes";
          }
          System.out.println("Do you want analyse
rewards with other scales (yes/no)?");
          wantToCont =sc.nextLine();
          System.out.println("Specify how you want
to track rewards (By 'order' or 'month' or 'year'):
");
          opt =sc.nextLine();

       }while(wantToCont.equals("yes"));
    }

    public int getRetailerPoints(int rid) {
        return
Integer.parseInt(retailers.get(String.valueOf(rid)).
get("rewardPoints"));
    }

    private int calReward(Map<String, String>
odetail) {

        int orderAmount = sir.orderTotal(odetail)[0];

        return orderAmount % 5;
    }

    public void returnOrder(int rid, int oid,
Map<String, Integer> rproducts, int days) {
        String rId = String.valueOf(rid);
        String oId = String.valueOf(oid);
        if (retailers.containsKey(rId)) {

            if (orders.containsKey(oId)) {
```

```
price " +
                "or if does not reach the count drop the price.
" );

System.out.println("----------------------------------------------
------------------------------------------------------------");

        System.out.println("Do you want recommendation
on customised sales range and threshold count?
(yes/no)");
        Scanner sc =new Scanner(System.in);
        String wantToContinue = sc.nextLine();

        while(wantToContinue.equals("yes"))
        {
            System.out.println("Specify the range of sales to
be considered (day or month or year) ");
            String range =sc.nextLine();
            System.out.println("Specify the Threshold count
for making recommendation :");
            thresholdCount =sc.nextInt();
            sc.nextLine();

            switch (range) {

                case "day":
                    System.out.println("Enter the
date(DD-MM-YYYY) for sale analysis :");
                    String date = sc.nextLine();
                    System.out.println(" The following
recommendation considered sales on" + date +
                            "and used threshold count ("+
thresholdCount +") to make decision : ");
                    if(NumOfProductInDay(orders,pname,date)
>= thresholdCount)
                    {
                        System.out.println("Recommended to
increase the price");

                    }else{
                        System.out.println("Recommended to
increase the price");
                    }

                case "month":
                    System.out.println("Enter the Month(MM)
for sale analysis :");
                    int month = sc.nextInt();
                    sc.nextLine();
                    System.out.println("Enter the Year(YYYY)
for sale analysis :");
                    int year = sc.nextInt();
                    sc.nextLine();
                    System.out.println(" The following
recommendation considered sales during" + month
+"/"+ year +
                            "and used threshold count ("+
thresholdCount +") to make decision : ");

if(NumOfProductInMonth(orders,pname,month,year) >=
thresholdCount)
                    {
                        System.out.println("Recommended to
increase the price");

                    }else{
                        System.out.println("Recommended to
increase the price");
                    }

                case "year":
                    System.out.println("Enter the Year(YYYY)
for sale analysis :");
                    int year1 = sc.nextInt();
                    sc.nextLine();
                    System.out.println(" The following
recommendation considered sales during" + year1 +
                            "and used threshold count ("+
thresholdCount +") to make decision : ");
```

```
            if (checkReturnTime(days, oid)) {
                sir.removePToOrder(rproducts);
                System.out.println("return
successful and products sent to inventory");
            } else {
                System.out.println("Order passes
return period co cannot be returned");
            }

        } else {
            System.out.println("Order id does not
exist!");
        }
    } else {
        System.out.println("Retailer id does not
exist!");
    }
  }

  public void addPointRetailer(int points, int id) {
    int curPoints =
Integer.parseInt(retailers.get(String.valueOf(id)).
get("rewardPoints")) + points;

retailers.get(String.valueOf(id)).put("rewardPoint
s", String.valueOf(curPoints));
    textEditor = new PoorTextEditor();
    textEditor.setRepositoryStrings(retailers);
    textEditor.writeToTextFile(repoPath +
"Retailers.txt");
  }

  public boolean removePointRetailer(int points,
int id) {

    int curPoints =
Integer.parseInt(retailers.get(String.valueOf(id)).
get("rewardPoints"));

    if (curPoints >= points) {

      curPoints =
Integer.parseInt(retailers.get(String.valueOf(id)).
get("rewardPoints")) - points;

retailers.get(String.valueOf(id)).put("rewardPoint
s", String.valueOf(curPoints));
      textEditor = new PoorTextEditor();
      textEditor.setRepositoryStrings(retailers);
      textEditor.writeToTextFile(repoPath +
"Retailers.txt");

      return true;
    } else {

      return false;
    }
  }


  public void registerRetailer(String name,
String retailerLocation) {

    Retailer rd = new BasicRetailer(name,
retailerLocation);
    String size = String.valueOf(retailers.size()
+ 1);

    retailers.put(size, rd.getRDetails());

    textEditor = new PoorTextEditor();
    textEditor.setRepositoryStrings(retailers);
    textEditor.writeToTextFile(repoPath +
"Retailers.txt");
    rd.print();

    System.out.println("Retailer registered with
id:" + size);
  }
```

```
if(NumOfProductInYear(orders,pname,year1) >=
thresholdCount)
            {
                System.out.println("Recommended to
increase the price");

            }else{
                System.out.println("Recommended to
increase the price");
            }

        }

System.out.println("----------------------------------------------
------------------------------------------------------------");
        System.out.println("Note : Threshold Count is
the minimum count of product required to be sold to rise
the price " +
                "or if does not reach the count drop the
price. " );

System.out.println("----------------------------------------------
------------------------------------------------------------");

        System.out.println("Do you want
recommendation on other customised sales range and
threshold count? (yes/no)");
        wantToContinue = sc.nextLine();
    }

  }

  public static String toCamelCase(String input) {
    // Split the input string into words based on spaces
    String[] words = input.split("\\s+");

    // Initialize a StringBuilder for the result
    StringBuilder camelCase = new StringBuilder();

    // Process each word
    for (int i = 0; i < words.length; i++) {
      String word = words[i].toLowerCase(); // Convert
to lowercase
      if (i == 0) {
        // First word should start with a lowercase
letter
        camelCase.append(word);
      } else {
        // Capitalize the first letter of subsequent
words

camelCase.append(Character.toUpperCase(word.charA
t(0))).append(word.substring(1));
      }
    }

    return camelCase.toString();
  }


}
public class SalesInventoryRequest {

  Map<String, Map<String, String>> products;
  private Map<String, Map<String, String>>
availableProducts;
  BasicStorageManage sm;

  public SalesInventoryRequest() {
    sm = new BasicStorageManage();
    products = sm.getProducts();
    availableProducts = sm.getAvailableProducts();
  }

  public void removePQ(String pname, int
quantityRemoved) {
    sm.removeProductCount(pname,
quantityRemoved);
```

```
    public boolean checkReturnTime(int days, int
orderId) {
        return days <= 15;
    }


    public void printReceipt(int orderId) {
        Map<String, String> odetails = new
HashMap<>(orders.get(String.valueOf(orderId)))
;
        sir.receiptPrint(odetails);
    }


    public void viewRetailers() {
        print(retailers, "Registered Retailers");
    }


    public String getRetailerById(int rid) {
        return
retailers.get(String.valueOf(rid)).get("name");
    }

    public void viewOrders() {
        print(orders, "Orders");
    }

    public void print(Map<String, Map<String,
String>> items, String whichItem) {

System.out.println("----------------------------------------
---------------");
        System.out.println(whichItem);
        for (Map.Entry<String, Map<String,
String>> itms : items.entrySet()) {
            System.out.println(itms.getKey() + ":");
            for (Map.Entry<String, String> sitm :
itms.getValue().entrySet()) {
                System.out.printf("%-30s %s%n",
sitm.getKey(), sitm.getValue());
            }
        }

System.out.println("----------------------------------------
---------------");
    }
}
public class DateManagement {


    public static void main(String[] args) {
        Scanner scanner = new
Scanner(System.in);

        // Input the first date
        System.out.println("Enter the first date in
the format DD-MM-YYYY:");
        String date1 = scanner.nextLine();

        // Input the second date
        System.out.println("Enter the second date
in the format DD-MM-YYYY:");
        String date2 = scanner.nextLine();

        int daysGap = daysGap(date1, date2);

        System.out.println(daysGap);
    }

    /**
     * Function to calculate the number of days in
a given month
     */
    public static int getDaysInMonth(int month, int
year) {
        switch (month) {
            case 1:
    }

    public void addPQ(String pname, int
quantityRemoved) {
        sm.addProductCount(pname, quantityRemoved);
    }

    public void changePrice(int price, String pname) {
        sm.changePrice(price, pname);
    }

    public int getPrice(String pname) {
        return
Integer.parseInt(availableProducts.get(pname).get("pric
e"));
    }

    public boolean checkAvailability(Map<String, Integer>
orderProducts) {

        for (Map.Entry<String, Integer> op :
orderProducts.entrySet()) {
            String pname = op.getKey();
            if (availableProducts.containsKey(pname)) {
                int avaCount =
Integer.parseInt(availableProducts.get(op.getKey()).get(
"count"));
                if (avaCount < op.getValue()) {
                    System.out.println("Insufficient quantity,
available quantity is " + avaCount + " for " + pname);
                }
            } else {
                System.out.println(pname + " not available
currently");
            }
        }
        return true;
    }


    public Map<String, String> addPToOrder(Map<String,
Integer> products) {
        Map<String, String> cOrder = new HashMap<>();

        if (checkAvailability(products)) {
            String pname;
            int quan;
            for (Map.Entry<String, Integer> p :
products.entrySet()) {
                pname = p.getKey();
                quan = p.getValue();
                cOrder.put(pname, String.valueOf(quan));
                removePQ(pname, quan);
            }
        }
        return cOrder;
    }

    public void removePToOrder(Map<String, Integer>
products) {
        String pname;
        int quan;
        for (Map.Entry<String, Integer> p :
products.entrySet()) {
            pname = p.getKey();
            quan = p.getValue();
            addPQ(pname, quan);

        }
    }

    public void receiptPrint(Map<String, String> odetails)
{
        String rid = odetails.remove("retailerId");
        String date=odetails.remove("date");
        String appliedRewardPoints=
odetails.remove("appliedRewardPoints");

        int total = 0;
```

```
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                return 31;
            case 4:
            case 6:
            case 9:
            case 11:
                return 30;
            case 2:
                return isLeapYear(year) ? 29 : 28;
            default:
                throw new
IllegalArgumentException("Invalid month");
        }
    }

    /**
     * Function to check if a year is a leap year
     */
    public static boolean isLeapYear(int year) {
        if (year % 4 == 0) {
            if (year % 100 == 0) {
                return year % 400 == 0;
            }
            return true;
        }
        return false;
    }


    public boolean isInYear(String date, int year)
    {
        String[] dateParts = date.split("-");
        int actualYear=
Integer.parseInt(dateParts[2]);

        return actualYear == year;
    }


    public boolean isInMonth(String date, int
month, int year)
    {
        String[] dateParts = date.split("-");
        int actualMonth=
Integer.parseInt(dateParts[1]);
        int actualYear =
Integer.parseInt(dateParts[2]);

        return (actualMonth == month &&
actualYear == year);
    }



    /**
     * Function to calculate the difference in days
between two dates
     */
    public static int daysGap(String date1, String
date2) {
        String[] firstDateParts = date1.split("-");
        int day1 =
Integer.parseInt(firstDateParts[0]);
        int month1 =
Integer.parseInt(firstDateParts[1]);
        int year1 =
Integer.parseInt(firstDateParts[2]);

        String[] secondDateParts = date2.split("-");
        int day2 =
Integer.parseInt(secondDateParts[0]);
        int month2 =
Integer.parseInt(secondDateParts[1]);
        int year2 =
        String pname;
        String quan;
        int price;
        int ptotal;

System.out.println("----------------------------------------------
-------------------------------------------------------");
        System.out.printf("%60s %n", "Receipt");

System.out.println("----------------------------------------------
-------------------------------------------------------");
        System.out.println("Retailer id :" + rid);
        System.out.println("Order date :" + date);
        System.out.printf("%-30s %-30s %-30s %s%n",
"product name", "quantity", "per quantity", "total");
        for (Map.Entry<String, String> itms :
odetails.entrySet()) {
            pname = itms.getKey();
            quan = itms.getValue();
            price = getPrice(pname);
            ptotal = price * Integer.parseInt(quan);
            total += ptotal;
            System.out.printf("%-30s %-30s %-30d %d%n",
pname, quan, price, ptotal);
        }

        System.out.println("Actual Total Amount :" + total);
        System.out.println("Applied Reward points :" +
appliedRewardPoints);
        System.out.println("Paid Amount :" + (total-
Integer.parseInt(appliedRewardPoints)));


System.out.println("----------------------------------------------
-------------------------------------------------------");
    }

    public int[] orderTotal(Map<String, String>
orderDetails)
    {
        Map<String, String> odetails = new
HashMap<>(orderDetails);

        String rid = odetails.remove("retailerId");
        String date=odetails.remove("date");
        String appliedRewardPoints=
odetails.remove("appliedRewardPoints");

        int total = 0;
        String pname;
        String quan;
        int price;
        int ptotal;
        for (Map.Entry<String, String> itms :
odetails.entrySet()) {
            pname = itms.getKey();
            quan = itms.getValue();
            price = getPrice(pname);
            ptotal = price * Integer.parseInt(quan);
            total += ptotal;
        }

        int[] rewardDetail =new int[3];
        // order total
        rewardDetail[0] = total;
        // reward point earned with the order
        rewardDetail[1] = total%5;
        // reward point applied for the order
        rewardDetail[2] =
Integer.parseInt(appliedRewardPoints);

        return rewardDetail;
    }

    public void viewAvaProducts() {

        print(availableProducts, "Available products at
Storage");
```

```
Integer.parseInt(secondDateParts[2]);

        int totalDays = 0;

        //handle months counted twice and
unnecessary months
        if (year1 == year2) {
            for(int month= month1; month < month2;
month++) {
                totalDays += getDaysInMonth(month,
year1);
            }
            // Subtract days already passed in the
first month, exclude the current day
            totalDays -= (day1-1);
            // Add days passed in the last month
            totalDays += day2;
        }else{
            // Add days for the full years in between
            for (int year = year1+1; year < year2;
year++) {
                totalDays += isLeapYear(year) ? 366 :
365;
            }

            // Add days for months in the first year
            for (int month = month1; month <= 12;
month++) {
                totalDays += getDaysInMonth(month,
year1);
            }

            // Subtract days already passed in the
first month, exclude the current day
            totalDays -= (day1-1);

            // Add days for months in the second
year
            for (int month = 1; month < month2;
month++) {
                totalDays += getDaysInMonth(month,
year2);
            }

            // Add days passed in the last month
            totalDays += day2;
        }

        return totalDays;
    }

    /**
     * Function to check if the first date is earlier
than the second date
     */
    public static boolean isEarlier(int day1, int
month1, int year1, int day2, int month2, int
year2) {

        if (year1 < year2) return true;
        if (year1 > year2) {
            System.out.println("Enter correct dates
the second date must be after the first date");
            return false;
        }
        if (month1 < month2) return true;
        if (month1 > month2) {
            System.out.println("Enter correct dates
the second date must be after the first date");
            return false;
        }
        if (day1 < day2) {
            return true;
        }
        if (day1 > day2) {
            System.out.println("Enter correct dates
the second date must be after the first date");
            return false;
        }
    }

    public void viewProducts() {
        print(products, "registered products at Storage");
    }

    public Map<String, Map<String, String>> regProd()
    {
        return products;
    }

    public void print(Map<String, Map<String, String>>
items, String whichItem) {

System.out.println("----------------------------------------------
-----");
        System.out.println(whichItem);
        for (Map.Entry<String, Map<String, String>> itms :
items.entrySet()) {
            System.out.println(itms.getKey() + ":");
            for (Map.Entry<String, String> sitm :
itms.getValue().entrySet()) {
                System.out.printf("%-30s %s%n",
sitm.getKey(), sitm.getValue());
            }
        }

System.out.println("----------------------------------------------
-----");
    }

}
public class SalesCommand {
    Map<String, Runnable> commands = new
HashMap<>();
    SalesController sc = new BasicSalesController();

    public SalesCommand() {
        setCommands();
    }

    private void setCommands() {
        Scanner scan = new Scanner(System.in);

        commands.put("Register Retailer", () -> {
            System.out.println("Enter Retailer Name:");
            String retailerName = scan.nextLine();
            System.out.println("Enter Retailer Location:");
            String retailerLocation = scan.nextLine();
            sc.registerRetailer(retailerName,
retailerLocation);
        });

        commands.put("Add Order", () -> {
            System.out.println("Enter Retailer id:");
            int rid = scan.nextInt();
            scan.nextLine();
            System.out.println("Enter the date of order in the
format DD-MM-YYYY:");
            String date = scan.nextLine();

            Map<String, Integer> products = new
HashMap<>();
            boolean flag = true;
            while (flag) {
                System.out.println("Enter product name:");
                String pname = scan.nextLine();
                System.out.println("Enter quantity");
                int quan = scan.nextInt();
                scan.nextLine();
                products.put(pname, quan);

                System.out.println("Do you want to still add
(yes/no):");
                String response = scan.nextLine();

                if (response.equals("no")) {
                    flag = false;
                }
```

```
        // when the dates are same
        return true;
    }
}

public class RewardRetailer {

    SalesInventoryRequest sir = new
SalesInventoryRequest();
    BasicSalesManage bsm = new
BasicSalesManage();
    DateManagement dm = new
DateManagement();

    public void rewardByOrderId(int rid, int oid,
Map<String, Map<String, String>> orders) {

        String rname = bsm.getRetailerById(rid);
        Map<String, String> odetail =
orders.get(String.valueOf(oid));

        int[] rewardDetail = sir.orderTotal(odetail);


System.out.println("----------------------------------
--------------------------------------------------------------------
---");
        System.out.printf("%40s %n", "Reward
details of " + rname + " associated with " + "
order id " + oid);

System.out.println("----------------------------------
--------------------------------------------------------------------
---");
        System.out.printf("%-30s %-30s %s%n",
"Order Total", "Points Earned", "Points Used");
        System.out.printf("%-30s %-30s %-30s
%s%n", oid, rewardDetail[0], rewardDetail[1],
rewardDetail[2]);

System.out.println("----------------------------------
--------------------------------------------------------------------
---");

    }

    public void rewardByMonth(int rid, int month,
int year, Map<String, Map<String, String>>
orders) {

        int orderCount=0;
        String rname = bsm.getRetailerById(rid);
        int[] rewardDetail;

System.out.println("----------------------------------
--------------------------------------------------------------------
---");
        System.out.printf("%40s %n", "Reward
details of " + rname + " in " +month +" month "+
year + " year");

System.out.println("----------------------------------
--------------------------------------------------------------------
---");

        for (Map.Entry<String, Map<String,
String>> order : orders.entrySet()) {
            String oid = order.getKey();
            Map<String, String> odetail =
order.getValue();
            int ridInOrder =
Integer.parseInt(odetail.get("retailerId"));

            if (rid == ridInOrder) {
                if (dm.isInMonth(odetail.get("date"),
month, year)) {
                    ++orderCount;
                    rewardDetail =
                }
                sc.addOrder(rid, date, products);
            });
            commands.put("Print Receipt", () -> {
                System.out.println("Enter Order id:");
                int oid = scan.nextInt();
                scan.nextLine();
                sc.printReceipt(oid);
            });
            commands.put("Change Price", () -> {
                System.out.println("Enter Product Name:");
                String pname = scan.nextLine();
                System.out.println("Enter new price");
                int price = scan.nextInt();
                scan.nextLine();

                sc.changePrice(price, pname);
            });
            commands.put("View Retailers", () -> {
                sc.viewRetailers();
            });
            commands.put("View Orders", () -> {
                sc.viewOrders();
            });
            commands.put("View Registered Products", () -> {
                sc.viewRegisteredProducts();
            });
            commands.put("View Available Products", () -> {
                sc.viewAvaProducts();
            });
            commands.put("Return Products", () -> {
                System.out.println("Enter Retailer id:");
                int rid = scan.nextInt();
                scan.nextLine();
                System.out.println("Enter order id:");
                int oid = scan.nextInt();
                scan.nextLine();
                System.out.println("How many days has it been
since the purchase? :");
                int days = scan.nextInt();
                scan.nextLine();
                Map<String, Integer> products = new
HashMap<>();
                boolean flag = true;
                while (flag) {
                    System.out.println("Enter product name:");
                    String pname = scan.nextLine();
                    System.out.println("Enter quantity");
                    int quan = scan.nextInt();
                    scan.nextLine();
                    products.put(pname, quan);

                    System.out.println("Do you want to still add
(yes/no):");
                    String response = scan.nextLine();

                    if (response.equals("no")) {
                        flag = false;
                    }
                }
                sc.returnOrder(rid, oid, products, days);
            });

            commands.put("NumOfProductSold", () -> {
                System.out.println("Enter the product Name :");
                String pname = scan.nextLine();
                sc.NumOfProductSold(pname);

            });
            commands.put("NumOfProductInDay", () -> {
                System.out.println("Enter the product Name :");
                String pname = scan.nextLine();
                System.out.println("Enter the Date
(DD-MM-YYYY) :");
                String date = scan.nextLine();
                sc.NumOfProductInDay(pname,date);
            });
            commands.put("NumOfProductInMonth", () -> {
                System.out.println("Enter the product Name :");
```

```
sir.orderTotal(odetail);
                System.out.printf("%-30s %-30s
%-30s %s%n", "Order ID", "Order Total", "Points
Earned", "Points Used");
                System.out.printf("%-30s %-30s
%-30s %s%n", oid, rewardDetail[0],
rewardDetail[1], rewardDetail[2]);
            }
        }
    }
    if(orderCount == 0)
    {
        System.out.println(rname+" does not
placed any orders in " +month +" month "+ year
+ " year");
    }

System.out.println("----------------------------------------
--------------------------------------------------------------------
---");


  }

  public void rewardByYear(int rid, int year,
Map<String, Map<String, String>> orders) {
    int orderCount=0;
    String rname = bsm.getRetailerById(rid);
    int[] rewardDetail;

System.out.println("----------------------------------------
--------------------------------------------------------------------
---");
    System.out.printf("%40s %n", "Reward
details of " + rname + " in " + year + " year");

System.out.println("----------------------------------------
--------------------------------------------------------------------
---");

    for (Map.Entry<String, Map<String,
String>> order : orders.entrySet()) {
        String oid = order.getKey();
        Map<String, String> odetail =
order.getValue();
        int ridInOrder =
Integer.parseInt(odetail.get("retailerId"));

        if (rid == ridInOrder) {
            if (dm.isInYear(odetail.get("date"),
year)) {
                ++orderCount;
                rewardDetail =
sir.orderTotal(odetail);
                System.out.printf("%-30s %-30s
%-30s %s%n", "Order ID", "Order Total", "Points
Earned", "Points Used");
                System.out.printf("%-30s %-30s
%-30s %s%n", oid, rewardDetail[0],
rewardDetail[1], rewardDetail[2]);
            }
        }
    }
    if(orderCount == 0)
    {
        System.out.println(rname+" does not
placed any orders in " + year + " year");
    }

System.out.println("----------------------------------------
--------------------------------------------------------------------
---");

  }

}
```

```
        String pname = scan.nextLine();
        System.out.println("Enter the Month(MM) :");
        int month = scan.nextInt();
        scan.nextLine();
        System.out.println("Enter the Year(YYYY) :");
        int year = scan.nextInt();
        scan.nextLine();
        sc.NumOfProductInMonth(pname,month,year);
      });
      commands.put("NumOfProductInYear", () -> {
        System.out.println("Enter the product Name :");
        String pname = scan.nextLine();
        System.out.println("Enter the Year(YYYY) :");
        int year = scan.nextInt();
        scan.nextLine();
        sc.NumOfProductInYear(pname,year);
      });
      commands.put("MonthSaleReport", () -> {
        System.out.println("Enter the Month(MM) :");
        int month = scan.nextInt();
        scan.nextLine();
        System.out.println("Enter the Year(YYYY) :");
        int year = scan.nextInt();
        scan.nextLine();
        sc.MonthSaleReport(month,year);
      });
      commands.put("DaySaleReport", () -> {
        System.out.println("Enter the Date
(DD-MM-YYYY) :");
        String date = scan.nextLine();
        sc.DaySaleReport(date);
      });
      commands.put("YearSaleReport", () -> {
        System.out.println("Enter the Year(YYYY) :");
        int year = scan.nextInt();
        scan.nextLine();
        sc.YearSaleReport(year);
      });
      commands.put("priceManipulate", () -> {
        System.out.println("Enter the product Name :");
        String pname = scan.nextLine();
        sc.priceManipulate(pname);
      });
      commands.put("trackRetailerReward", () -> {
        System.out.println("Enter the Retailer Id :");
        int rid = scan.nextInt();
        sc.trackRetailerReward(rid);
      });

      commands.put("AvailablePoints",() ->{
        System.out.println("Enter the Retailer Id :");
        int rid = scan.nextInt();
        sc.getRetailerPoints(rid);
      });

      commands.put("help", () -> {
        System.out.println("Available Commands:");
        System.out.printf("%-25s - %s\n", "'Register
Retailer'", "Register a new retailer.");
        System.out.printf("%-25s - %s\n", "'Add Order'",
"Add a new order.");
        System.out.printf("%-25s - %s\n", "'Print
Receipt'", "Print a receipt for an order.");
        System.out.printf("%-25s - %s\n", "'Change
Price'", "Change the price of a product.");
        System.out.printf("%-25s - %s\n", "'View
Retailers'", "View the list of registered retailers.");
        System.out.printf("%-25s - %s\n", "'View
Registered Products'", "View all products registered in
the system.");
        System.out.printf("%-25s - %s\n", "'View
Available Products'", "View all currently available
products.");
        System.out.printf("%-25s - %s\n", "'View
Orders'", "View the list of orders.");
        System.out.printf("%-25s - %s\n", "'Return
Products'", "Handle product returns.");
        System.out.printf("%-25s - %s\n",
"'NumOfProductSold'", "Finds the specified product sold
```

```
count across all sales");
            System.out.printf("%-25s - %s\n",
"'NumOfProductInDay'", "Finds the specified product
sold count on particular day");
            System.out.printf("%-25s - %s\n",
"'NumOfProductInMonth'", "Finds the specified product
sold count in a particular month");
            System.out.printf("%-25s - %s\n",
"'NumOfProductInYear'", "Finds the specified product
sold count in a particular year");
            System.out.printf("%-25s - %s\n",
"'MonthSaleReport'", "Generates sale report for a
particular month");
            System.out.printf("%-25s - %s\n",
"'DaySaleReport'", "Generates sale report for a
particular day");
            System.out.printf("%-25s - %s\n",
"'YearSaleReport'", "Generates sale report for a
particular year");
            System.out.printf("%-25s - %s\n",
"'priceManipulate'", "Gives suggestion on price
manipulation of a product");
            System.out.printf("%-25s - %s\n",
"'trackRetailerReward'", "Provides retailer reward history
on different scales");
            System.out.printf("%-25s - %s\n",
"'AvailablePoints'", "Returns the available reward points
of a retailer");
        });

        commands.put("exit", () -> {
            try {
                App.prompt();
            } catch (Exception e) {

            }
        });
    }

    public void perform(String command) {
        Runnable action =
commands.get(command.trim());
        if (action != null) {
            action.run();
        } else {
            System.out.println("Unknown command: " +
command);   }}}
```